Dual-Pivot Quicksort and Beyond

Analysis of Multiway Partitioning and Its Practical Potential

Sebastian Wild

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Vortrag zur wissenschaftlichen Aussprache
im Rahmen des Promotionsverfahrens

8. Juli 2016

Dual-P??? Q?????? and Beyond

Analysis of Multi??? P??????? and Its Practical Potential

Sebastian Wild

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Vortrag zur wissenschaftlichen Aussprache
im Rahmen des Promotionsverfahrens

8. Juli 2016

Dual-Pivot Quicksort and Beyond

Analysis of Multiway Partitioning and Its Practical Potential

Sebastian Wild

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Vortrag zur wissenschaftlichen Aussprache
im Rahmen des Promotionsverfahrens

8. Juli 2016

What is sorting?

What do you think when you hear **sorting**?

What is sorting?

What do you think when you hear **sorting**?



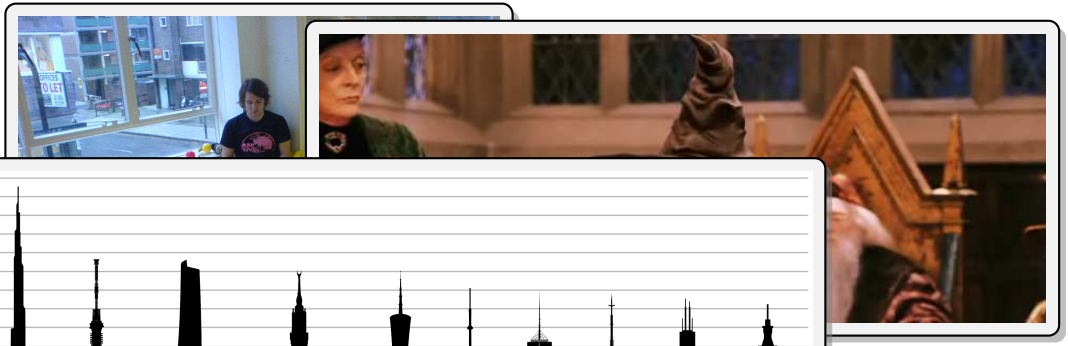
What is sorting?

What do you think when you hear **sorting**?



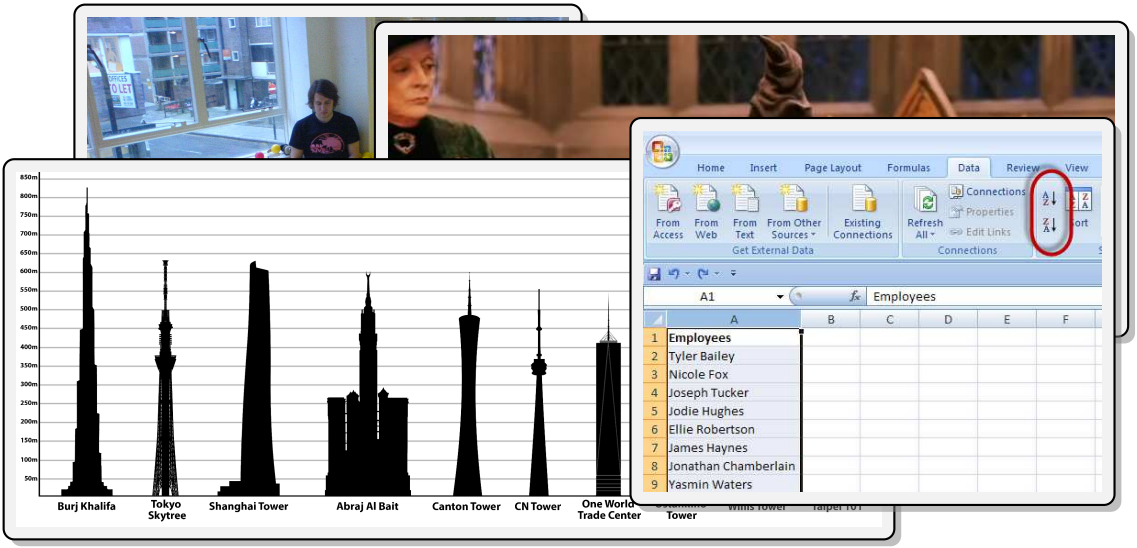
What is sorting?

What do you think when you hear **sorting**?



What is sorting?

What do you think when you hear **sorting**?



What is sorting?

⇝ Two meanings of **sorting**

1. Separate different **sorts** of things

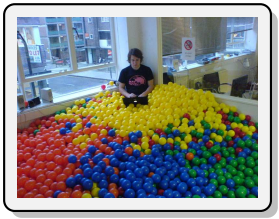
2. Put items into **order** (alphabetical, numerical, . . .)

What is sorting?

⇝ Two meanings of **sorting**

1. Separate different **sorts** of things





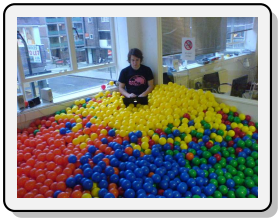
2. Put items into **order** (alphabetical, numerical, . . .)

What is sorting?

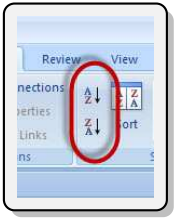
⇝ Two meanings of **sorting**

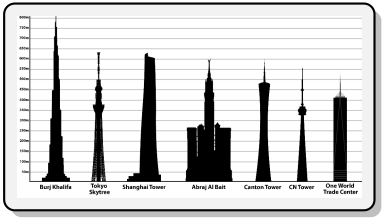
1. Separate different **sorts** of things





2. Put items into **order** (alphabetical, numerical, . . .)



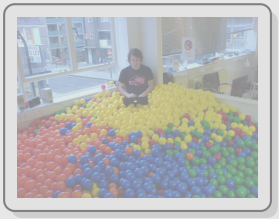


What is sorting?

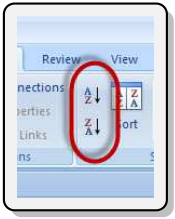
⇝ Two meanings of **sorting**

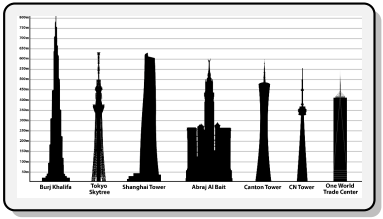
1. Separate different **sorts** of things





2. Put items into **order** (alphabetical, numerical, . . .)





Model of Computation

How do computers sort?

1	2	3	4	5	6	7	8	9
50	2	75	17	10	78	33	72	98

Input given as **array** of n items

- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?

1	2	3	4	5	6	7	8	9
50	2	75	17	10	78	33	72	98

Input given as **array** of n items

- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e. g., numbers $1, \dots, n$

Model of Computation

How do computers sort?

1	2	3	4	5	6	7	8	9
50	2	75	17	10	78	33	72	98

e.g. Excel rows

Input given as **array** of n items

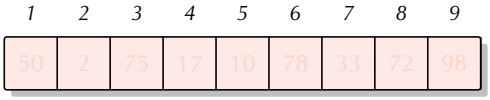
- index-based access (5th item $\mathtt{A}[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $\mathtt{A}[5] < \mathtt{A}[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

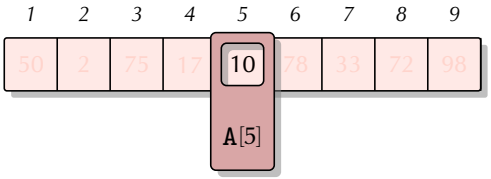
- index-based access (5th item $\mathtt{A}[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $\mathtt{A}[5] < \mathtt{A}[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

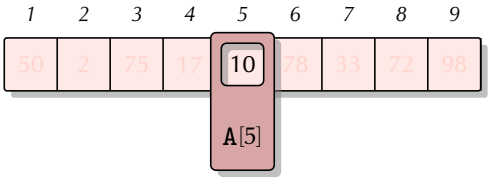
- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

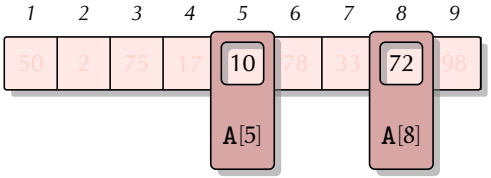
- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

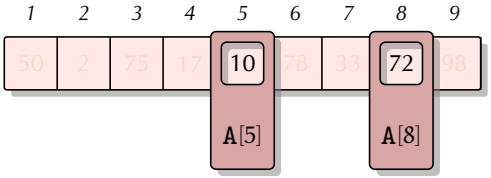
- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

- index-based access (5th item $A[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $A[5] < A[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?

1	2	3	4	5	6	7	8	9
50	2	75	17	10	78	33	72	98

e.g. Excel rows

Input given as **array** of n items

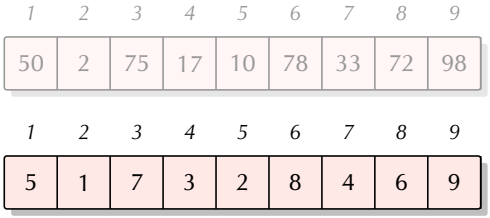
- index-based access (5th item $\mathtt{A}[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $\mathtt{A}[5] < \mathtt{A}[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

Model of Computation

How do computers sort?



e.g. Excel rows

Input given as **array** of n items

- index-based access (5th item $\mathtt{A}[5]$ is 10)
- we can read and write cells
- Only info about data: **pairwise comparisons**
  Is $\mathtt{A}[5] < \mathtt{A}[8]$? Yes, $10 < 72$

$\rightsquigarrow$ only relative ranking of elements important

$\rightsquigarrow$ may assume, e.g., numbers $1, \dots, n$

What is Quicksort?

How to make a computer sort?

Selectionsort

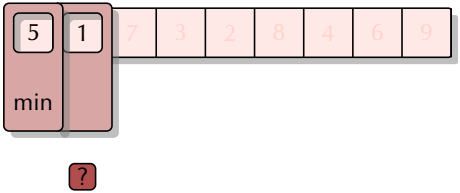
Find smallest item and put it in place.

What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

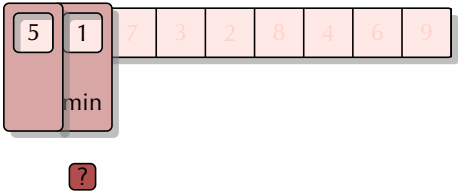


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

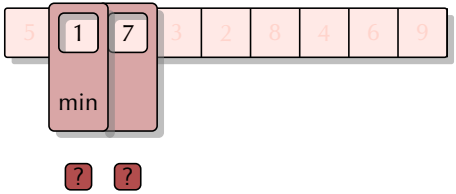


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

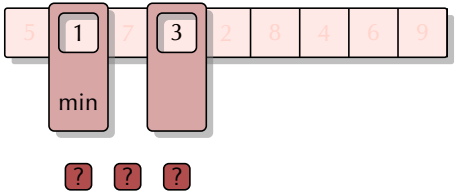


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

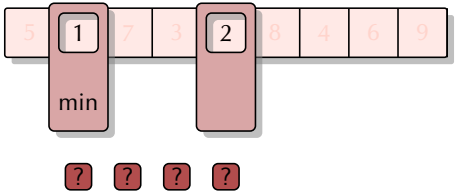


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

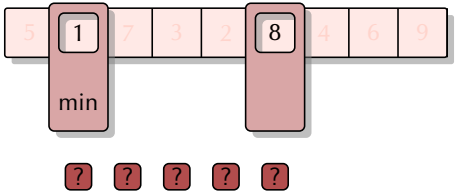


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

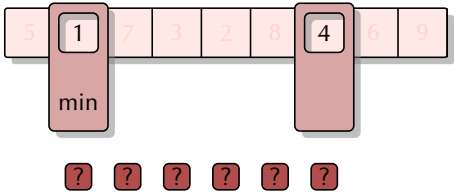


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

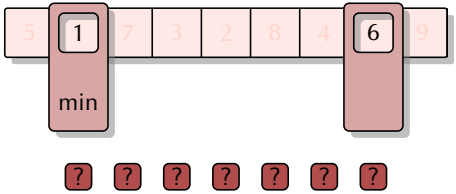


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

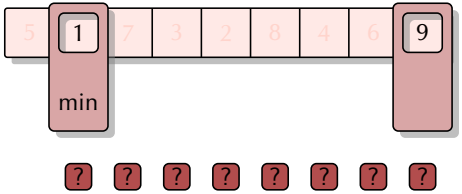


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

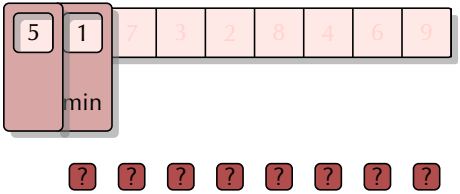


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

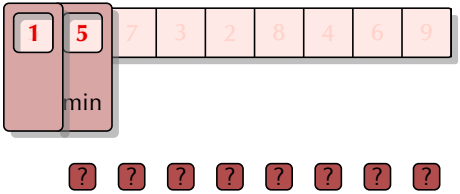


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

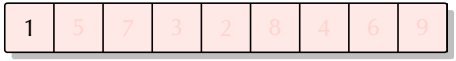


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



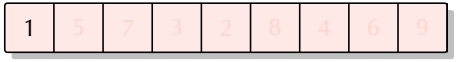


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



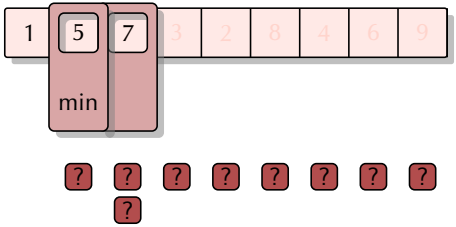


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

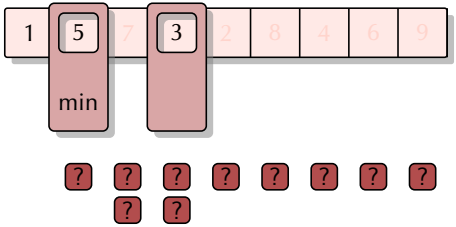


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

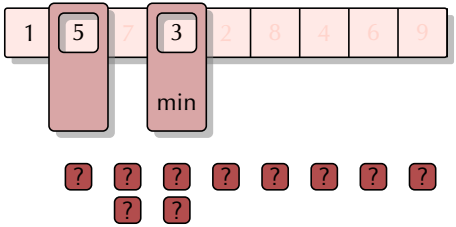


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

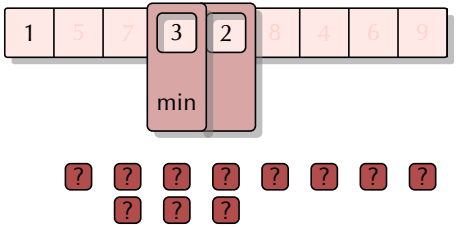


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

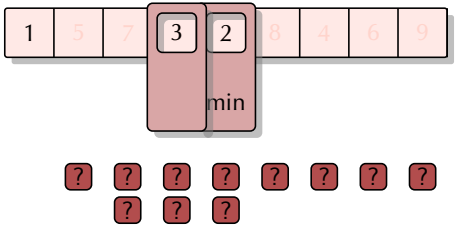


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

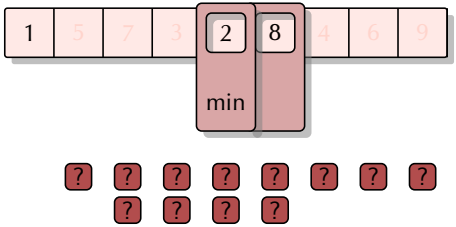


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

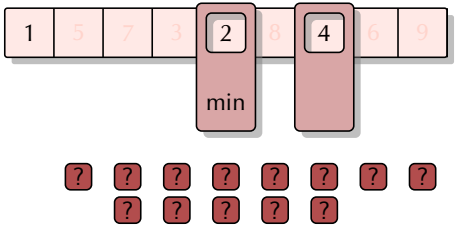


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

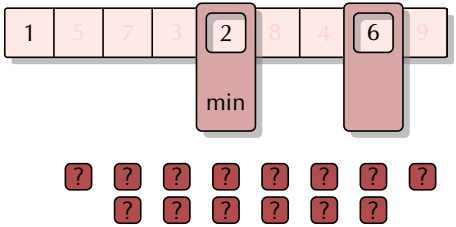


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

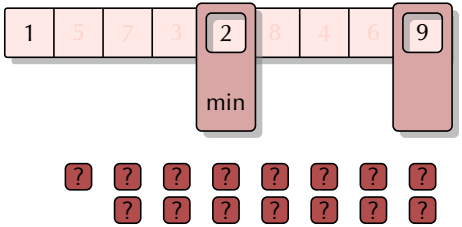


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

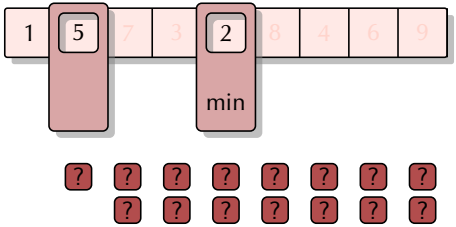


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

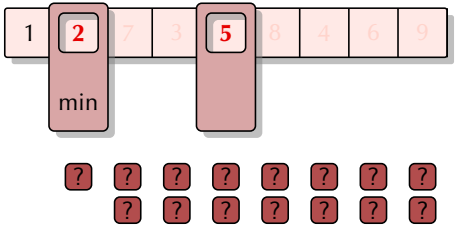


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.

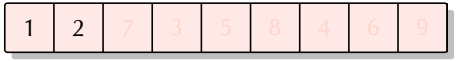


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



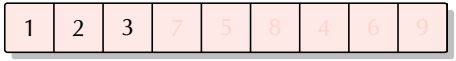


What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



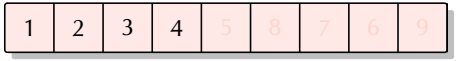


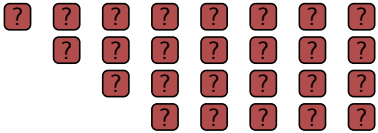
What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.





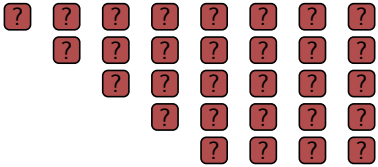
What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.





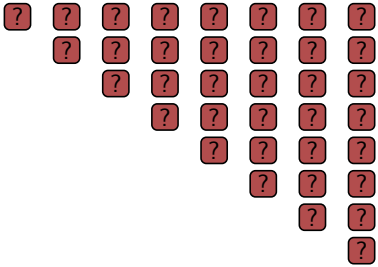
What is Quicksort?

How to make a computer sort?

Selectionsort

Find smallest item and put it in place.





What is Quicksort?

