Dual-Pivot Quicksort — Asymmetries in Sorting

Sebastian Wild Markus E. Nebel

[wild, nebel] @cs.uni-kl.de

Computer Science Department



Feb 2014

Dagstuhl Seminar 14 091

Data Structures and Advanced Models of Computation on Big Data

Sorting Algorithms in Practice

Many inventions
by algorithms comunity

vs.

Few methods
successful in practice



Sorting methods listed on Wikipedia

- C
- C++
- Java 6
- .NET
- Haskell

Quicksort
+Mergesort variant as stable sort

- Python

Timsort

Sorting methods of standard libraries for random access data

Sorting Algorithms in Practice

Many inventions
by algorithms comunity

vs.

Few methods
successful in practice



Sorting methods listed on Wikipedia

- C
- C++
- Java 6
- .NET
- Haskell

Quicksort
+ Mergesort variant as stable sort

- Python

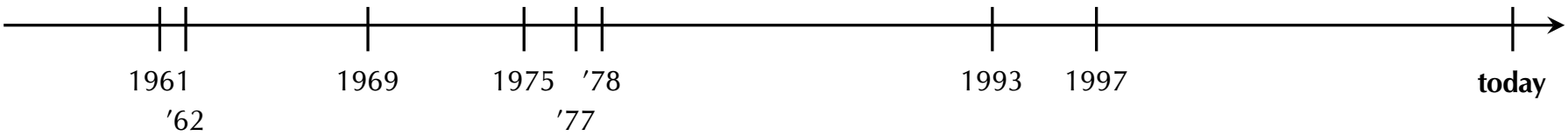
Timsort

Sorting methods of standard libraries for random access data

History of Quicksort in Practice

- **1961,62 Hoare:** first publication, average case analysis
- **1969 Singleton:** median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick:** detailled analysis of many optimizations
- **1993 Bentley, McIlroy:** *Engineering a Sort Function*
- **1997 Musser:** $\mathcal{O}(n \log n)$ worst case by bounded recursion depth

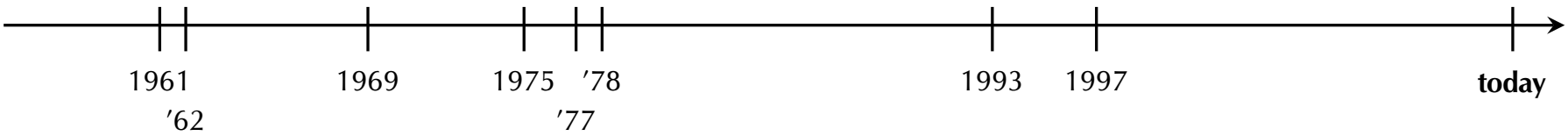
$\rightsquigarrow$ Basic algorithm settled since 1961; latest tweaks from 1990's.
Since then: Almost identical in all programming libraries!



History of Quicksort in Practice

- **1961,62 Hoare:** first publication, average case analysis
- **1969 Singleton:** median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick:** detailled analysis of many optimizations
- **1993 Bentley, McIlroy:** *Engineering a Sort Function*
- **1997 Musser:** $\mathcal{O}(n \log n)$ worst case by bounded recursion depth

$\rightsquigarrow$ Basic algorithm settled since 1961; latest tweaks from 1990's.
Since then: Almost identical in all programming libraries!

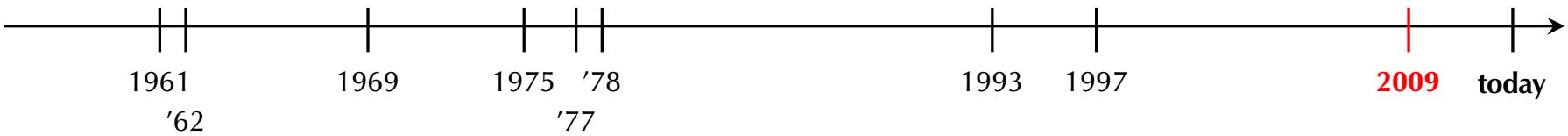


History of Quicksort in Practice

- **1961,62 Hoare:** first publication, average case analysis
- **1969 Singleton:** median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick:** detailled analysis of many optimizations
- **1993 Bentley, McIlroy:** *Engineering a Sort Function*
- **1997 Musser:** $\mathcal{O}(n \log n)$ worst case by bounded recursion depth