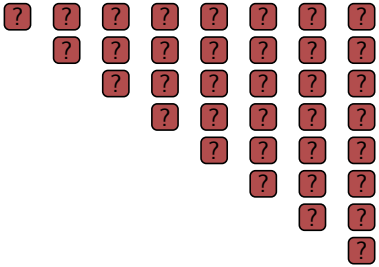
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



What is Quicksort?

How to make a computer sort?

Selectionsort

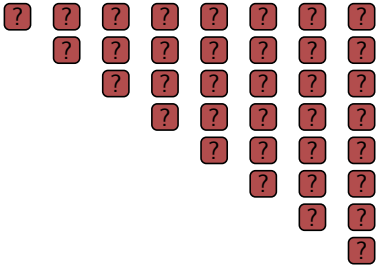
Find smallest item and put it in place.



Each round puts one item in place.
$\rightsquigarrow$ Remaining list **one** smaller.

Can we do better?

36 Comparisons



What is Quicksort?

How to make a computer sort?

Selectionsort

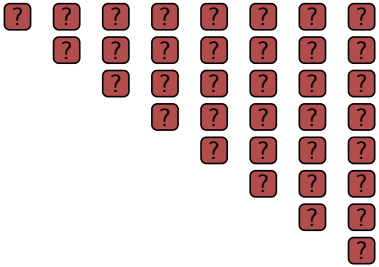
Find smallest item and put it in place.



Each round puts one item in place.
$\rightsquigarrow$ Remaining list **one** smaller.

Can we do better?

36 Comparisons



What is Quicksort?

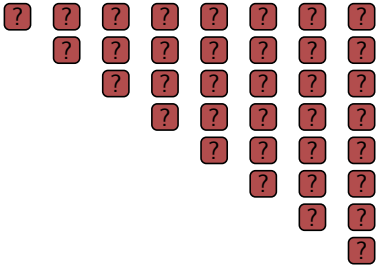
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



Each round puts one item in place.
$\rightsquigarrow$ Remaining list **one** smaller.

Can we do better?

Yes! Finding the minimum costs the same as computing the **rank** of **any** element.

number of elements
smaller than given element

What is Quicksort?

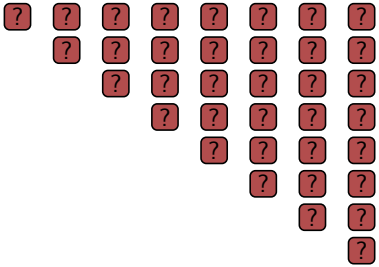
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



What is Quicksort?

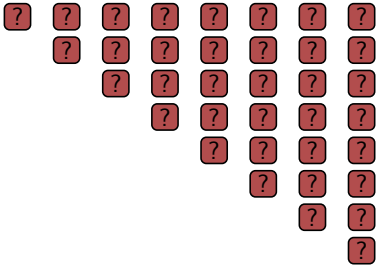
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).

What is Quicksort?

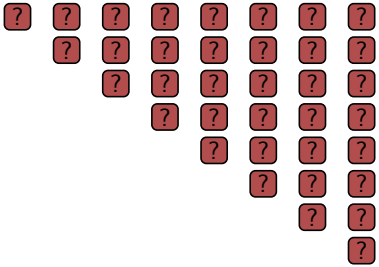
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).

What is Quicksort?

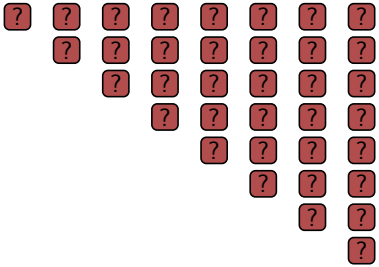
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

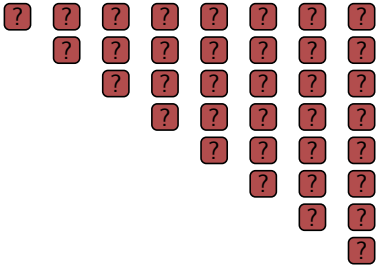
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

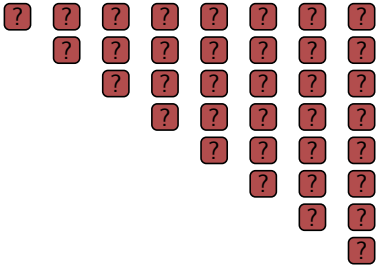
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



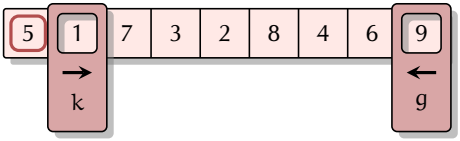
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

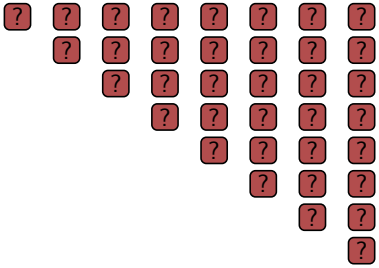
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



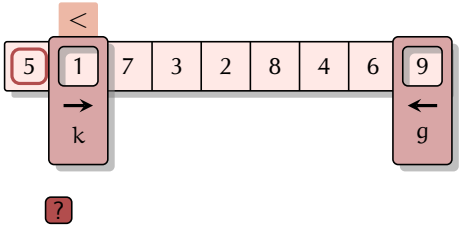
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

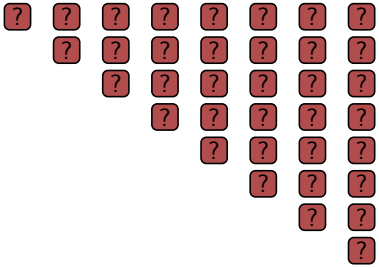
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



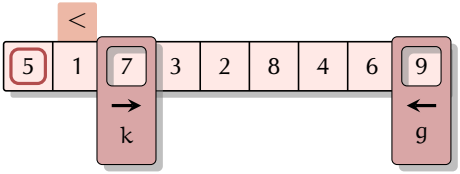
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

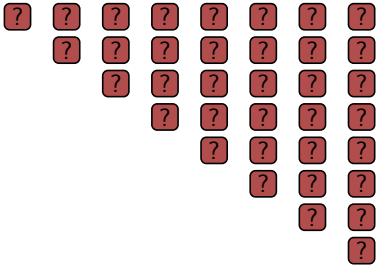
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



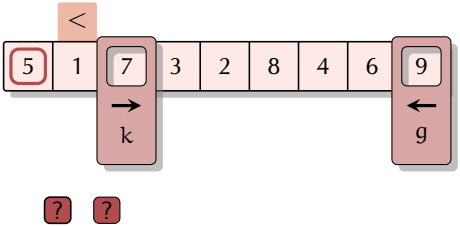
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

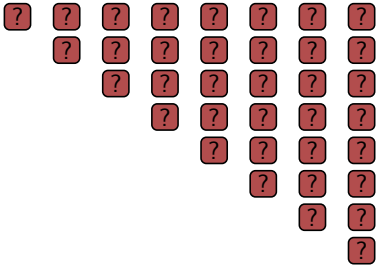
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



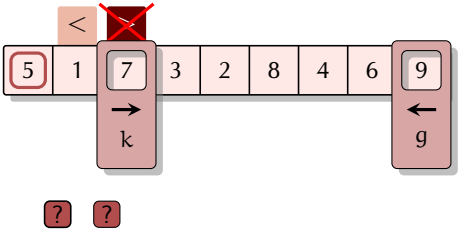
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

How to make a computer sort?

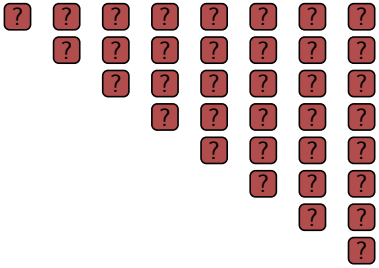
kth smallest $\rightsquigarrow$ $A[k]$

Selectionsort

Find smallest item and put it in place.

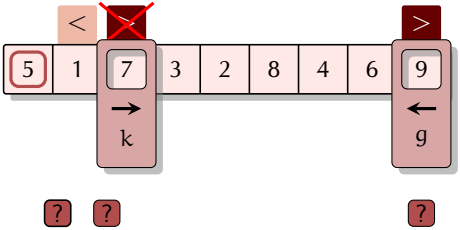


36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

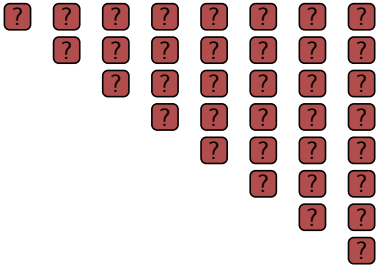
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



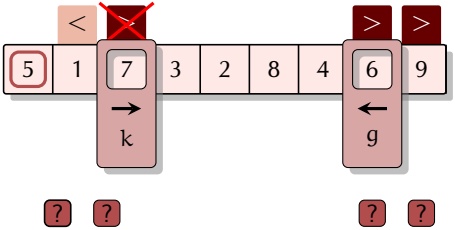
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

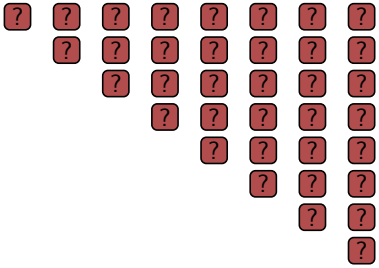
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



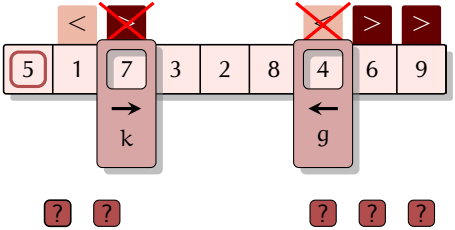
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

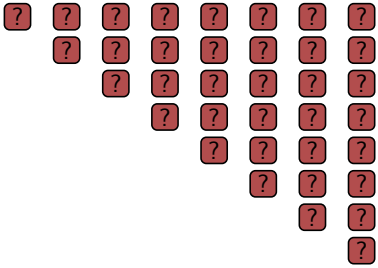
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



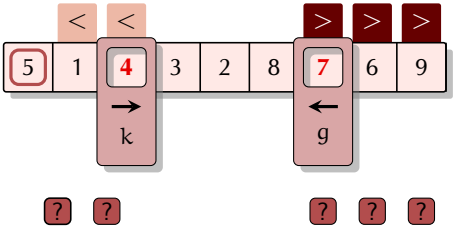
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

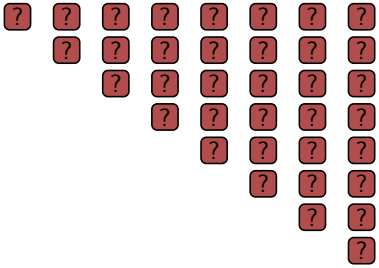
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



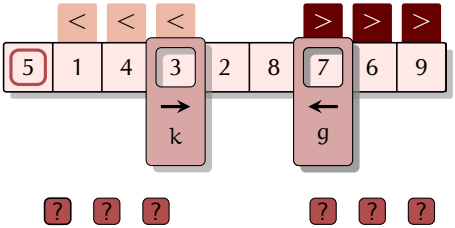
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

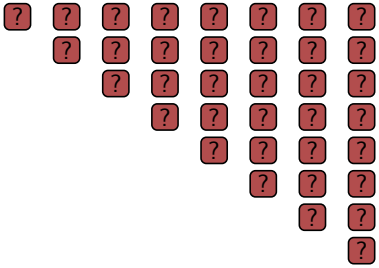
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



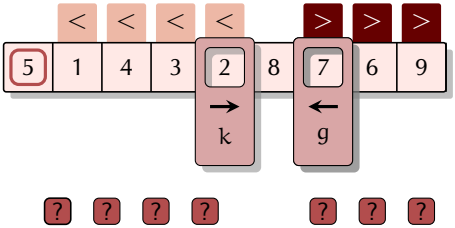
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

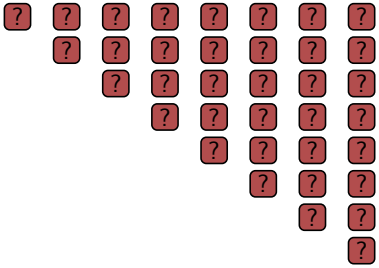
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



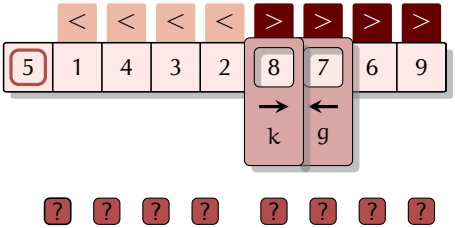
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



What is Quicksort?

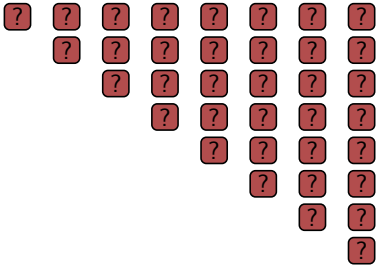
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



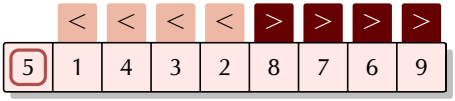
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

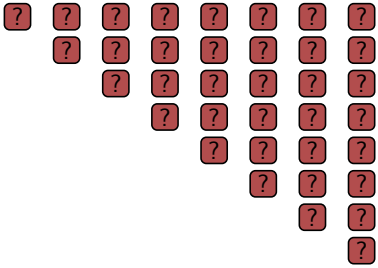
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



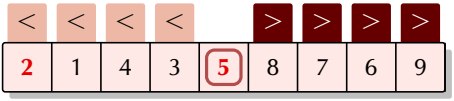
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

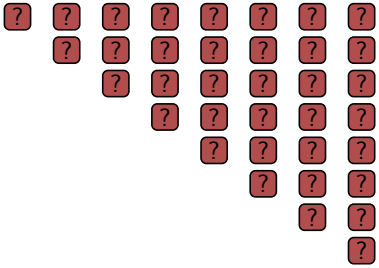
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



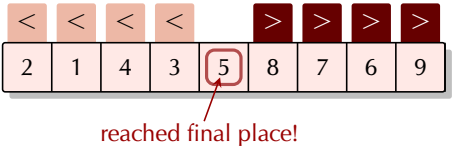
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathbf{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

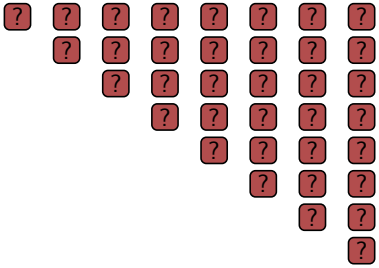
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



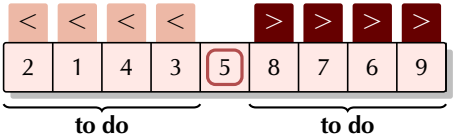
36 Comparisons



kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

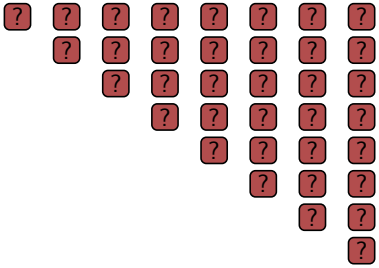
kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Selectionsort

Find smallest item and put it in place.



36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

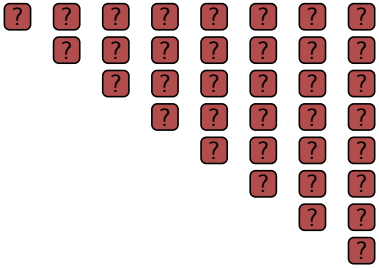
kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Selectionsort

Find smallest item and put it in place.



36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

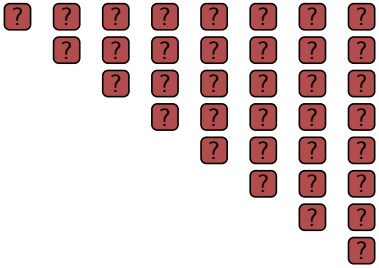
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



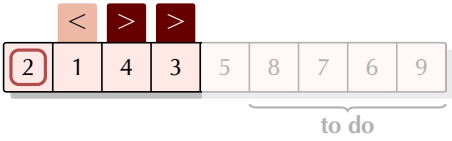
36 Comparisons

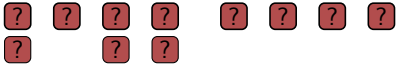


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

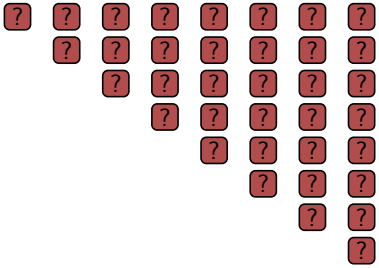
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



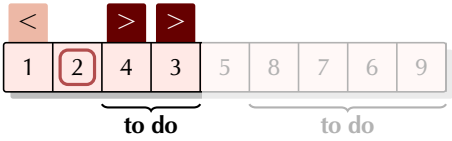
36 Comparisons

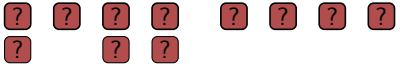


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

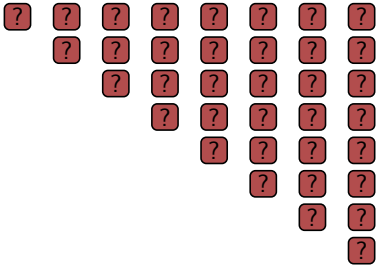
kth smallest $\rightsquigarrow$ A[k]

Selectionsort

Find smallest item and put it in place.

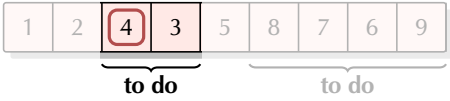


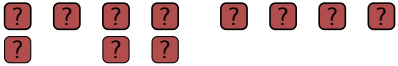
36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

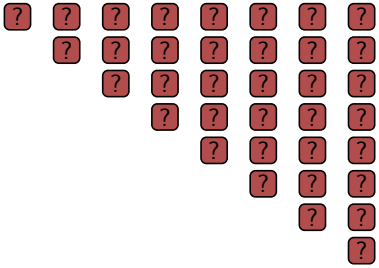
kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Selectionsort

Find smallest item and put it in place.



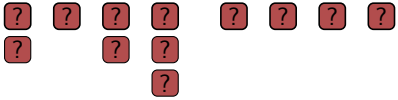
36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

kth smallest $\rightsquigarrow$ A[k]

Selectionsort

Find smallest item and put it in place.

Quicksort

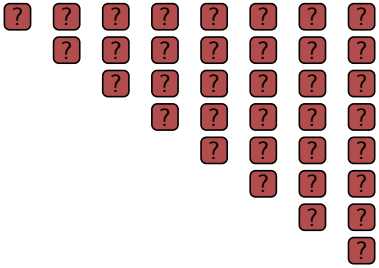
Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).