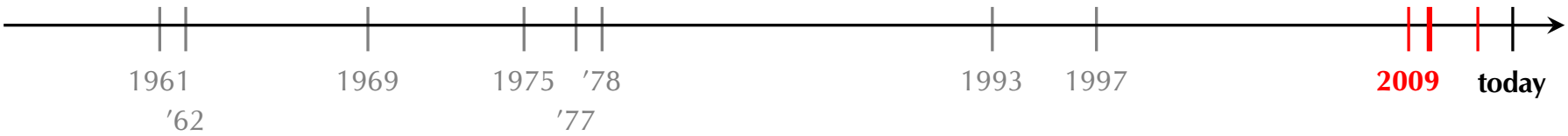
$\rightsquigarrow$ Basic algorithm settled since 1961; latest tweaks from 1990's.
Since then: Almost identical in all programming libraries!

- **Until 2009**: **Java 7** switches to a new **dual pivot** Quicksort!



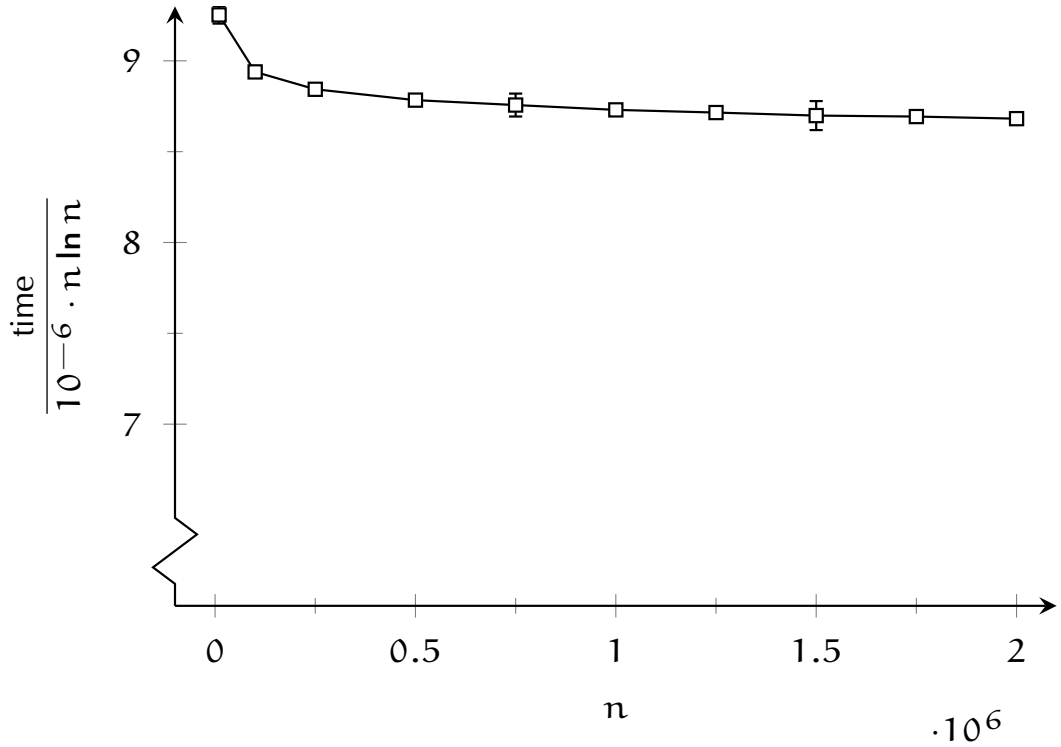
Discovery of Yaroslavskiy's Algorithm

2009	**Vladimir Yaroslavskiy** (researcher in St. Petersburg) experiments with Quicksort with **two pivots**
11 Sep 2009	announcement on Java core library mailing list
29 Oct 2009	**inclusion** in development version of library
2009 – 2011	optimizations by Josh Bloch, Jon Bentley, many others
28 July 2011	**public release** of Java 7 with Yaroslavskiy's Quicksort