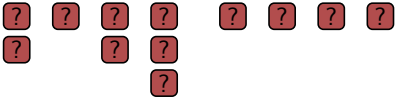




to do

36 Comparisons





What is Quicksort?

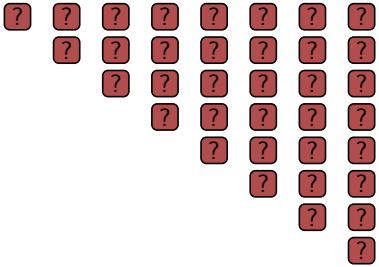
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



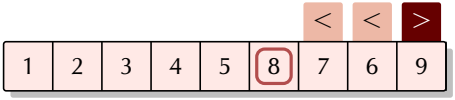
36 Comparisons

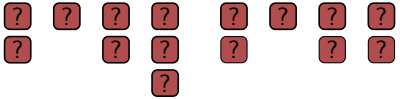


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

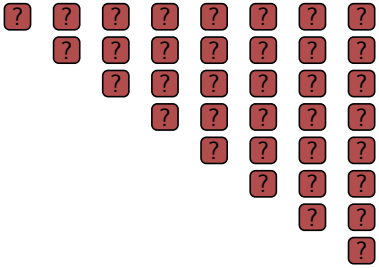
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



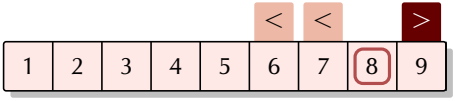
36 Comparisons

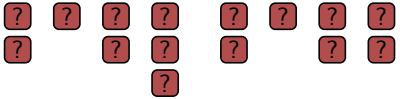


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

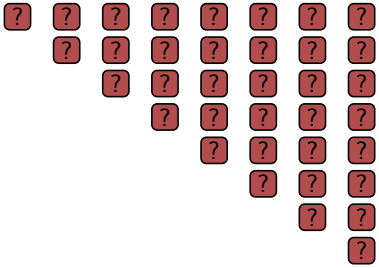
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



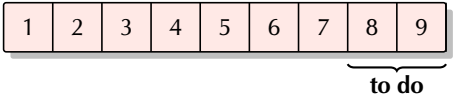
36 Comparisons

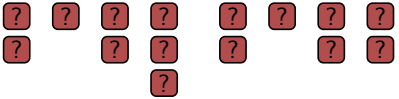


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

How to make a computer sort?

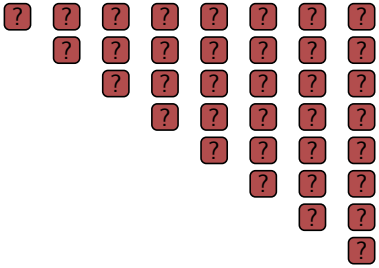
kth smallest $\rightsquigarrow$ $\mathbf{A}[k]$

Selectionsort

Find smallest item and put it in place.

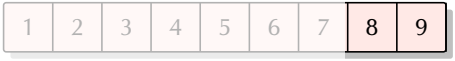


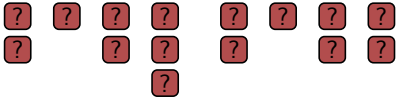
36 Comparisons



Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

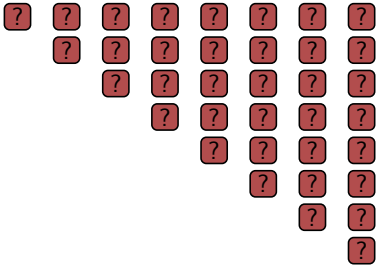
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons

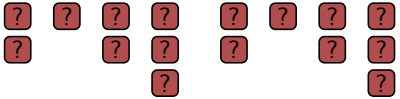


kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).





What is Quicksort?

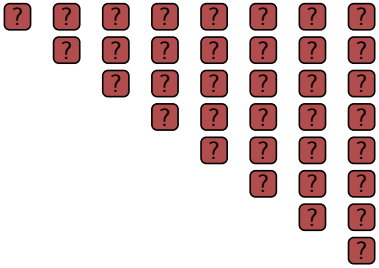
How to make a computer sort?

Selectionsort

Find smallest item and put it in place.



36 Comparisons



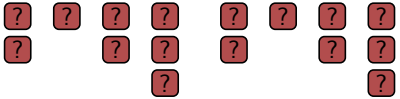
kth smallest $\rightsquigarrow$ $\mathbf{A}[\mathrm{k}]$

Quicksort

Find **final place** of some item *(pivot)*,
put small items left, others right (*partition*).



16 Comparisons



What is Quicksort?

How to make a comput

Selecti

Find smallest item a

rt

item *(pivot)*,

ers right *(partition)*.

1	2	3	4	5	6	7	8	9

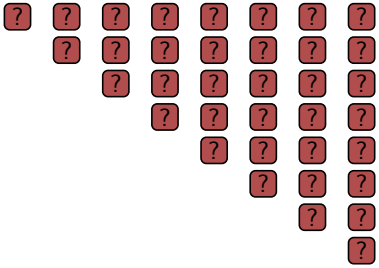
Much **less** comparisons in Quicksort

. . . unless pivots are max/min!

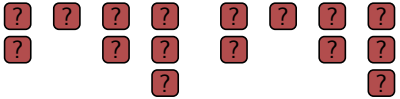
But **average/expected** behavior is good.

⇝ Quicksort is method of choice in practice.

36 Comparisons



16 Comparisons



What is Quicksort?

How to make a comput

Selecti

Find smallest item a

rt

item *(pivot)*,

ers right (*partition*).

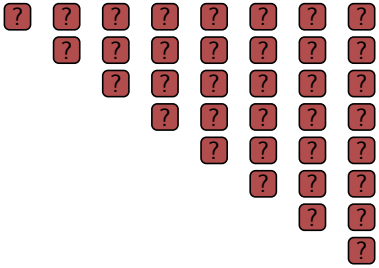
1	2	3	4	5	6	7	8	9

Much **less** comparisons in Quicksort
. . . unless pivots are max/min!

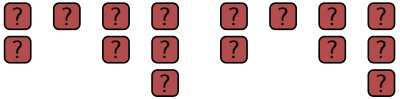
But **average/expected** behavior is good.

⇝ Quicksort is method of choice in practice.

36 Comparisons



16 Comparisons



What is Quicksort?

How to make a comput

Selecti

Find smallest item a

rt

item *(pivot)*,

ers right *(partition)*.

1	2	3	4	5	6	7	8	9

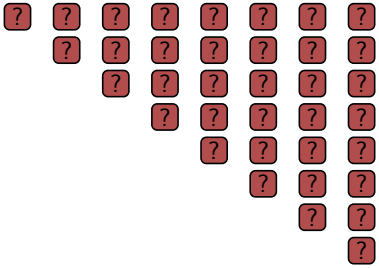
Much **less** comparisons in Quicksort
. . . unless pivots are max/min!

(actually: almost always good)

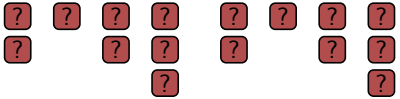
But **average/expected** behavior is good.

⇝ Quicksort is method of choice in practice.

36 Comparisons



16 Comparisons



What is Quicksort?

How to make a comput

Selecti

Find smallest item a

ort

item *(pivot)*,

ers right *(partition)*.

1	2	3	4	5	6	7	8	9

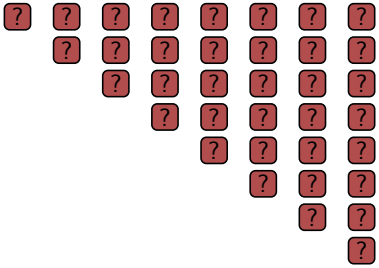
Much **less** comparisons in Quicksort
. . . unless pivots are max/min!

(actually: almost always good)

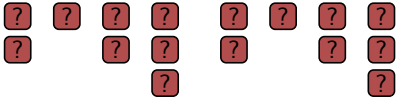
But **average/expected** behavior is good.

$\rightsquigarrow$ Quicksort is method of choice in practice.

36 Comparisons



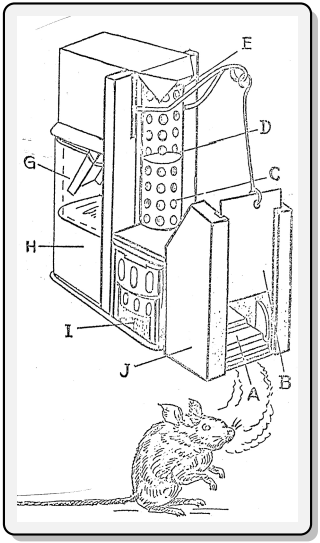
16 Comparisons



Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

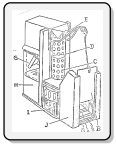
R. Sedgewick and K. Wayne, Algorithms 4th ed.



Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.



Many variants tried

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*

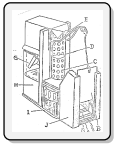
   $\rightsquigarrow$ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots $\rightsquigarrow$ 3 classes: small, medium and large

   $\rightsquigarrow$ subproblems are smaller,
   but partitioning is more expensive.

Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.



Many variants tried

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*

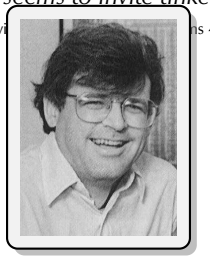
   $\rightsquigarrow$ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots $\rightsquigarrow$ 3 classes: small, medium and large

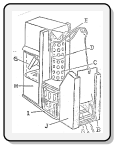
   $\rightsquigarrow$ subproblems are smaller,
   but partitioning is more expensive.

Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewi ns 4th ed.





Many variants tried . . . many **not** helpful

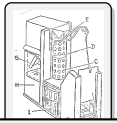
Two important tuning options:

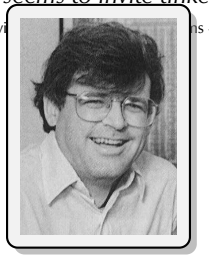
1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.

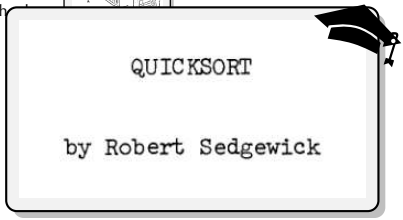
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewi… …s 4th…







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*

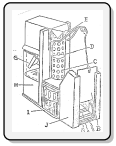
   ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large

   ⇝ subproblems are smaller,
   but partitioning is more expensive.

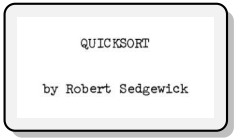
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*

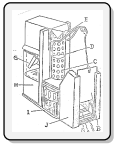
   ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large

   ⇝ subproblems are smaller,
   but partitioning is more expensive.

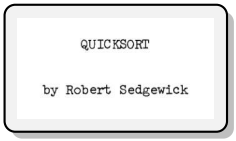
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

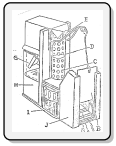
Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.

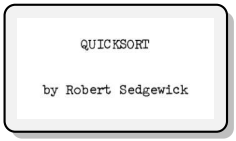
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*

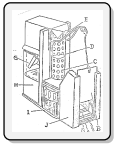
   ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large

   ⇝ subproblems are smaller,
   but partitioning is more expensive.

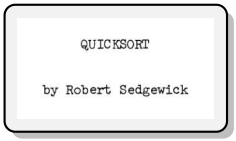
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

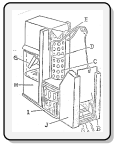
Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.

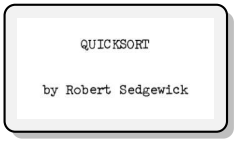
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!

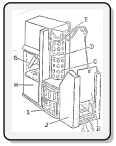


2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.

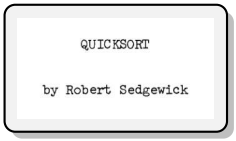
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

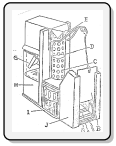
1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.



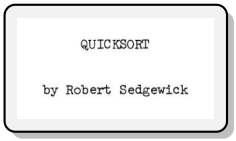
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

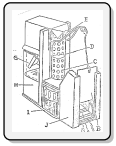
1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.



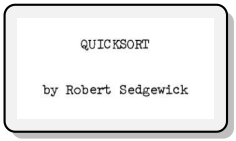
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

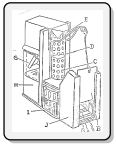
1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - $\rightsquigarrow$ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots $\rightsquigarrow$ 3 classes: small, medium and large
   - $\rightsquigarrow$ subproblems are smaller,
     but partitioning is more expensive.



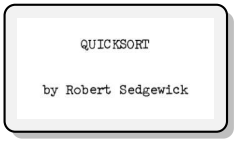
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.



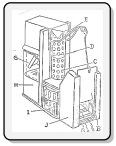


conventional wisdom

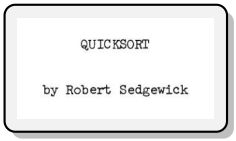
Variants of Quicksort

It is tempting to try to develop ways to improve quicksort:
a faster sorting algorithm is computer science's "better mousetrap,"
and quicksort is a venerable method that seems to invite tinkering.

R. Sedgewick and K. Wayne, Algorithms 4th ed.







Many variants tried … many **not** helpful

Two important tuning options:

1. Choose **pivots** wisely ***(pivot sampling)***
   - for example: middle element of three *(median-of-three)*
   - ⇝ pivot never min or max!
2. Split into $s \geqslant 2$ **subproblems** at once ***(multiway partitioning)***
   - for example: 2 pivots ⇝ 3 classes: small, medium and large
   - ⇝ subproblems are smaller,
     but partitioning is more expensive.





conventional wisdom

till 2009!

Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***

Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***

Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***

Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***

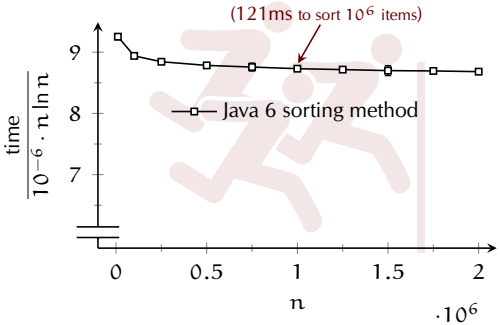
Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***



Normalized Java runtimes (in *ms*).

Average and standard deviation of
1000 random permutations per size.

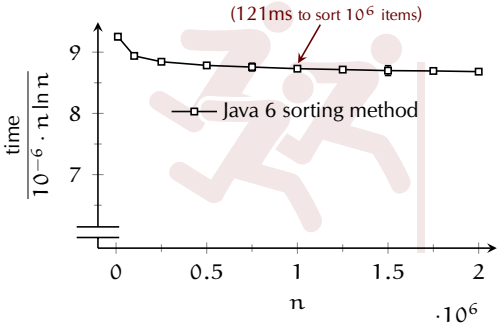
Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***



Normalized Java runtimes (in *ms*).

Average and standard deviation of
1000 random permutations per size.

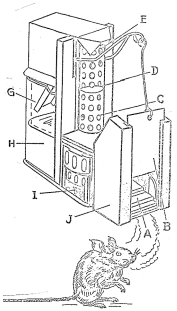
Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

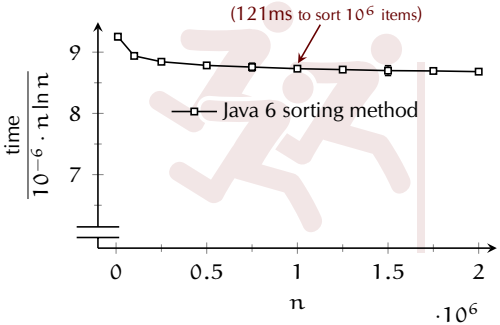
What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***



- started as hobby project by Vladimir Yaroslavskiy
- 20 % faster than highly-tuned Java 6!



Normalized Java runtimes (in *ms*).

Average and standard deviation of 1000 random permutations per size.

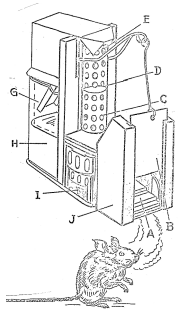
Java 7's Dual-Pivot Quicksort

used e.g. for Android apps

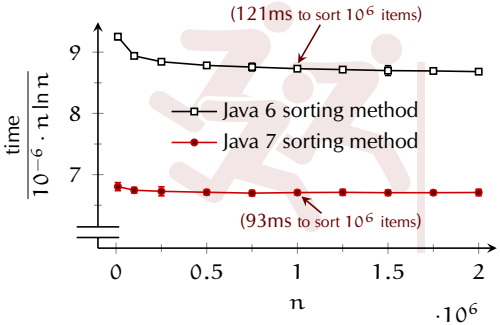
What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***



- started as hobby project by Vladimir Yaroslavskiy
- 20 % faster than highly-tuned Java 6!



Normalized Java runtimes (in *ms*).

Average and standard deviation of 1000 random permutations per size.

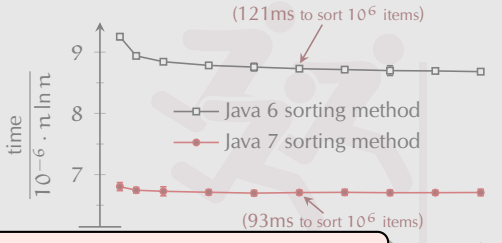
Java 7's Dual-Pivot Quicksort

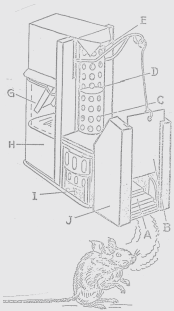
used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***





No theoretical explanation for running time known in 2009!

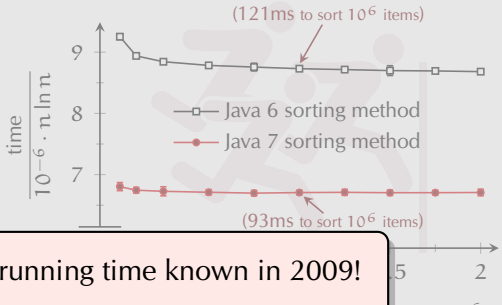
Java 7's Dual-Pivot Quicksort

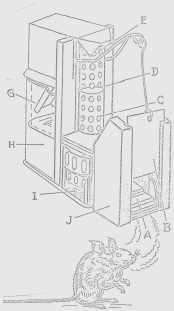
used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***





No theoretical explanation for running time known in 2009!

Only lucky experiments?

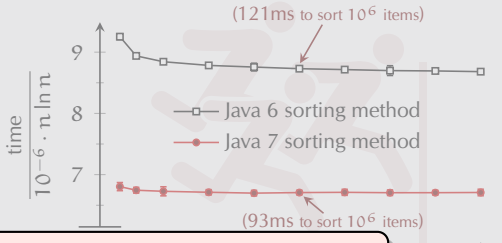
Java 7's Dual-Pivot Quicksort

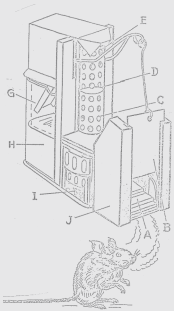
used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***





No theoretical explanation for running time known in 2009!

Only lucky experiments?

Why did no-one come up with this earlier?

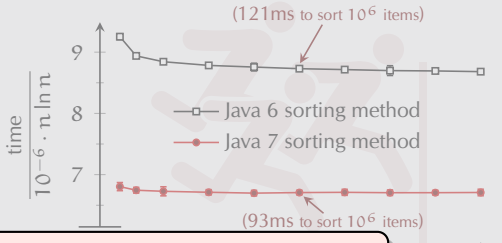
Java 7's Dual-Pivot Quicksort

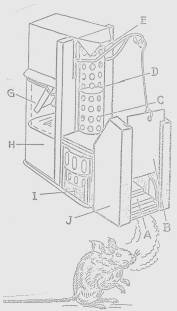
used e.g. for Android apps

What happened in 2009? *Java 7 sorting method was released!*

as in example before
+ median-of-three
+ Insertionsort

- till 2009, Java used ***classic Quicksort***
- Java 7 uses **new** ***dual-pivot Quicksort***





No theoretical explanation for running time known in 2009!

Only lucky experiments?

Why did no-one come up with this earlier?

That was the starting point for my work!

Overview of my work

My Thesis

Multiway partitioning has genuine potential to speed up Quicksort.

Overview of my work

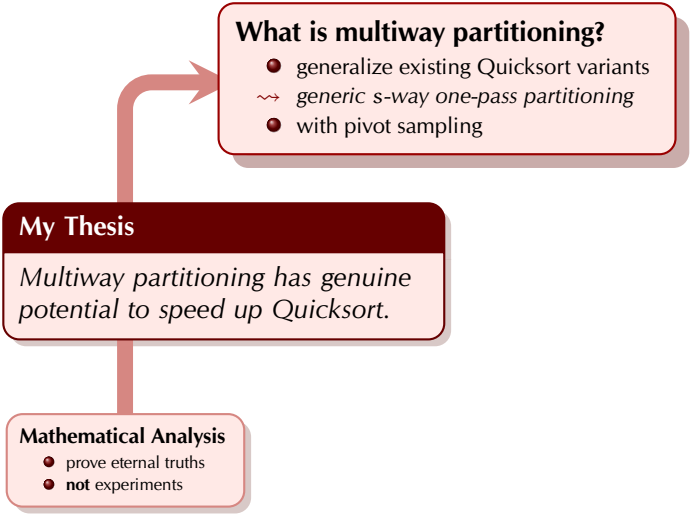
My Thesis

Multiway partitioning has genuine potential to speed up Quicksort.

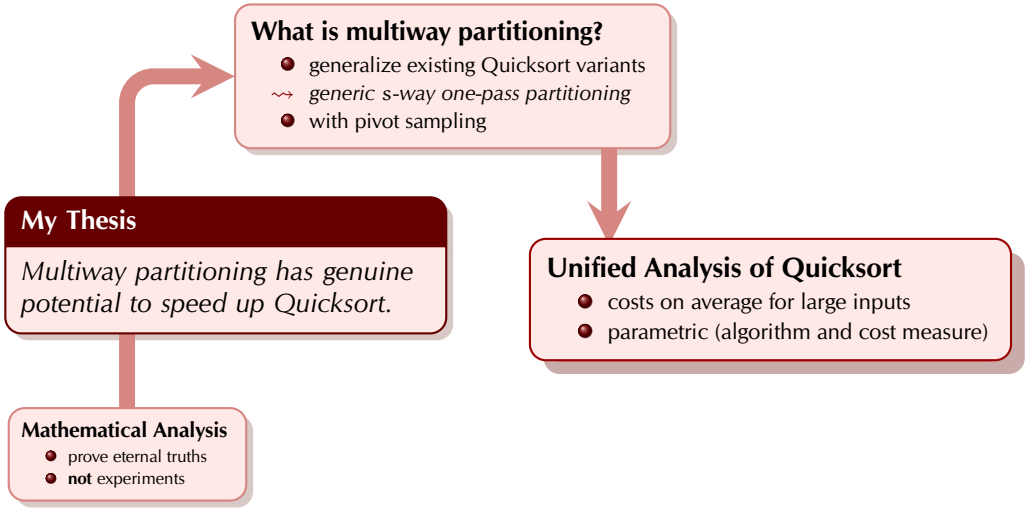
Mathematical Analysis

- prove eternal truths
- **not** experiments

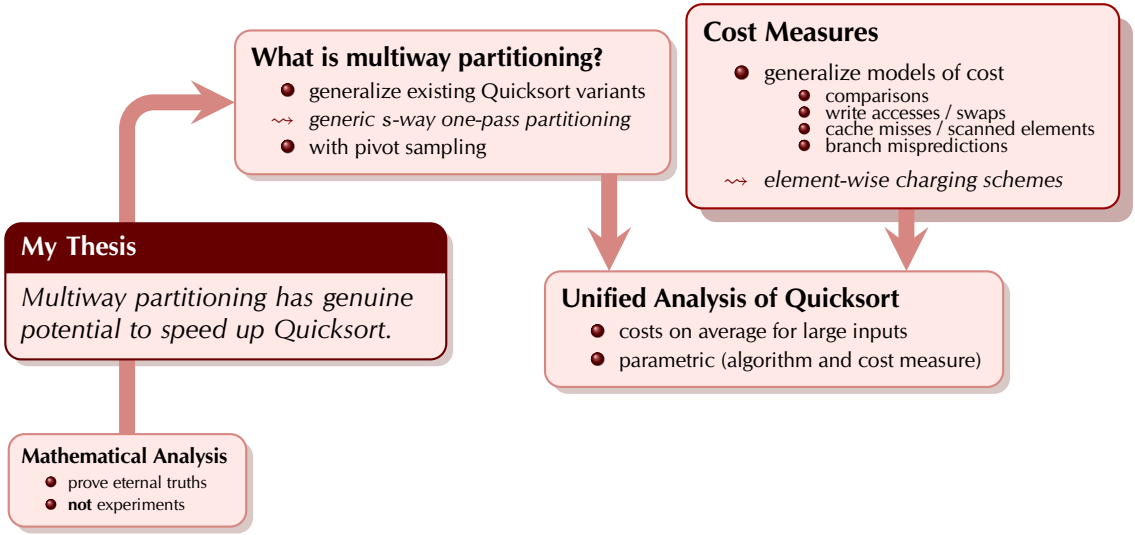
Overview of my work



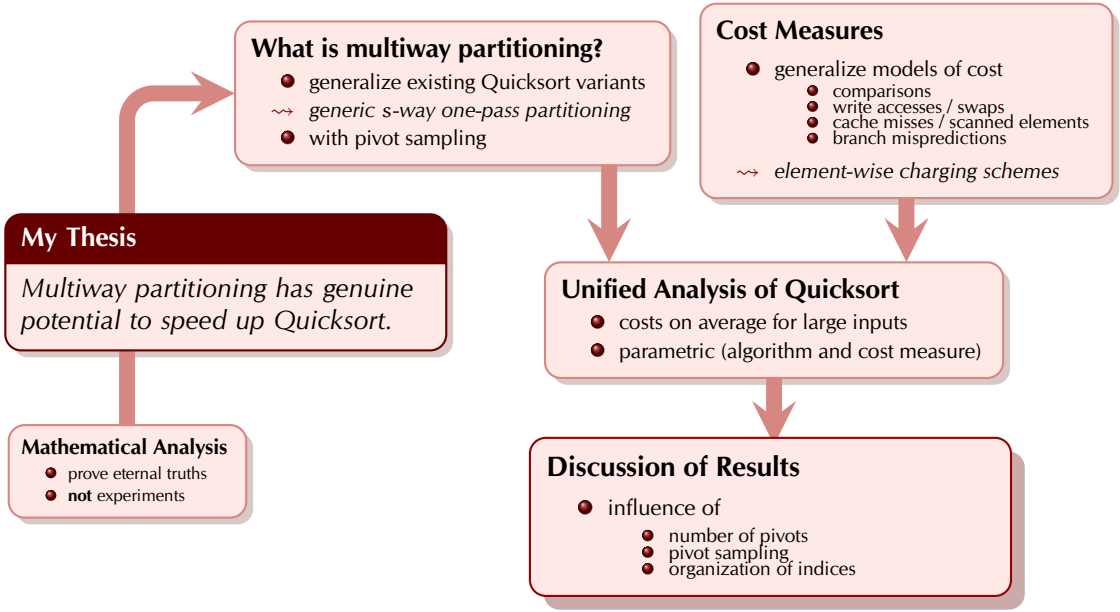
Overview of my work



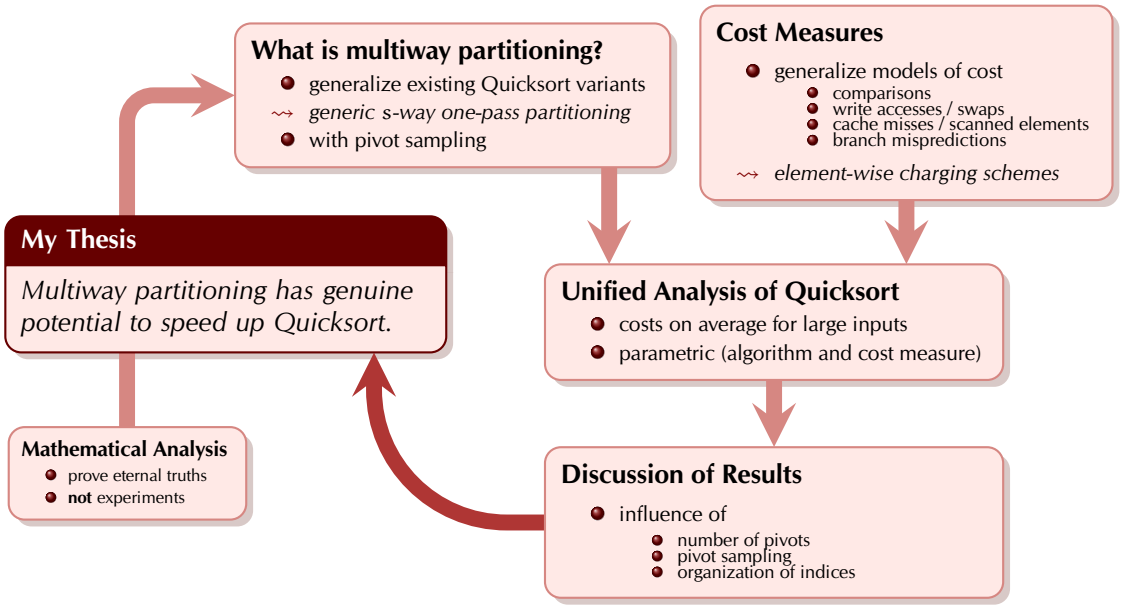
Overview of my work



Overview of my work



Overview of my work



Entropy

What is entropy?

1. **Thermodynamics:** unusable part of thermal energy
2. **Mathematics:** information content of a random variable

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

…all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **Thermodynamics:** unusable part of thermal energy
2. **Mathematics:** information content of a random variable

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

…all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **Thermodynamics:** unusable part of thermal energy
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **~~Thermodynamics:~~** ~~unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **~~Thermodynamics:~~** ~~unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

. . . all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **~~Thermodynamics:~~** ~~unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

…all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. **~~Thermodynamics:~~** ~~unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

…all clear?
Let's see an application.

- Urn with **colored** balls.
- Alice draws ball X uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

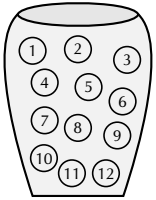
1. **~~Thermodynamics:~~** ~~unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.



- Urn with **colored** balls.
- Alice draws ball $\textcircled{x}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

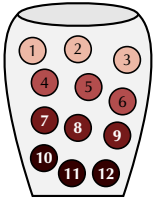
1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.



- Urn with **colored** balls.
- Alice draws ball $\textcircled{x}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

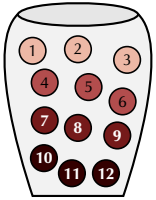
1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.



- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

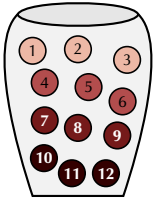
1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u\}$.
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

...all clear?
Let's see an application.



- Urn with **colored** balls.
- Alice draws ball $\textcircled{X}$ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

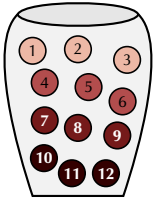
probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u$
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

ion.

$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$



- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

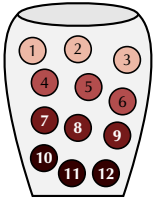
probability that $X = i$

Definition ***(Entropy)***

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u$
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

ion.

$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$



- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

$\rightsquigarrow$

Theorem ***(Shannon's Source Coding Theorem)***

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

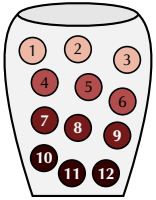
probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u$
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

ion.

$\mathbb{P}[C = ●] = 3/12 = 0.25$
$\mathbb{P}[C = ●] = 3/12 = 0.25$
$\mathbb{P}[C = ●] = 3/12 = 0.25$
$\mathbb{P}[C = ●] = 3/12 = 0.25$



- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

$= 4 \cdot \frac{1}{4} \log_2(4) = 2$

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Entropy

What is entropy?

1. ~~**Thermodynamics:** unusable part of thermal energy~~ (Physicists' party? Next door, please.)
2. **Mathematics:** information content of a random variable

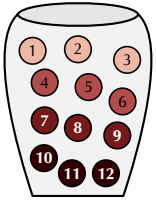
probability that $X = i$

Definition *(Entropy)*

Let X be a random variable with $\mathbb{P}[X = i] = p_i$ for $i \in \{1, \dots, u$
Its *entropy* is defined as $\mathcal{H}(X) = \sum_{i=1}^{u} p_i \log_2(1/p_i)$.

ion.

$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$
$\mathbb{P}[C = \bullet] = 3/12 = 0.25$



- Urn with **colored** balls.
- Alice draws ball Ⓧ uniformly at random from urn.
- Bob wants to **guess its color** C by **Yes/No-Questions**.

⇝ **Theorem** *(Shannon's Source Coding Theorem)*

On average, Bob **can't do with less** than $\mathcal{H}(C)$ questions.

$= 4 \cdot \frac{1}{4} \log_2(4) = 2$

We say: We must acquire $\mathcal{H}(C)$ ***bits of information*** to know C.

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

Theorem *(Information-Theoretic Lower Bound for Sorting)*

$\rightsquigarrow$ We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

$\rightsquigarrow$ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

Theorem *(Information-Theoretic Lower Bound for Sorting)*

We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

⇝ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$ $\quad \widehat{=}$ colors

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

Theorem *(Information-Theoretic Lower Bound for Sorting)*

$\rightsquigarrow$ We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

$\rightsquigarrow$ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$ $\quad \widehat{=}$ colors

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

Theorem *(Information-Theoretic Lower Bound for Sorting)*

$\rightsquigarrow$ We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

$\rightsquigarrow$ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$ $\quad$ $\widehat{=}$ colors

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

Theorem *(Information-Theoretic Lower Bound for Sorting)*

We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

⇝ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$ $\quad$ $\widehat{=}$ colors

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

$\rightsquigarrow$ **Theorem** *(Information-Theoretic Lower Bound for Sorting)*

We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

$\rightsquigarrow$ To achieve this bound, every comparison must yield 1 bit of information!

Information Theory & Sorting

Where is the connection to sorting?



$n(n-1)\cdots 2\cdot 1$ $\quad \widehat{=}$ colors

- Now, Alice draws one of the $n!$ possible **orderings** of n items.
- Bob guesses the order by asking comparisons.

Assumptions:
(1) All orderings equally likely.
(2) Only comparisons allowed.

asymptotic error term for $n \to \infty$

$\rightsquigarrow$ **Theorem** *(Information-Theoretic Lower Bound for Sorting)*

We cannot sort with less than $\log_2(n!) = n\log_2(n) \pm O(n)$ comparisons.

$\mathcal{H}(C)$

We can reverse this idea to analyze Quicksort.

$\rightsquigarrow$ To achieve this bound, every comparison must yield 1 bit of information!

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *. . . stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) + O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *. . . stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) + O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *... stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) + O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *... stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) + O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *... stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) \pm O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$
   What is x? *... stay tuned.*
2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\dfrac{\log_2(n!)}{x} = \dfrac{1}{x} n \log_2(n) \pm O(n)$ comparisons.

Comparisons in Quicksort

How many comparisons does Quicksort use on average?

Entropy-based analysis:

1. Compute the **average information yield**: $x \frac{\text{bit}}{\text{comparison}}$. $(0 < x \leqslant 1)$

   What is x? *... stay tuned.*

2. Need to learn $\log_2(n!)$ bits $\rightsquigarrow$ Use $\frac{\log_2(n!)}{x} = \frac{1}{x} n \log_2(n) \pm O(n)$ comparisons.

not mathematically rigorous ...
but correct result!

rigorous proof $\rightarrow$ dissertation

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P

- Outcome C is either small or large.

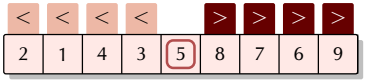
⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

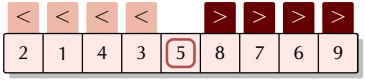
⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

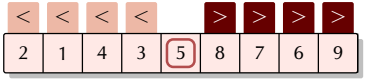
⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

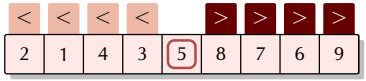
⇝ Information yield is $\mathcal{H}(C) =$
$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0, 1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is X?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

prob. of item to be $< P$

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0, 1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0, 1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. ⇝ random relative ranking

⇝ $\mathbb{P}[<] = P$, $\mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is π?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

We use the **equivalent**

Uniform Model

- n numbers drawn i.i.d. Uniform$(0, 1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is π?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$

$$-\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$$

prob. of item to be $<$ P

Note: pivot value P is **random**.

What is its distribution?