Running Time Experiments

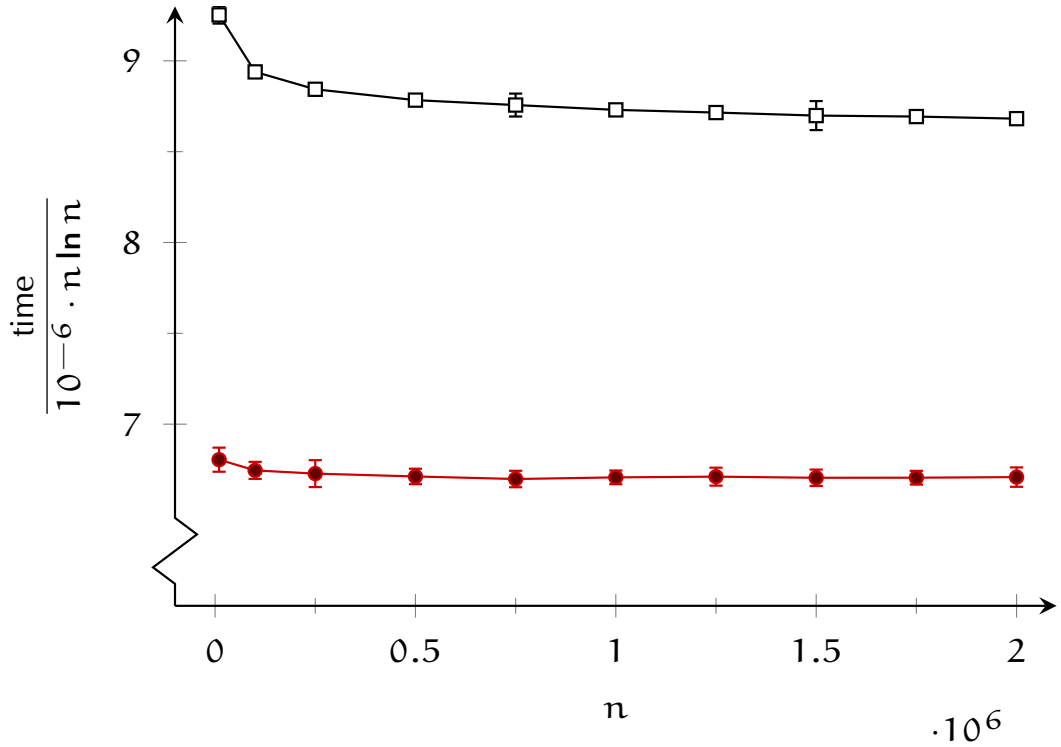
Why switch to new, unknown algorithm?



Normalized Java runtimes (in *ms*). Average and standard deviation of 1000 random permutations per size.

Running Time Experiments

Why switch to new, unknown algorithm? Because it is faster!



Normalized Java runtimes (in *ms*). Average and standard deviation of 1000 random permutations per size.

Dual Pivot Quicksort

- High Level Algorithm:
  1. Partition array arround two pivots $p \leqslant q$.
  2. Sort 3 subarrays recursively.

How to do partitioning?

Dual Pivot Quicksort

- High Level Algorithm:
  1. Partition array arround two pivots $p \leqslant q$.
  2. Sort 3 subarrays recursively.

How to do partitioning?

1. For each element x, determine its **class**
   - **small** for $x < p$
   - **medium** for $p < x < q$
   - **large** for $q < x$

   by comparing x to p **and/or** q
2. Arrange elements according to classes 

Dual Pivot Quicksort — Previous Work

Using two pivots is not a new idea!

- **Robert Sedgewick, 1975**
  - in-place dual pivot Quicksort implementation
  - **more** comparisons and swaps than classic Quicksort
- **Pascal Hennequin, 1991**
  - list-based Quicksort with r pivots
  - $r = 2 \rightsquigarrow$ **same** #comparisons as classic Quicksort ($\sim 2\,n \ln n$)

Dual Pivot Quicksort — Previous Work

Using two pivots is not a new idea!