Answer: $P \overset{\mathcal{D}}{=} \text{Uniform}(0,1)$

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

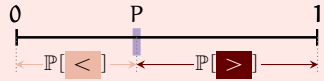
We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow$ $\mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

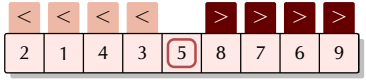


Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big]$$
$$= -2\int_0^1 z\log_2(z)\,dz$$
$$= \frac{1}{2}\log_2(e) \approx 0.721348$$

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

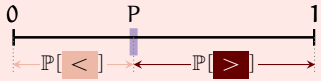
We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

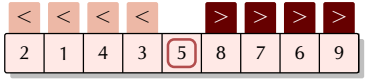


Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big]$$
$$= -2\int_0^1 z\log_2(z)\,dz$$
$$= \frac{1}{2}\log_2(e) \approx 0.721348$$

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

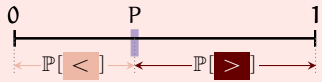
We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

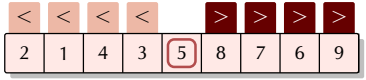


Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$
$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P \log_2(P) - (1-P) \log_2(1-P)\Big]$$
$$= -2 \int_0^1 z \log_2(z)\, dz$$
$$= \frac{1}{2} \log_2(e) \approx 0.721348$$

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

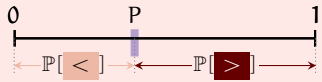
We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

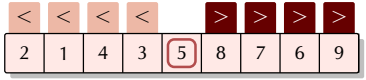


Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

Excursion: *Input Models*

Typical: *Random Permutations*

- items: $1, \dots, n$
- all orderings equally likely

same average sorting costs

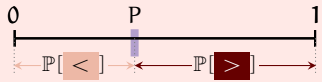
We use the **equivalent**

Uniform Model

independent & identically distributed

- n numbers drawn i.i.d. Uniform$(0,1)$
- i.i.d. $\rightsquigarrow$ random relative ranking

$\rightsquigarrow \mathbb{P}[<] = P, \quad \mathbb{P}[>] = 1 - P$

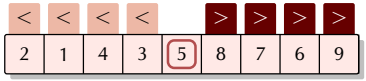


Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) =$
$-\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P \log_2(P) - (1-P) \log_2(1-P)\Big]$$
$$= -2 \int_0^1 z \log_2(z)\, dz$$
$$= \frac{1}{2} \log_2(e) \approx 0.721348$$

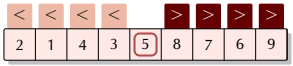
Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P

<	<	<	<		>	>	>	>
2	1	4	3	5	8	7	6	9



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \ldots, P_{s-1}$
  - s classes $c_1, \ldots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \ldots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3$, $\vec{t} = (1, 2, 3)$

⇝ $\vec{D} \stackrel{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \ldots, t_s+1)$

$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{t_r+1}$

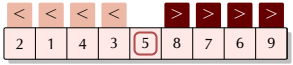
$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right)\cdot\log_2(e)$$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - ⇝ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    - ⇝ Select pivots from $k = 6$ elements

⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1 + 1, \dots, t_s + 1)$

⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r + 1}{k+1}\big(H_{k+1} - H_{t_r+1}\big) \cdot \log_2(e)$

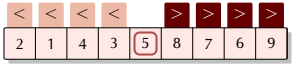
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - ⇝ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    - ⇝ choose pivots from $k = 6$ candidates

⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1 + 1, \dots, t_s + 1)$

⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r + 1}{k+1}\left(H_{k+1} - H_{t_r+1}\right) \cdot \log_2(e)$

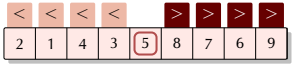
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - ⇝ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    - ⇝ choose pivots from $k = 8$ sample items



⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right)\cdot\log_2(e)$

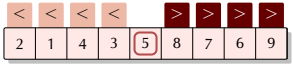
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - $\rightsquigarrow$ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    - $\rightsquigarrow$ choose pivots from $k = 8$ sample items

$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right)\cdot\log_2(e)$$

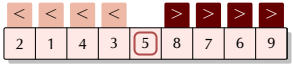
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items

$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right) \cdot \log_2(e)$

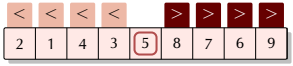
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

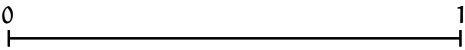
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$

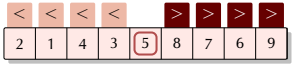
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

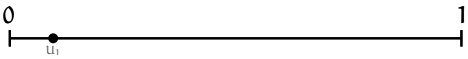
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$

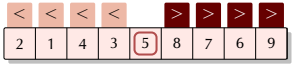
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

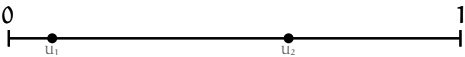
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

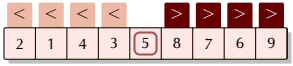
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

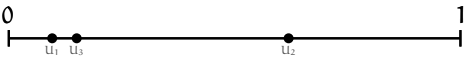
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

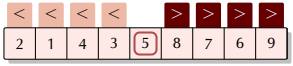
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ where $H_{t_r+1} = \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

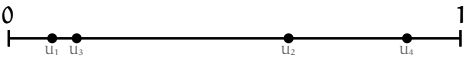
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  ⇝ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    ⇝ choose pivots from $k = 8$ sample items



⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

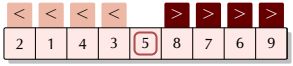
⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

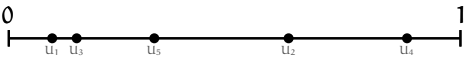
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

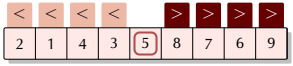
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

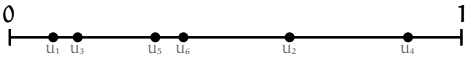
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  ⇝ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    ⇝ choose pivots from $k = 8$ sample items



⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right)\cdot\log_2(e)$

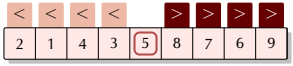
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

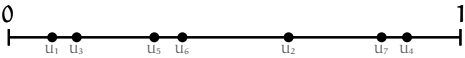
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big) \cdot \log_2(e)$

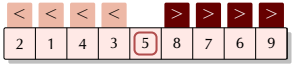
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

⇝ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

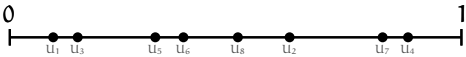
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  ⇝ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    ⇝ choose pivots from $k = 8$ sample items



⇝ $\vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

⇝ $\mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right) \cdot \log_2(e)$

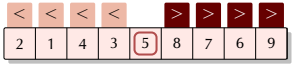
$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

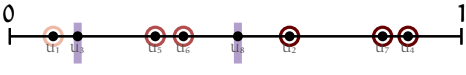
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

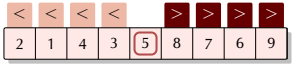
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

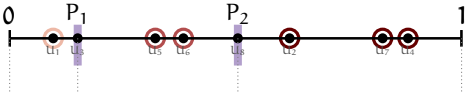
Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

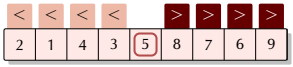
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

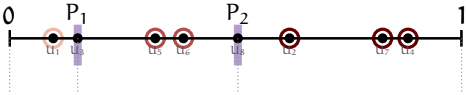
s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

no sampling $\rightsquigarrow \vec{t} = 0$

median-of-3 $\rightsquigarrow \vec{t} = (1,1)$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

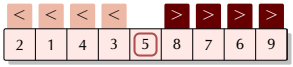
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

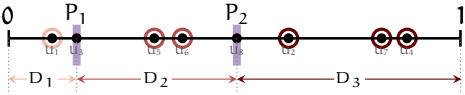
$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

no sampling $\rightsquigarrow \vec{t} = 0$

median-of-3 $\rightsquigarrow \vec{t} = (1,1)$

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - $\rightsquigarrow$ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

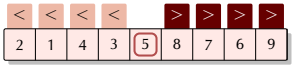
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$ with $H_{t_r+1} = \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\dots+\frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

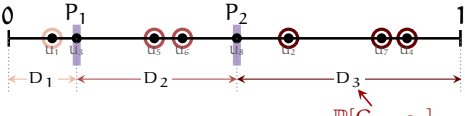
s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

no sampling $\rightsquigarrow \vec{t} = 0$

median-of-3 $\rightsquigarrow \vec{t} = (1,1)$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

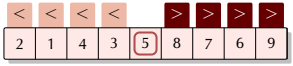
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\left(H_{k+1} - H_{t_r+1}\right)\cdot\log_2(e)$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

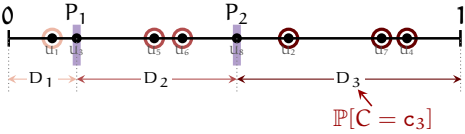
$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

no sampling $\rightsquigarrow \vec{t} = 0$

median-of-3 $\rightsquigarrow \vec{t} = (1,1)$

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - $\rightsquigarrow$ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

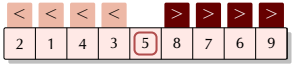
$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



- Outcome C is either small or large.

$\rightsquigarrow$ Information yield is $\mathcal{H}(C) = -\mathbb{P}[<]\log_2(\mathbb{P}[<]) - \mathbb{P}[>]\log_2(\mathbb{P}[>])$

Average over P

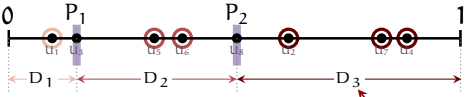
$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \mathbb{E}\Big[-P\log_2(P) - (1-P)\log_2(1-P)\Big] = -2\int_0^1 z\log_2(z)\,dz = \frac{1}{2}\log_2(e) \approx 0.721348$$

s-way Quicksort with pivot sampling:

no sampling $\rightsquigarrow \vec{t} = 0$

median-of-3 $\rightsquigarrow \vec{t} = (1,1)$

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  - $\rightsquigarrow$ s classes $c_1, \dots, c_s$
- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$
  - **Example:** $s = 3, \quad \vec{t} = (1,2,3)$
    $\rightsquigarrow$ choose pivots from $k = 8$ sample items



$\rightsquigarrow \vec{D} \overset{\mathcal{D}}{=} \text{Dirichlet}(t_1+1, \dots, t_s+1)$

$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{t_r+1}$

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1}\big(H_{k+1} - H_{t_r+1}\big)\cdot\log_2(e)$$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



$\mathbb{E}[\mathcal{H}(C)] =$ information yield of classifying 1 element

$\rightsquigarrow$ If we need a comparisons for that, Quicksort uses

$$\frac{a}{\mathbb{E}[\mathcal{H}(C)]} \cdot n \log_2(n) \pm O(n)$$

comparisons on average.

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

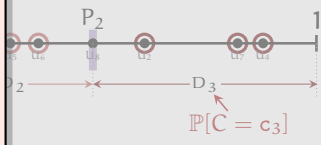
no sampling
$\rightsquigarrow \vec{t} = 0$

median-of-3
$\rightsquigarrow \vec{t} = (1,1)$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$

3, $\vec{t} = (1,2,3)$

vots from $k = 8$ sample items



$+1, \dots, t_s + 1)$

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r+1}{k+1} \big(H_{k+1} - H_{t_r+1}\big) \cdot \log_2(e)$$

$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Partitioning Entropy

What is x?

Classic Quicksort:

- Always compare with the pivot P



$\mathbb{E}[\mathcal{H}(C)] =$ information yield of classifying 1 element

$\rightsquigarrow$ If we need a comparisons for that, Quicksort uses

$$\frac{a}{\mathbb{E}[\mathcal{H}(C)]} \cdot n \log_2(n) \pm O(n)$$

comparisons on average.

s-way Quicksort with pivot sampling:

- $s-1$ pivots: $P_1, \dots, P_{s-1}$
  $\rightsquigarrow$ s classes $c_1, \dots, c_s$

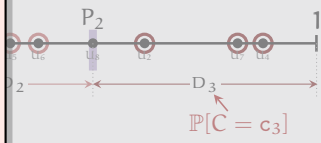
no sampling
$\rightsquigarrow \vec{t} = 0$

median-of-3
$\rightsquigarrow \vec{t} = (1,1)$

- **Pivot Sampling**: parameter $\vec{t} = (t_1, \dots, t_s) \in \mathbb{N}_0^s$

3, $\vec{t} = (1,2,3)$

vots from $k = 8$ sample items



$+1, \dots, t_s + 1)$

$$\rightsquigarrow \mathbb{E}[\mathcal{H}(C)] = \sum_{r=1}^{s} \frac{t_r + 1}{k+1} \left(H_{k+1} - H_{t_r+1}\right) \cdot \log_2(e)$$

$H_{t_r+1} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{t_r+1}$

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort

- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons ≈ 1.846 (without sampling)

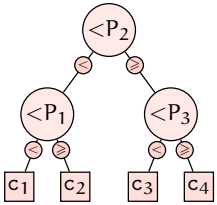
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

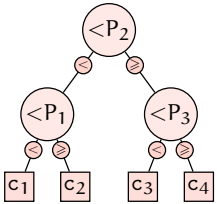
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

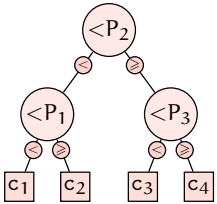
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

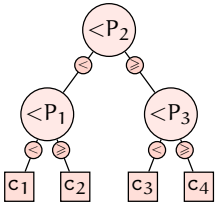
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

VS.

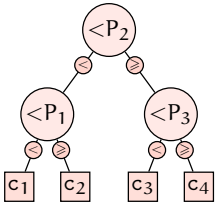
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

VS.

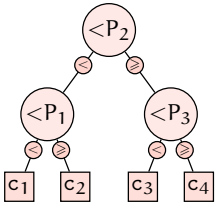
Classic Quicksort

- $2 n \ln(n)$ comparisons (without sampling)

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? Let's try an example.

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

VS.

Classic Quicksort

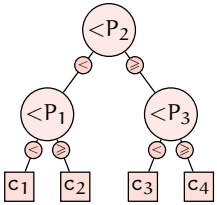
- $2 n \ln(n)$ comparisons (without sampling)

Aha! Multiway Quicksort is better!

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

VS.

Classic Quicksort

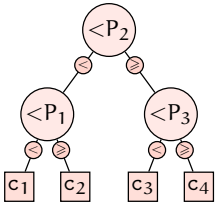
- $2 n \ln(n)$ comparisons (without sampling)
- $\frac{12}{7} n \ln(n)$ comparisons (median-of-3)
  ≈ 1.714

Aha! Multiway Quicksort is better!

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? Let's try an example.

4-way Quicksort



- Need $a = 2$ comparisons to classify 1 item.
- $\frac{24}{13} n \ln(n)$ comparisons (without sampling)
  ≈ 1.846

VS.

Classic Quicksort

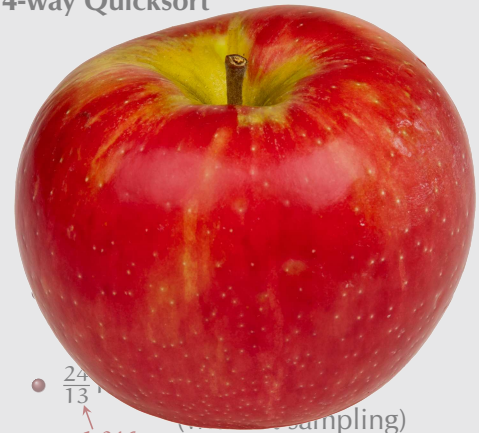
- $2 n \ln(n)$ comparisons (without sampling)
- $\frac{12}{7} n \ln(n)$ comparisons (median-of-3)
  ≈ 1.714

Aha! Multiway Quicksort is better!
...or not; it depends on **sampling**.

Binary vs. Multiway Partitioning

Now, does multiway partitioning perform better? *Let's try an example.*

4-way Quicksort



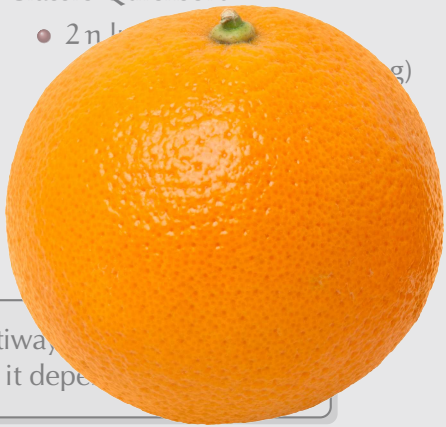
- $\frac{24}{13}$ (…sampling)

≈ 1.846

VS.

Classic Quicksort

- $2\,n\,l$… …g)



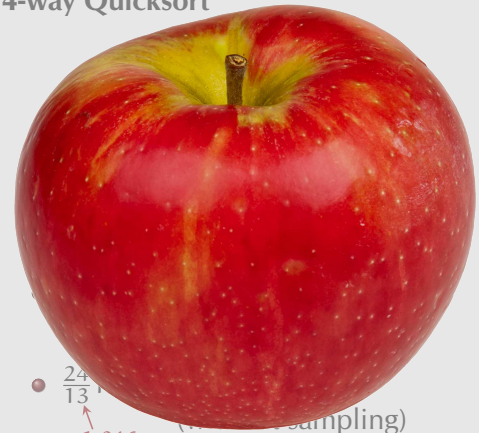
Aha! Multiway …
…or not; it depe…

Binary vs. Multiway Partitioning

Now, d

How to **separate** influence of multiway partitioning from that of sampling?

4-way Quicksort



- $\frac{24}{13}$ ... sampling)

≈ 1.846

VS.

Classic Quicksort

- $2\,n$ ... g)



Aha! Multiwa...
...or not; it depe...

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning ...

(without sampling)

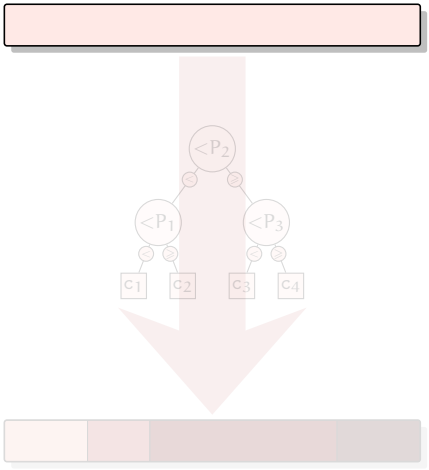
... corresponds to three binary partitionings.

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning …

(without sampling)



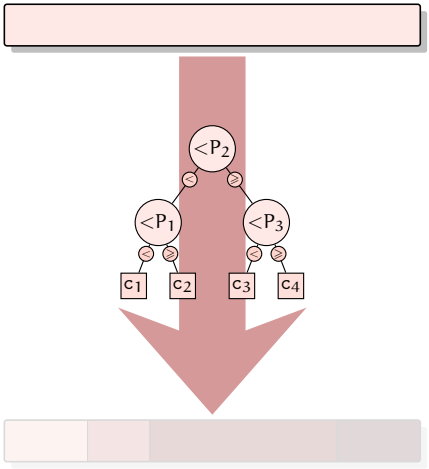
… corresponds to three binary partitionings.

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning ...

(without sampling)



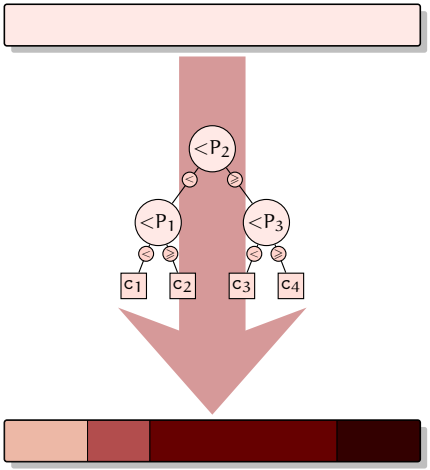
... corresponds to three binary partitionings.