- **Robert Sedgewick, 1975**
  - in-place dual pivot Quicksort implementation
  - **more** comparisons and swaps than classic Quicksort
- **Pascal Hennequin, 1991**
  - list-based Quicksort with r pivots
  - $r = 2 \rightsquigarrow$ **same** #comparisons as classic Quicksort ($\sim 2\,n \ln n$)

$\rightsquigarrow$ *Using two pivots does not pay.*

Dual Pivot Quicksort — Comparison "Bound"

How many comparisons to determine classes (small , medium or large) ?

- Assume compare with p first (**Hennequin, 1991**).
  $\rightsquigarrow$ small elements need 1, others 2 comparisons

On average:

- $\frac{1}{3}$ small elements
- $\rightsquigarrow \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = \frac{5}{3}$ comparisons per element

$\rightsquigarrow$ **Any** partitioning method requires
$\frac{5}{3}(n-2) \sim \frac{20}{12} n$ comparisons, right?

Dual Pivot Quicksort — Comparison "Bound"

How many comparisons to determine classes (small , medium or large) ?

- Assume compare with p first (**Hennequin, 1991**).
  $\rightsquigarrow$ small elements need 1, others 2 comparisons

On average:

- $\frac{1}{3}$ small elements
- $\rightsquigarrow \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = \frac{5}{3}$ comparisons per element

$\rightsquigarrow$ **Any** partitioning method requires
$\frac{5}{3}(n-2) \sim \frac{20}{12} n$ comparisons, right?

Dual Pivot Quicksort — Comparison "Bound"

How many comparisons to determine classes (small , medium or large) ?

- Assume compare with p first (**Hennequin, 1991**).
  $\rightsquigarrow$ small elements need 1, others 2 comparisons

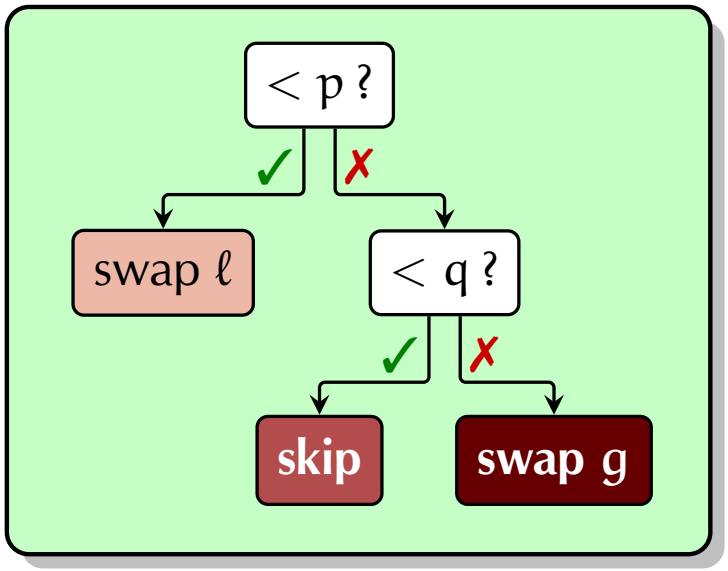
On average:

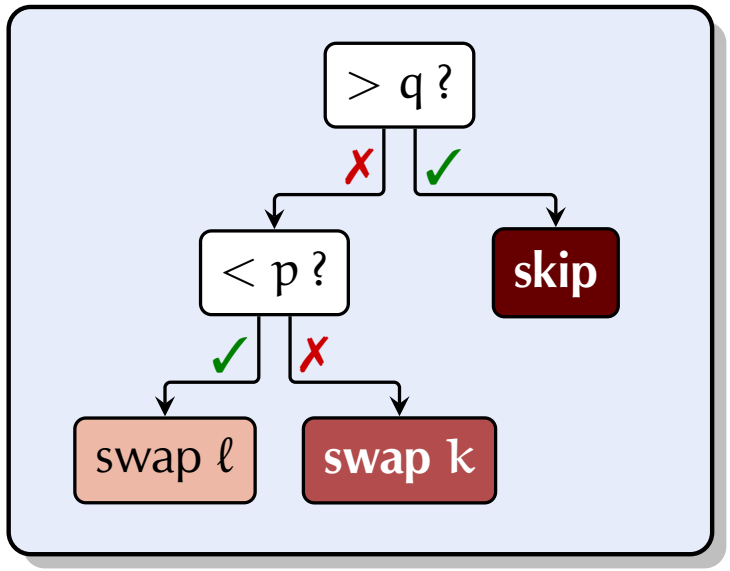
- $\frac{1}{3}$ small elements
- $\rightsquigarrow \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = \frac{5}{3}$ comparisons per element

$\rightsquigarrow$ **Any** partitioning method requires
$\frac{5}{3}(n-2) \sim \frac{20}{12} n$ comparisons, right?

- **No!** (Stay tuned . . .)

Yaroslavskiy's Algorithm



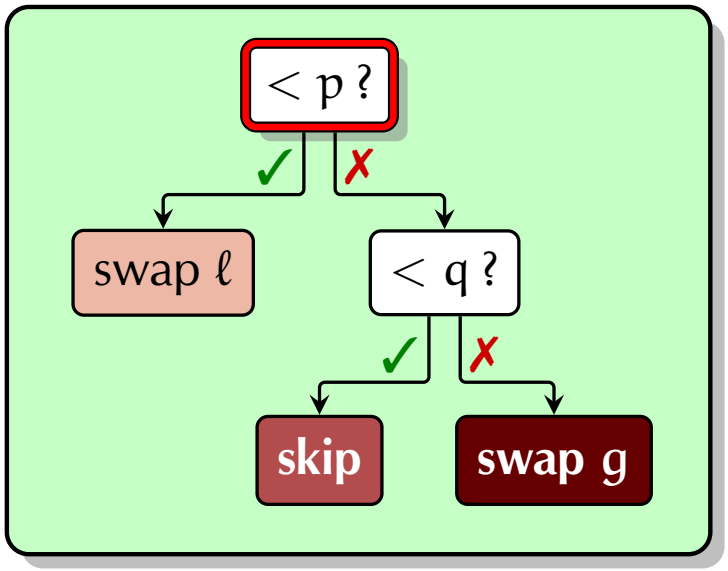


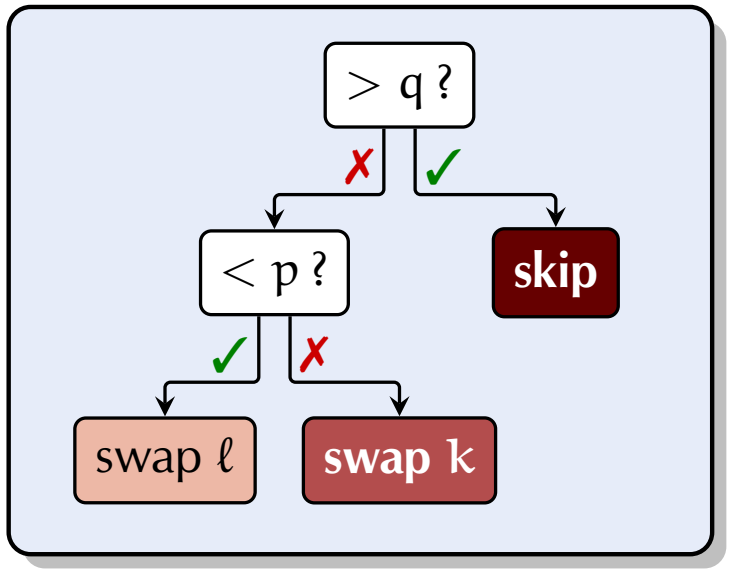


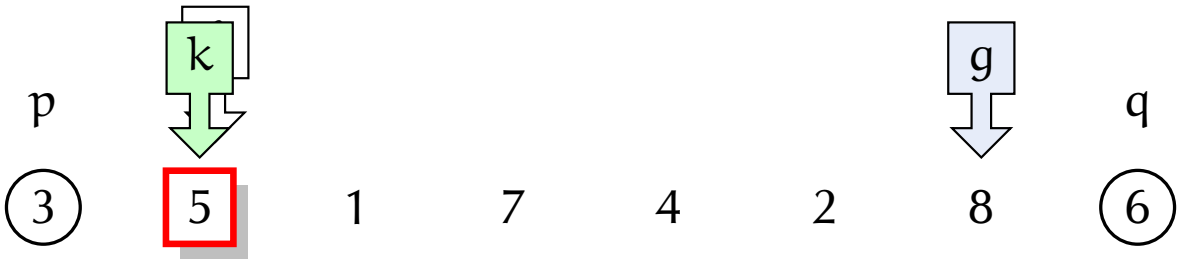
Invariant:



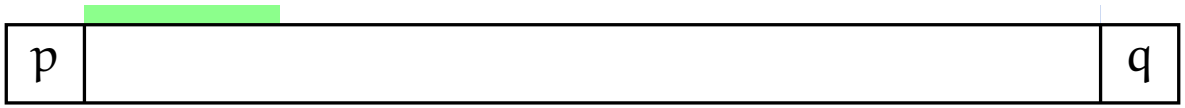
Yaroslavskiy's Algorithm



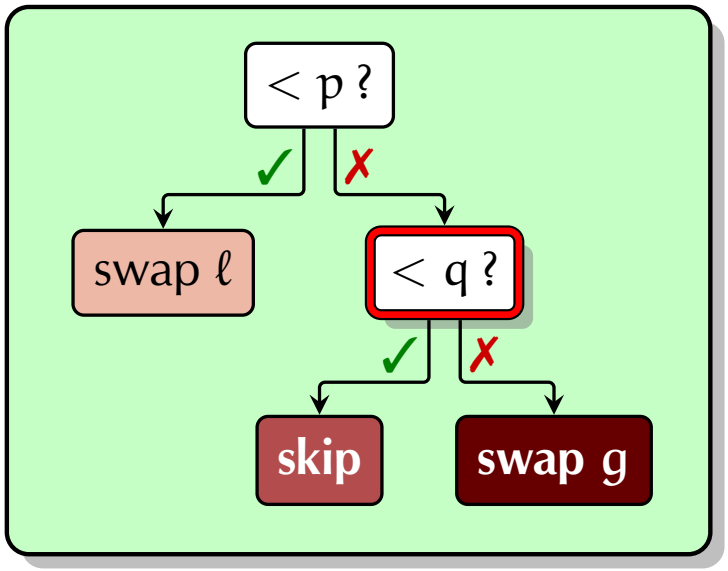


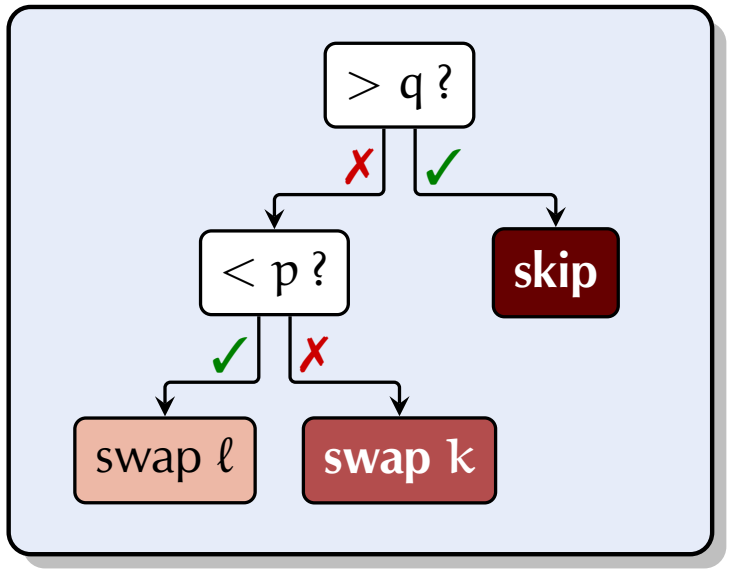


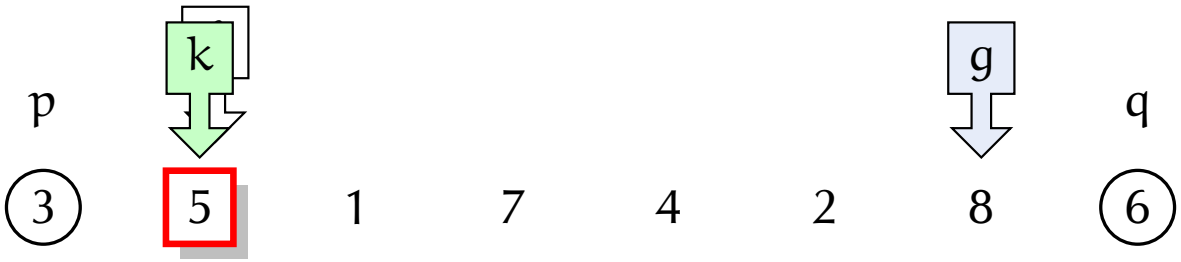