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning …

(without sampling)



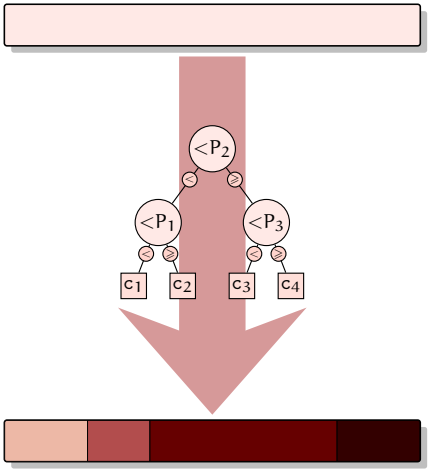
… corresponds to three binary partitionings.

Entropy-Equivalent Sampling

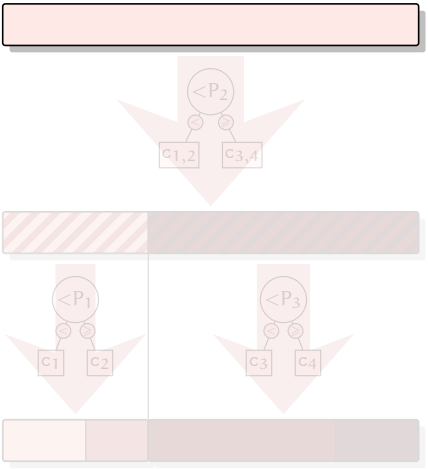
Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning . . .

(without sampling)



. . . corresponds to three binary partitionings.

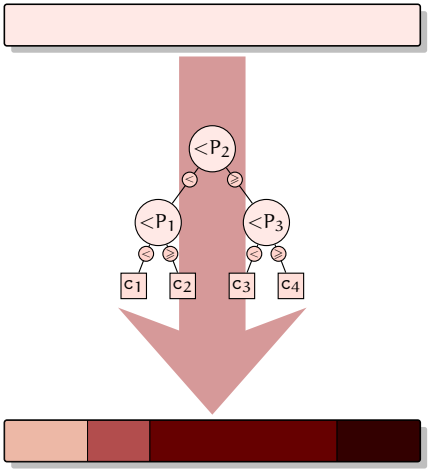


Entropy-Equivalent Sampling

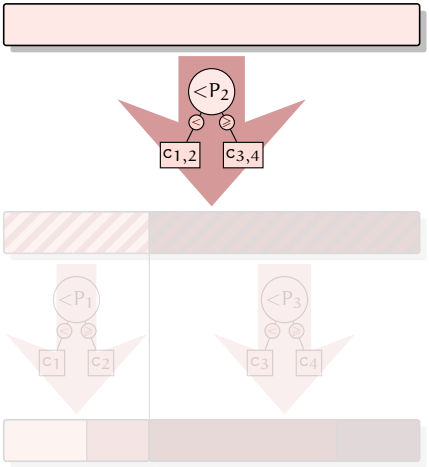
Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning . . .

(without sampling)



. . . corresponds to three binary partitionings.

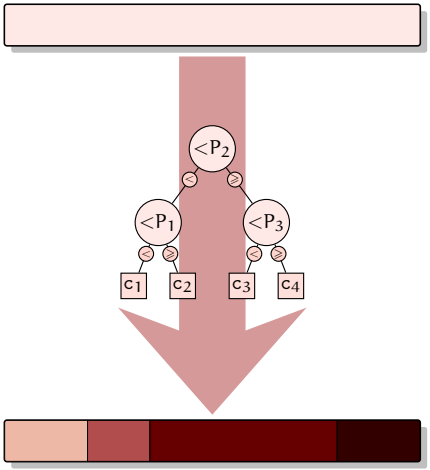


Entropy-Equivalent Sampling

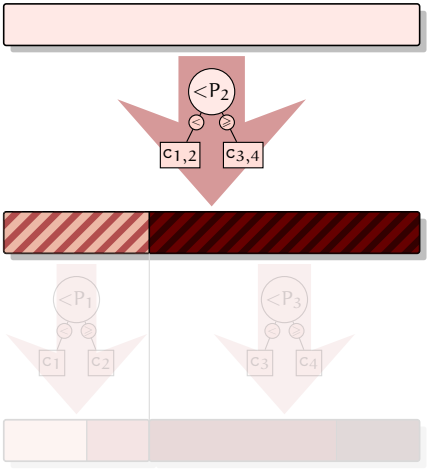
Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning . . .

(without sampling)



. . . corresponds to three binary partitionings.

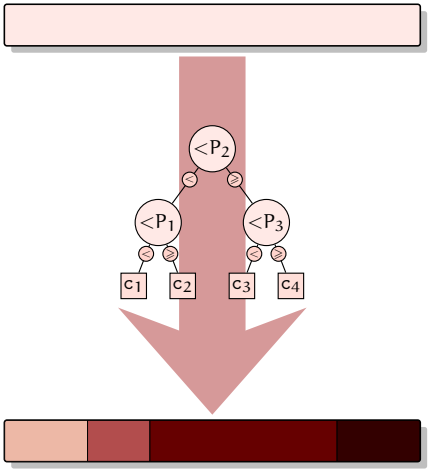


Entropy-Equivalent Sampling

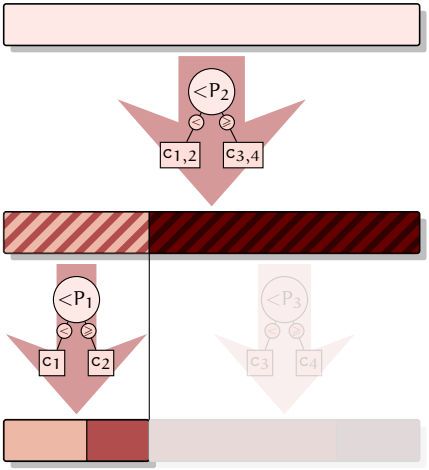
Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning . . .

(without sampling)



. . . corresponds to three binary partitionings.

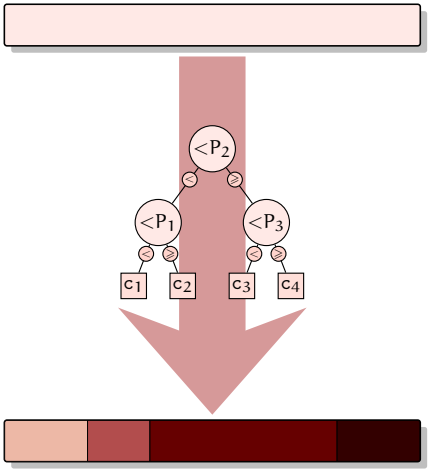


Entropy-Equivalent Sampling

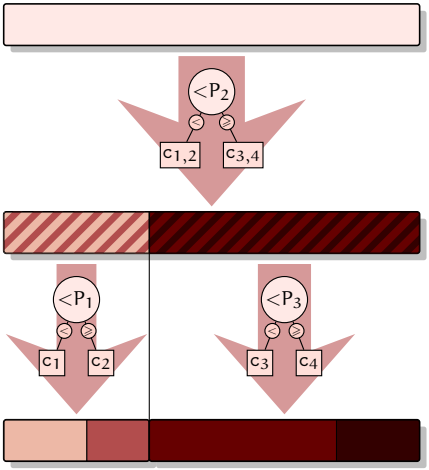
Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning . . .

(without sampling)



. . . corresponds to three binary partitionings.

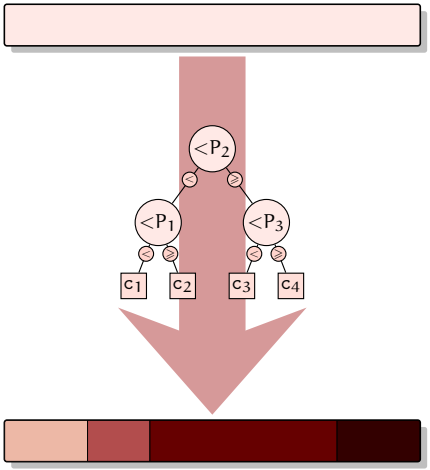


Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

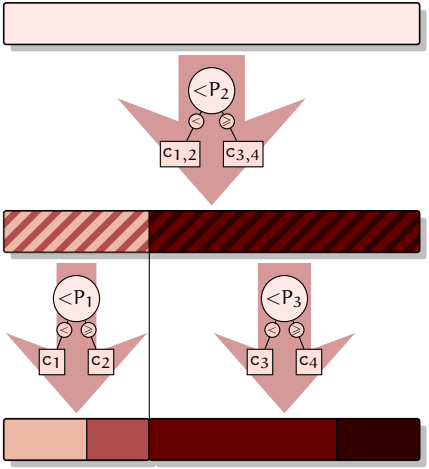
One round of 4-way partitioning ...

(without sampling)



... corresponds to three binary partitionings.

But how to choose pivots P_1, P_2, P_3?

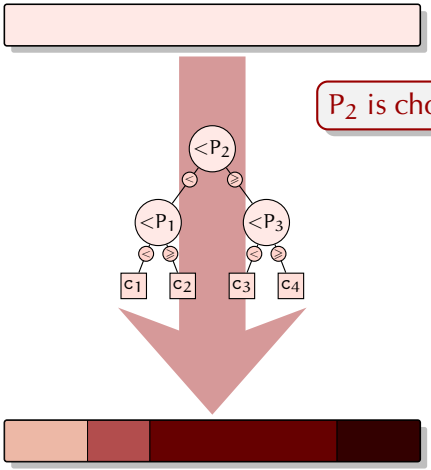


Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning ...

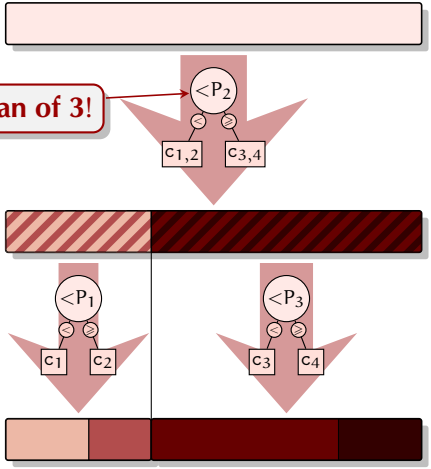
(without sampling)



... corresponds to three binary partitionings.

But how to choose pivots P_1, P_2, P_3?

P_2 is chosen as **median of 3**!



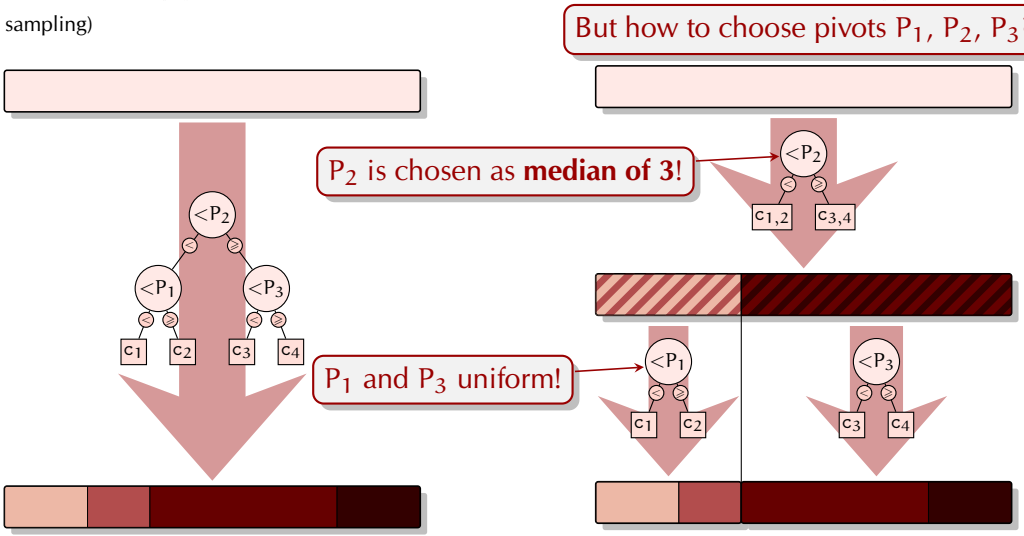
Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

One round of 4-way partitioning ...

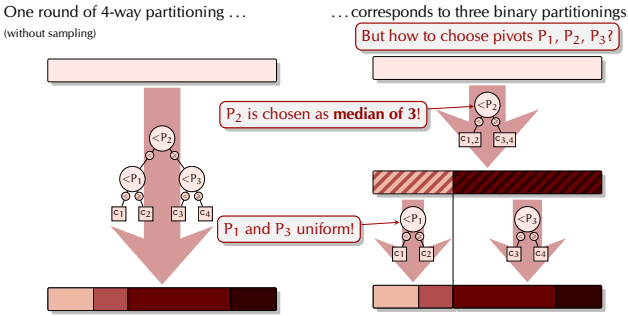
(without sampling)

... corresponds to three binary partitionings.



Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

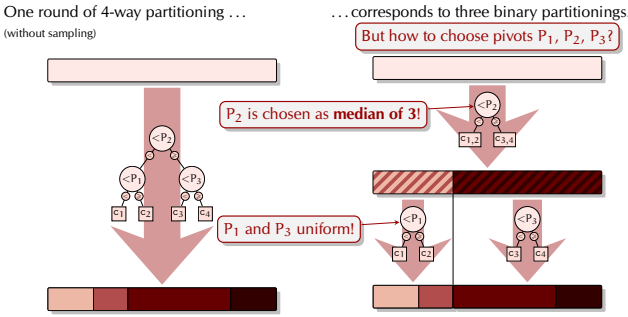


- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads / no sampling for tails

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

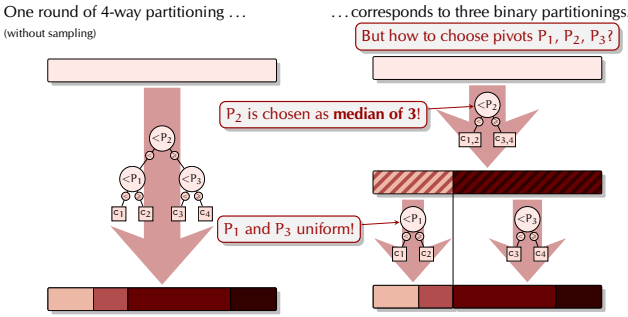


- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads; no sampling for tails

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

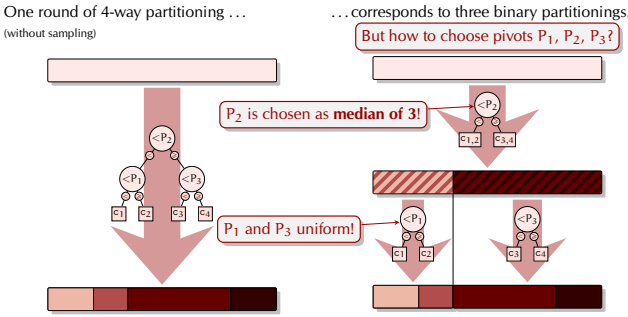


- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads; no sampling for tails }

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

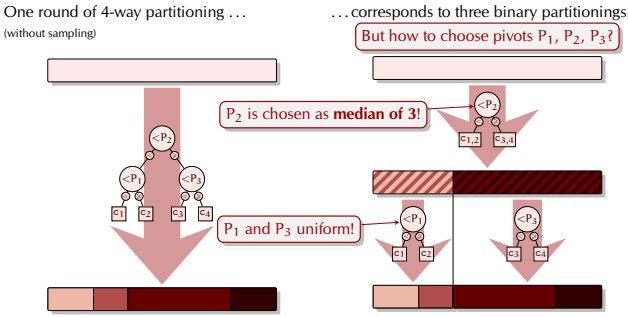


- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads; no sampling for tails }

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.

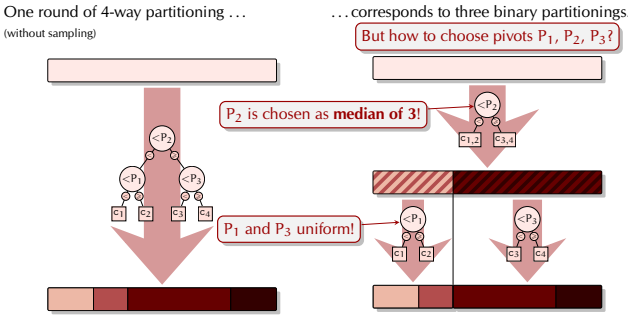


- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads; no sampling for tails }

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.



⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.

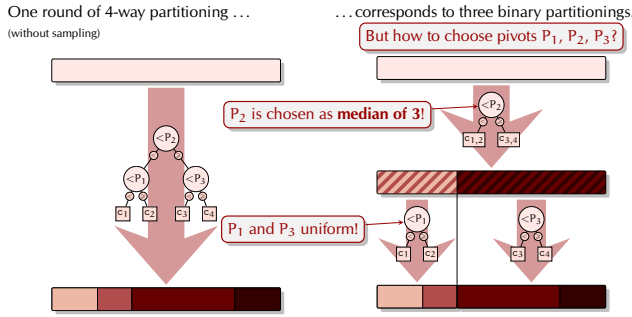
⇝ **same on average**:

1. flip a fair coin
2. { median-of-3 for heads; no sampling for tails }

random-parameter pivot sampling

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.



- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. { median-of-3 for heads; no sampling for tails }

random-parameter pivot sampling

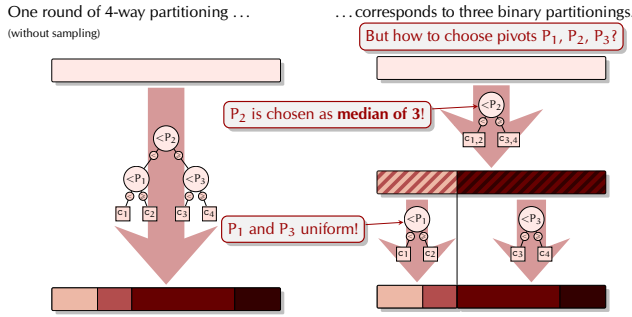
Theorem ***(Entropy-Equivalent Sampling)***

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Entropy-Equivalent Sampling

Key Idea: Simulate by binary partitioning.



- ⇝ can **simulate** 4-way partitioning **exactly** by **alternating** between median-of-3 and no sampling.
- ⇝ **same on average**:
  1. flip a fair coin
  2. $\begin{cases} \text{median-of-3} & \text{for heads} \\ \text{no sampling} & \text{for tails} \end{cases}$

random-parameter pivot sampling

Theorem ***(Entropy-Equivalent Sampling)***

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1, 2, 3)$ with 

$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)



rtitioning
between
mpling.

heads
tails
ot sampling

What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!

Theorem *(Entropy-Equivalent Sampling)*

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)



What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!

rtitioning
between
mpling.

heads
tails
ot sampling

Theorem *(Entropy-Equivalent Sampling)*

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)



What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!

(with 1 comparison tree)

rtitioning
between
mpling.

heads
tails
ot sampling

Theorem *(Entropy-Equivalent Sampling)*

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)

What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!
   (with 1 comparison tree)

So, is that the end of multiway Quicksort?



rtitioning
between
mpling.

heads
tails
ot sampling

Theorem *(Entropy-Equivalent Sampling)*

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)

What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!
   (with 1 comparison tree)



So, is that the end of multiway Quicksort?



No! . . . stay tuned

rtitioning
between
mpling.

heads
tails
ot sampling

Theorem *(Entropy-Equivalent Sampling)*

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

Entropy-Equivalent Sampling

Key Idea: Sir

One round of 4-wa
(without sampling)

What does that mean?

1. Can **simulate** any **multiway** method by binary partitioning.
2. In terms of **comparisons**, this can only get **better**!
   (with 1 comparison tree)



So, is that the end of multiway Quicksort?



No! …stay tuned

rtitioning
between
mpling.

heads
tails
ot sampling

Theorem ***(Entropy-Equivalent Sampling)***

For every s-way sampling parameter $\vec{t}$ there is a single-pivot random-parameter scheme with the same quality of pivots.

(formal meaning in dissertation)

Example: $\vec{t} = (1,2,3)$ with



$$\rightsquigarrow \vec{T} = \begin{cases} (1,6) & \text{with prob. } \frac{9}{16} \\ (2,3) & \text{otherwise} \end{cases}$$

The *Memory Wall*

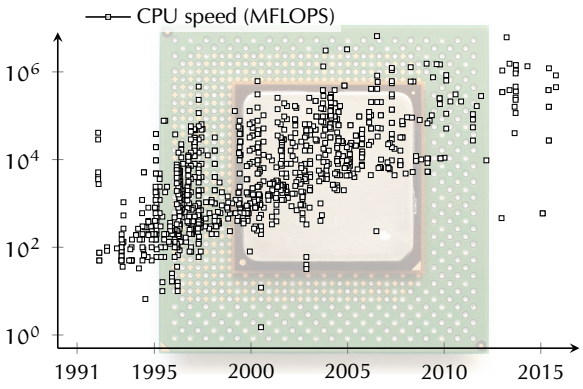
Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.

The *Memory Wall*

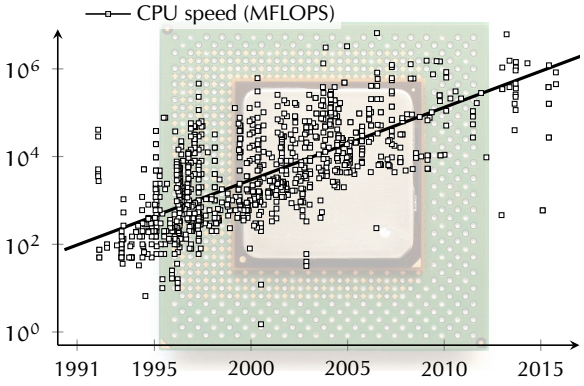
Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

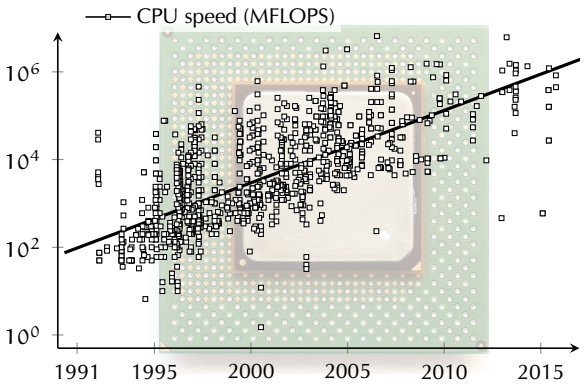
Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



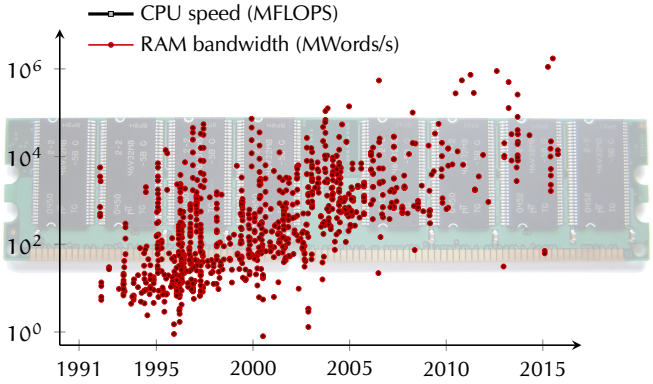
STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

Averaged annual growth rates:

Processor speed: 46%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



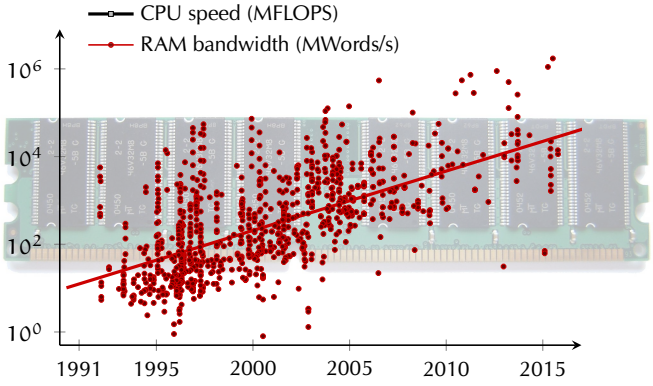
Averaged annual growth rates:

Processor speed: 46%

STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



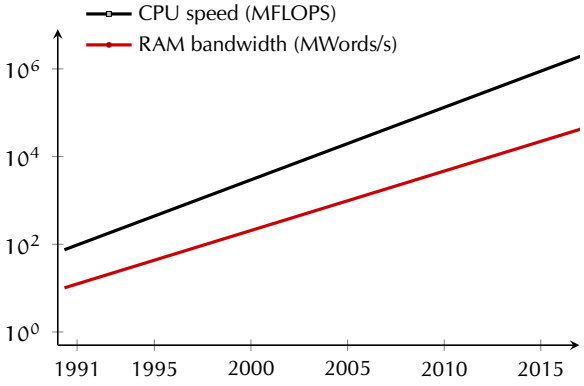
Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%

STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



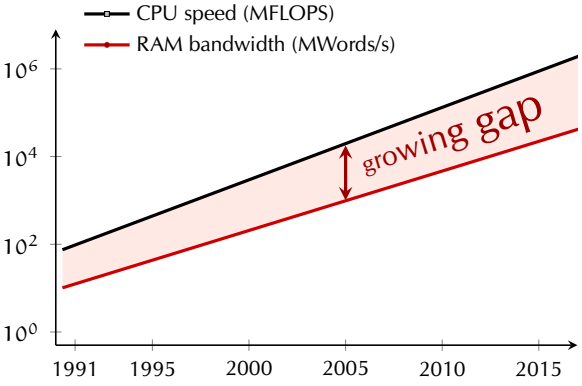
Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%

STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



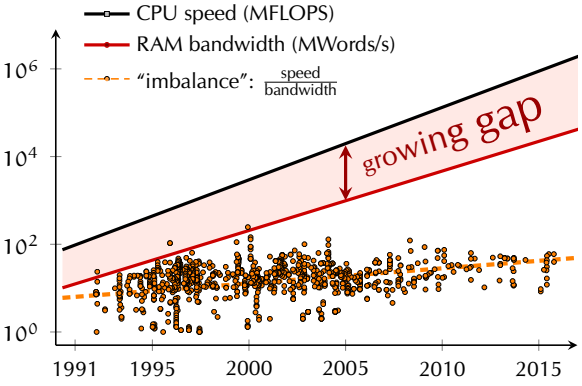
Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%

STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



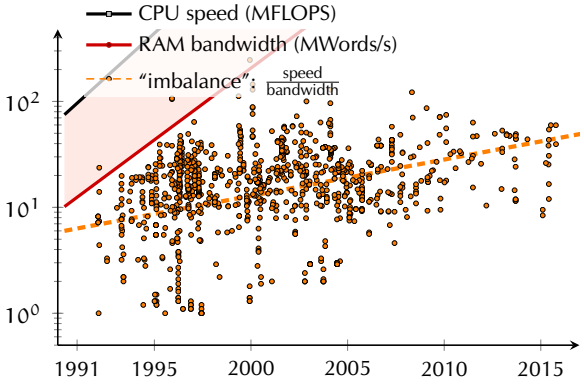
Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%

STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



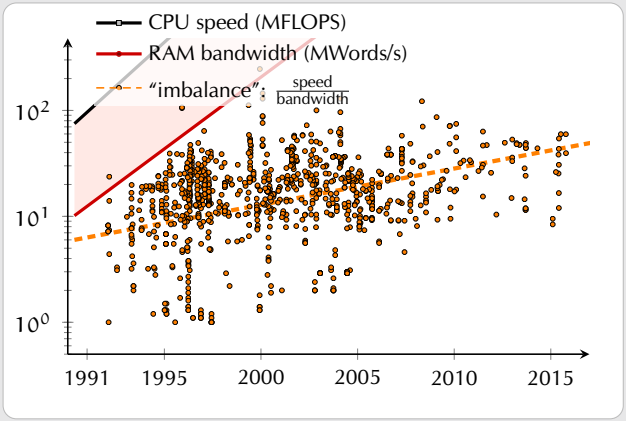
STREAM benchmark data with linear regressions
`www.cs.virginia.edu/stream/by_date/Balance.html`

Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%
$\rightsquigarrow$ Imbalance: 5%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need to **load** items from **memory**.



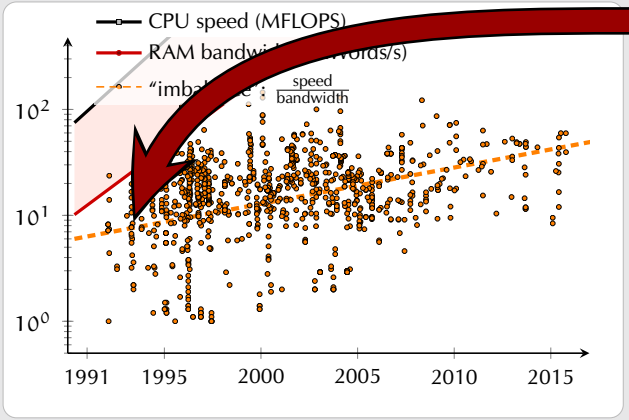
STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Averaged annual growth rates:

Processor speed: 46%
Memory Bandwidth: 37%
$\rightsquigarrow$ Imbalance: 5%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need



STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

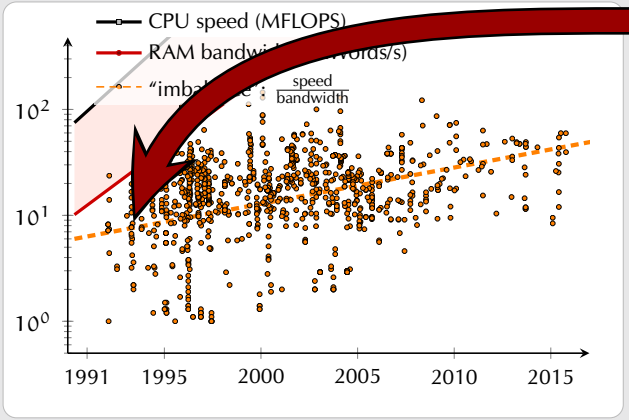
Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations
- $\rightsquigarrow$ optimized for CPU

Memory Bandwidth: 37%
$\rightsquigarrow$ Imbalance: 5%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need



STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

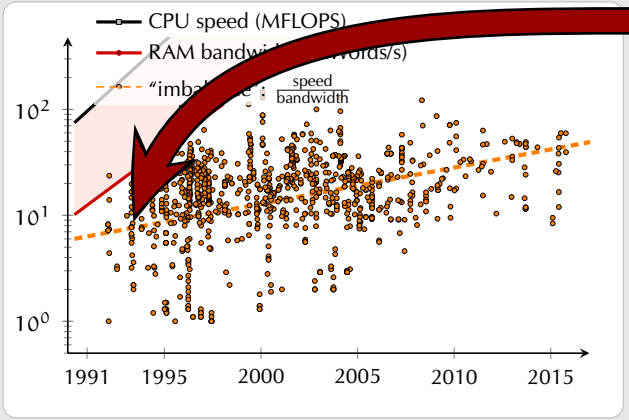
- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU

Processor speed: 46%
Memory Bandwidth: 37%
$\rightsquigarrow$ Imbalance: 5%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need

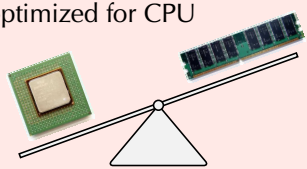


STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU

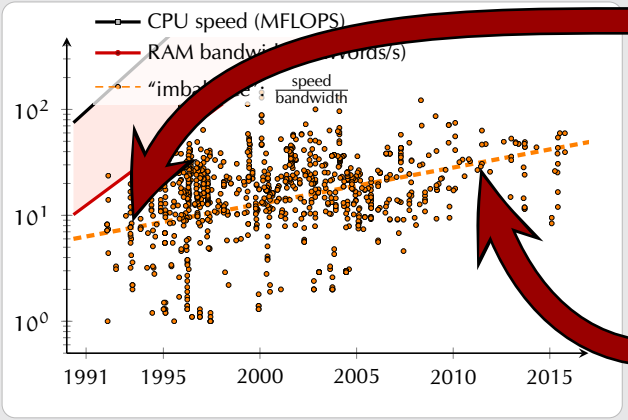


Memory Bandwidth: 37%

$\rightsquigarrow$ Imbalance: 5%

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need

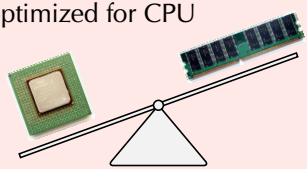


STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU



Processor speed: 40%

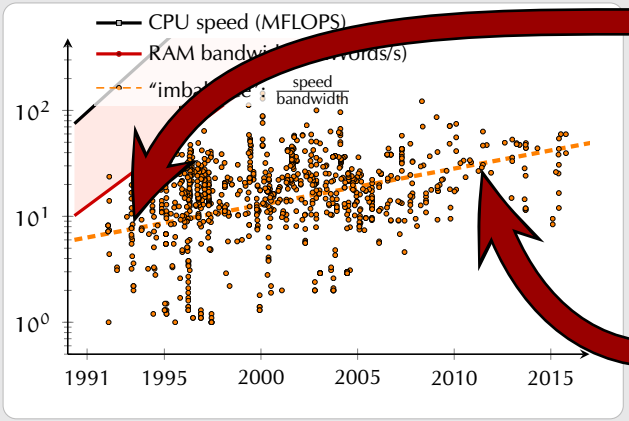
Dual-Pivot Quicksort, **2009 – today**

- 1 mem. transfer $\approx$ 30 operations

$\rightsquigarrow$ optimize for CPU **and bandwidth?**

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need

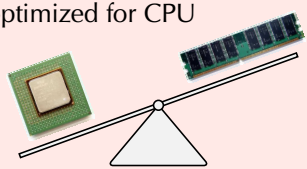


STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU



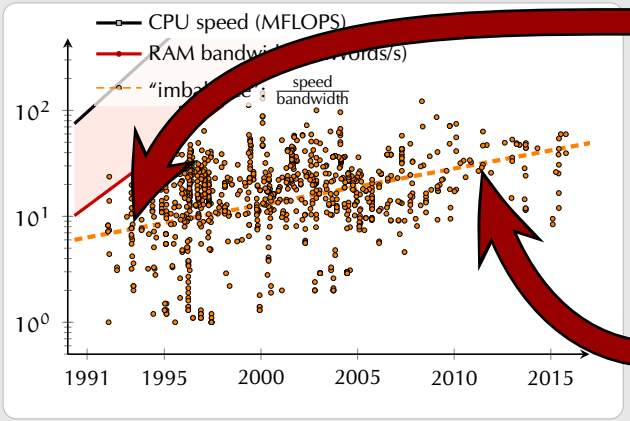
Dual-Pivot Quicksort, **2009 – today**

- 1 mem. transfer $\approx$ 30 operations

$\rightsquigarrow$ optimize for CPU and bandwidth?

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need

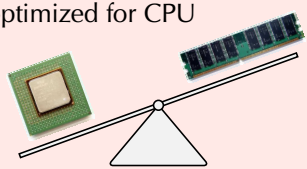


STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

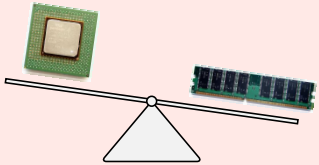
$\rightsquigarrow$ optimized for CPU



Dual-Pivot Quicksort, **2009 – today**

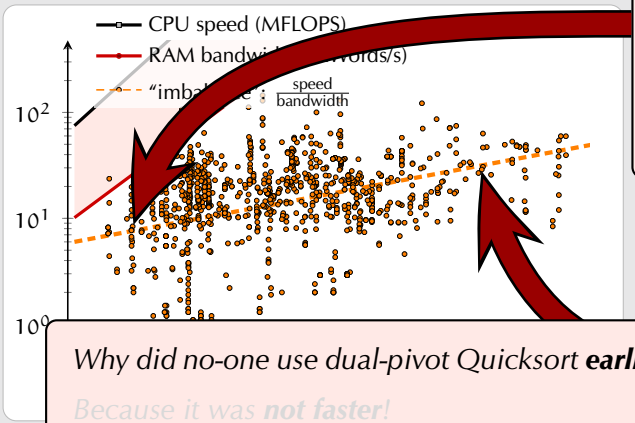
- 1 mem. transfer $\approx$ 30 operations

$\rightsquigarrow$ optimize for CPU **and bandwidth**?



The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need

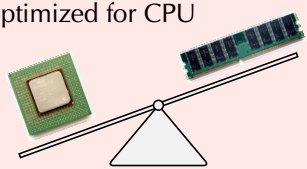


STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU

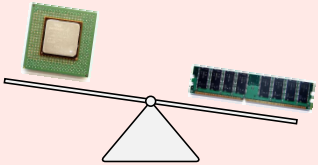


Processor speed: 40%

Dual-Pivot Quicksort, **2009 – today**

- 1 mem. transfer $\approx$ 30 operations

$\rightsquigarrow$ optimize for CPU **and bandwidth**?

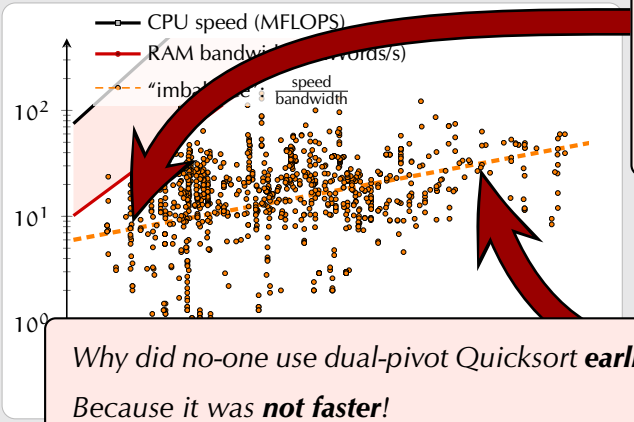


*Why did no-one use dual-pivot Quicksort **earlier**?*

*Because it was **not faster**!*

The *Memory Wall*

Comparisons $\widehat{=}$ **CPU** effort, but we also need



*Why did no-one use dual-pivot Quicksort **earlier**?*

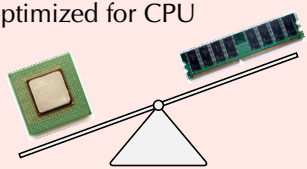
*Because it was **not faster**!*

STREAM benchmark data with linear regressions
www.cs.virginia.edu/stream/by_date/Balance.html

Classic Quicksort (Bentley & McIlroy) **1993**

- 1 mem. transfer $\approx$ 7 operations

$\rightsquigarrow$ optimized for CPU

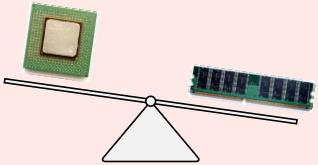


Processor speed: 40%

Dual-Pivot Quicksort, **2009 – today**

- 1 mem. transfer $\approx$ 30 operations

$\rightsquigarrow$ optimize for CPU **and bandwidth**?



Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)

- Iterators can
  - read left or right (one-directional!)
  - advance to next position
  - read and write current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

⇝ n scanned elements
(array scanned once)

4-way Quicksort

⇝ $1.5n$ scanned elements
(on average without sampling)

Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)
- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

⇝ n scanned elements
(array scanned once)

4-way Quicksort

⇝ $1.5n$ scanned elements
(on average without sampling)

Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)
- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

⇝ n scanned elements
(array scanned once)

4-way Quicksort

⇝ $1.5n$ scanned elements
(on average without sampling)

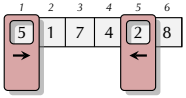
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

$\rightsquigarrow$ n scanned elements
(array scanned once)

4-way Quicksort

$\rightsquigarrow$ $1.5n$ scanned elements
(on average without sampling)

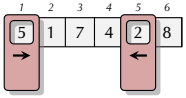
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

⇝ n scanned elements
(array scanned once)

4-way Quicksort

⇝ $1.5n$ scanned elements
(on average without sampling)

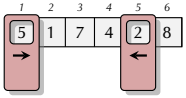
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

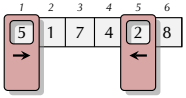
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

$\rightsquigarrow$ n scanned elements
(array scanned once)

4-way Quicksort

$\rightsquigarrow$ $1.5n$ scanned elements
(on average without sampling)

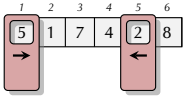
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

⇝ n scanned elements
(array scanned once)

4-way Quicksort

⇝ $1.5n$ scanned elements
(on average without sampling)

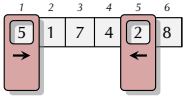
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

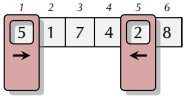
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

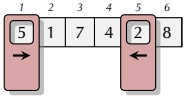
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)



- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort

$\rightsquigarrow$ n scanned elements
(array scanned once)

4-way Quicksort

$\rightsquigarrow$ $1.5n$ scanned elements
(on average without sampling)

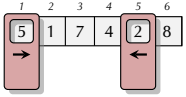
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)

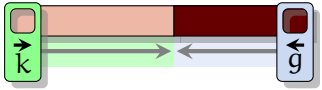


- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort



$\rightsquigarrow$ n scanned elements
(array scanned once)

4-way Quicksort

$\rightsquigarrow$ $1.5n$ scanned elements
(on average without sampling)

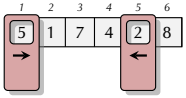
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)

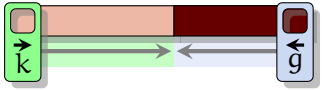


- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

$\approx$ **cache misses**, but hardware independent

Costs of Partitioning

Classic Quicksort



$\rightsquigarrow$ n scanned elements
(array scanned once)

4-way Quicksort

$\rightsquigarrow$ $1.5n$ scanned elements
(on average without sampling)

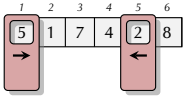
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)

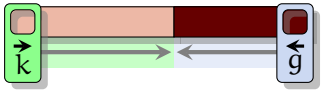


- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

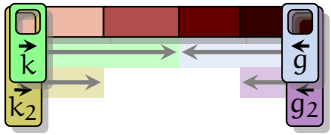
Classic Quicksort



⇝ n scanned elements
(array scanned once)

4-way Quicksort

details on how this works in one pass → in dissertation



⇝ $1.5n$ scanned elements
(on average without sampling)

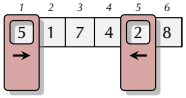
Scanned Elements

How to analyze bandwidth consumption?

Need good cost measure!

Proposal: count ***scanned elements***

- Access to array only through **iterators**
  (no random access!)

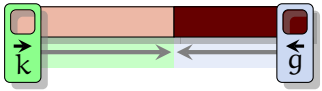


- Iterators can
  - head left or right (one-directional!)
  - **advance** to next position
  - **read** and **write** current cell
- Cost: number of **advances**

≈ **cache misses**, but hardware independent

Costs of Partitioning

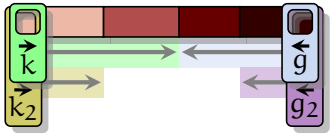
Classic Quicksort



⇝ n scanned elements
(array scanned once)

4-way Quicksort

details on how this works in one pass → in dissertation



⇝ $1.5n$ scanned elements
(on average without sampling)

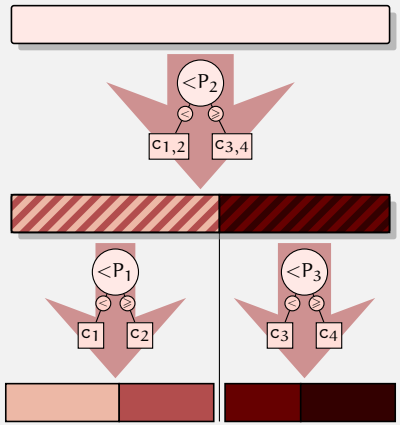
Scanned Elements

Ho

Ne

Pr

4-way simulated by 2-way

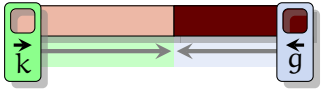


⇝ $2n$ scanned elements

≈ … dent

Costs of Partitioning

Classic Quicksort

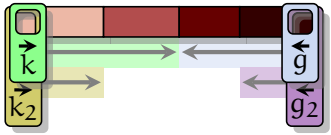


⇝ n scanned elements

(array scanned once)

4-way Quicksort

details on how this works in one pass → in dissertation



⇝ $1.5n$ scanned elements

(on average without sampling)

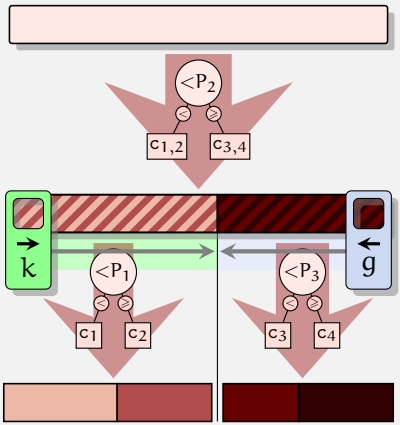
Scanned Elements

Ho

Ne

Pr

4-way simulated by 2-way

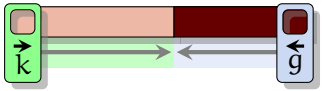


⇝ $2n$ scanned elements

dent

Costs of Partitioning

Classic Quicksort

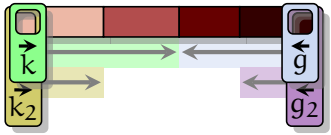


⇝ n scanned elements

(array scanned once)

4-way Quicksort

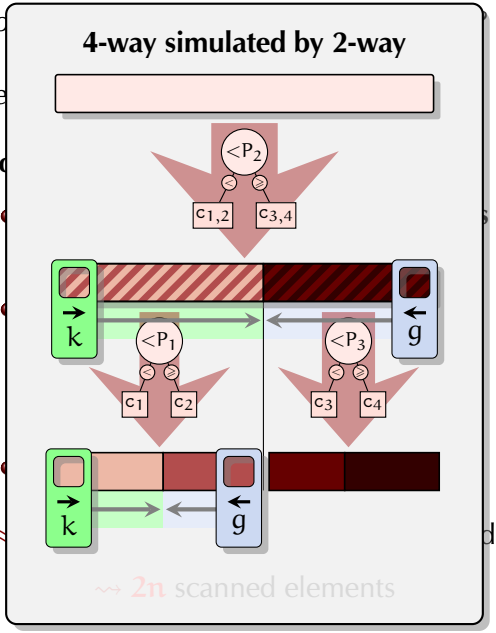
details on how this works in one pass → in dissertation



⇝ $1.5n$ scanned elements

(on average without sampling)

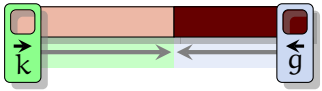
Scanned Elements



details on how this works in one pass → in dissertation

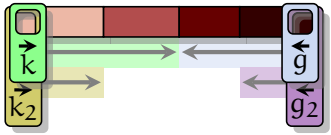
Costs of Partitioning

Classic Quicksort



⇝ n scanned elements
(array scanned once)

4-way Quicksort



⇝ $1.5n$ scanned elements
(on average without sampling)

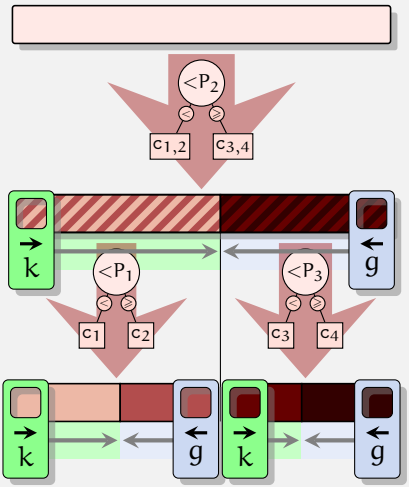
Scanned Elements

Ho

Ne

Pr

4-way simulated by 2-way

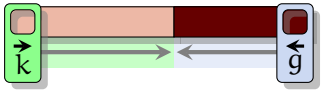


⇝ $2n$ scanned elements

dent

Costs of Partitioning

Classic Quicksort

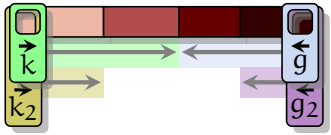


⇝ n scanned elements

(array scanned once)

4-way Quicksort

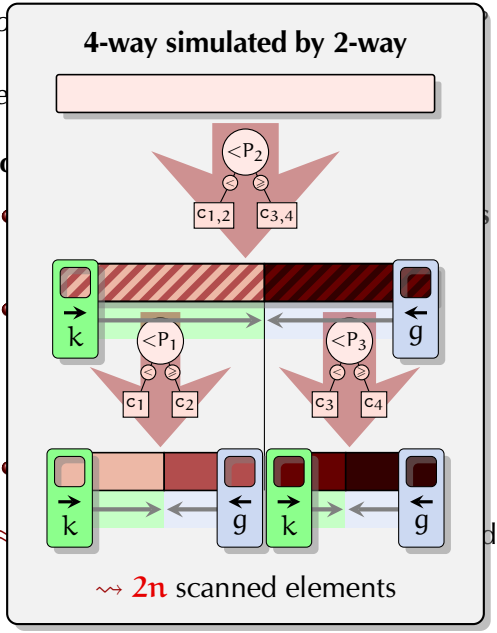
details on how this works in one pass → in dissertation



⇝ $1.5n$ scanned elements

(on average without sampling)

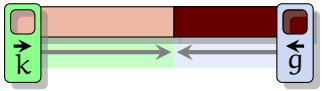
Scanned Elements



details on how this works in one pass → in dissertation

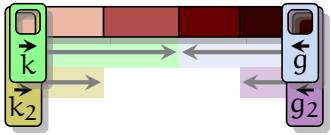
Costs of Partitioning

Classic Quicksort



⇝ n scanned elements
(array scanned once)

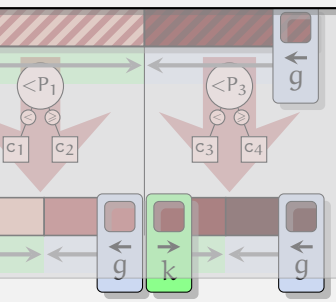
4-way Quicksort



⇝ $1.5n$ scanned elements
(on average without sampling)

Scanned Elements

4-way simulated by 2-way



⇝ 2n scanned elements

Multiway uses much **less** **scanned elements**, irrespective of pivot sampling.

Costs of Partitioning

ksort



⇝ n scanned elements
(array scanned once)

4-way Quicksort

details on how this works in one pass → in dissertation



⇝ 1.5n scanned elements
(on average without sampling)

Conclusion

You have seen:

My Thesis

1. *Multiway partitioning has genuine potential to speed up Quicksort.*

…namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

3. Idea of mathematical analysis

$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Conclusion

You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

3. Idea of mathematical analysis

$$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

Conclusion

You have seen:

My Thesis

1. *Multiway partitioning has genuine potential to speed up Quicksort.*

. . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

3. Idea of mathematical analysis

$$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

Conclusion

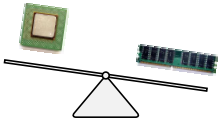
You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

3. Idea of mathematical analysis

   $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

Conclusion

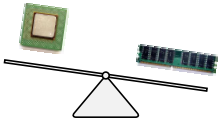
You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.



3. Idea of mathematical analysis

$$\mathcal{H}(C) = -\mathbb{P}[\,<\,] \log_2(\mathbb{P}[\,<\,]) - \mathbb{P}[\,>\,] \log_2(\mathbb{P}[\,>\,])$$

Conclusion

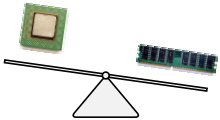
You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.



3. Idea of mathematical analysis

$$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

You have *not* seen: (in this talk)

- Generic multiway partitioning code

- Using two comparison trees

- Branch mispredictions

- Treatment of equal keys

- Intriguing interplay of parameters

Conclusion

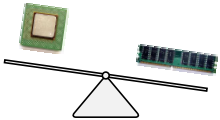
You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

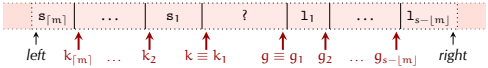
   

3. Idea of mathematical analysis

   $$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

You have *not* seen: (in this talk)

- Generic multiway partitioning code

- Using two comparison trees

- Branch mispredictions

- Treatment of equal keys

- Intriguing interplay of parameters

Conclusion

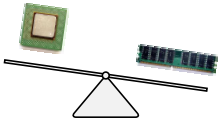
You have seen:

1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

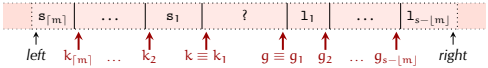
   

3. Idea of mathematical analysis

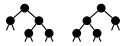
   $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

You have *not* seen: (in this talk)

- Generic multiway partitioning code

- Using two comparison trees

- Branch mispredictions

- Treatment of equal keys

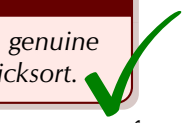
- Intriguing interplay of parameters

Conclusion

You have seen:

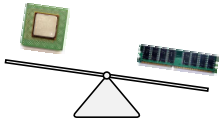
1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

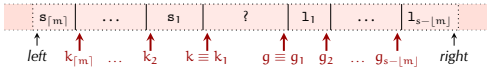
   

3. Idea of mathematical analysis

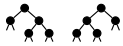
   $\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$

You have *not* seen: (in this talk)

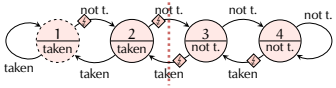
- Generic multiway partitioning code

- Using two comparison trees

- Branch mispredictions

- Treatment of equal keys

- Intriguing interplay of parameters

Conclusion

You have seen:

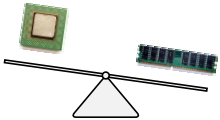
1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

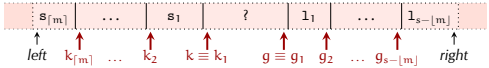


3. Idea of mathematical analysis

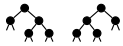
$$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

You have *not* seen: (in this talk)

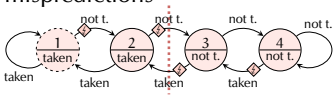
- Generic multiway partitioning code



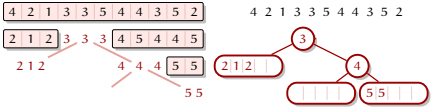
- Using two comparison trees



- Branch mispredictions



- Treatment of equal keys



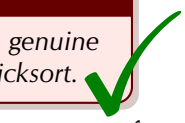
- Intriguing interplay of parameters

Conclusion

You have seen:

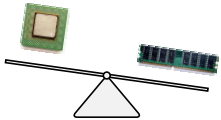
1. **My Thesis**

   Multiway partitioning has genuine potential to speed up Quicksort.

   . . . namely: it reduces memory transfers.

2. Evidence suggesting high impact on running time.

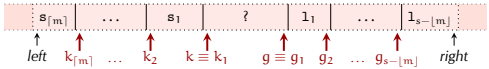
   

3. Idea of mathematical analysis

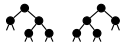
   $$\mathcal{H}(C) = -\mathbb{P}[<] \log_2(\mathbb{P}[<]) - \mathbb{P}[>] \log_2(\mathbb{P}[>])$$

You have *not* seen: (in this talk)

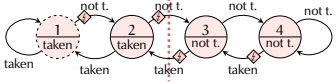
- Generic multiway partitioning code

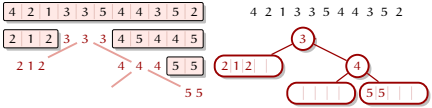
- Using two comparison trees

- Branch mispredictions

- Treatment of equal keys

- Intriguing interplay of parameters

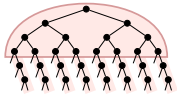
  



Image sources

Icons by *Freepik* and *Gregor Cresnar* from `www.flaticon.com`.

Ball Pit: `https://www.flickr.com/photos/davidsingleton/2175975083/`
Sorting Hat: `http://harrypotter.wikia.com/wiki/Sorting_Hat`
Highest Towers: Wikimedia
Mouse Trap: bobbleheaddad.com
Robert Sedgewick: `http://www.cs.princeton.edu/~rs/`
Apple: Wikimedia
Orange: Wikimedia
CPU Die: Wikimedia
RAM: Wikimedia