Invariant:



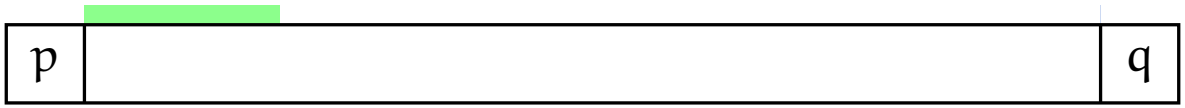
Yaroslavskiy's Algorithm



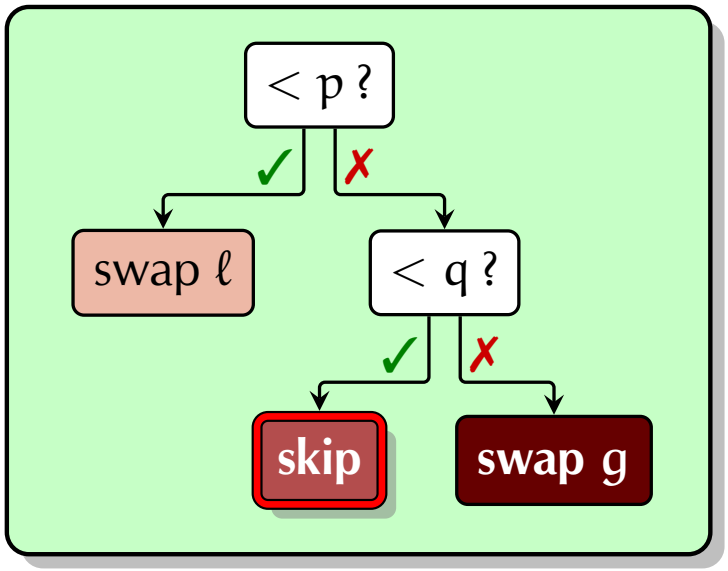


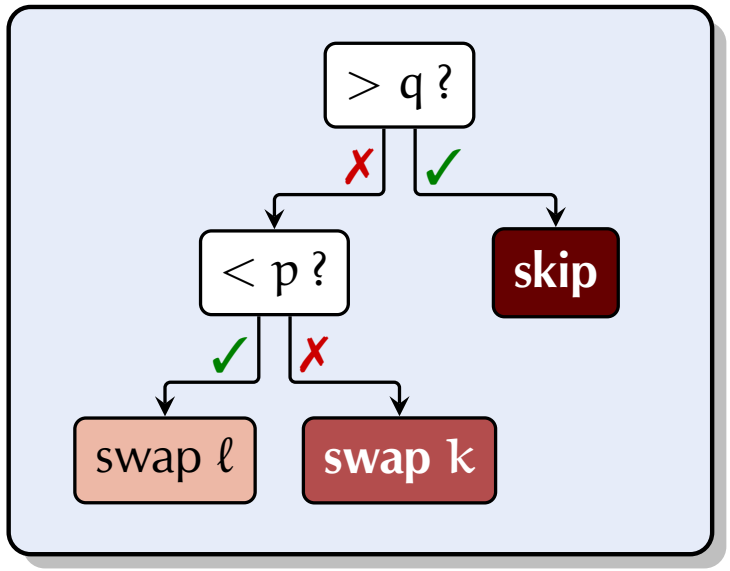


Invariant:



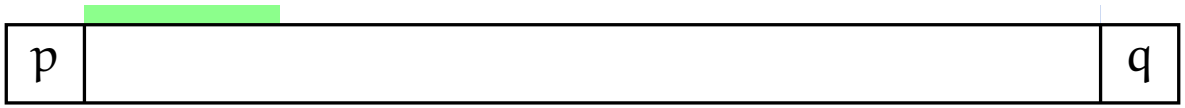
Yaroslavskiy's Algorithm



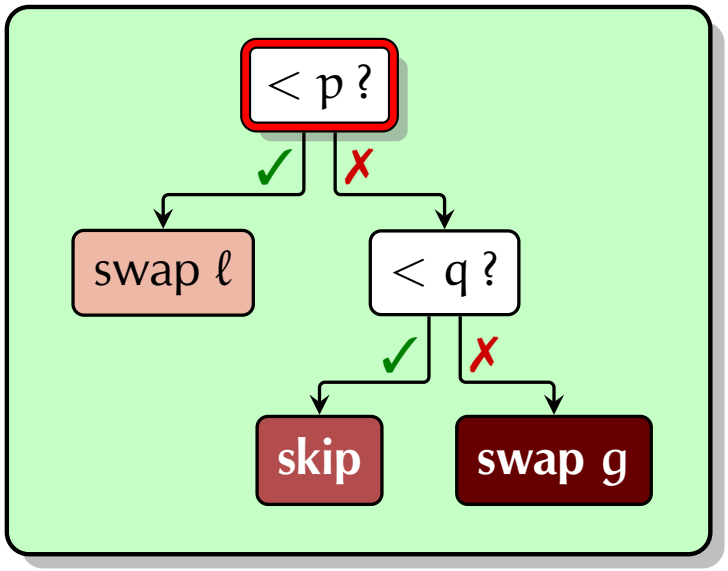


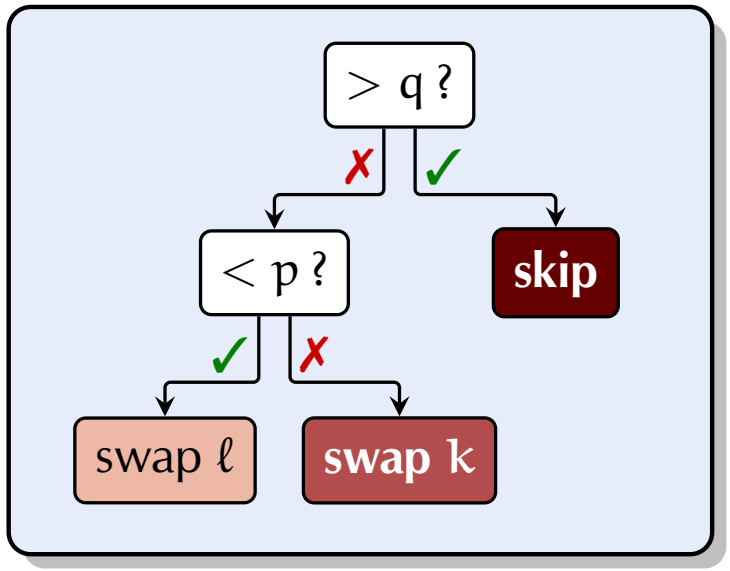


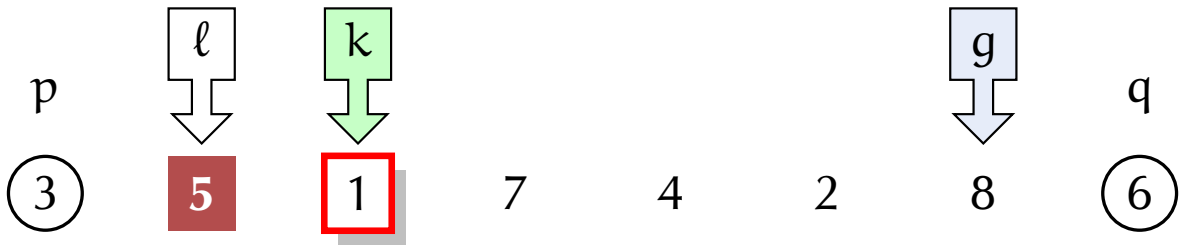
Invariant:



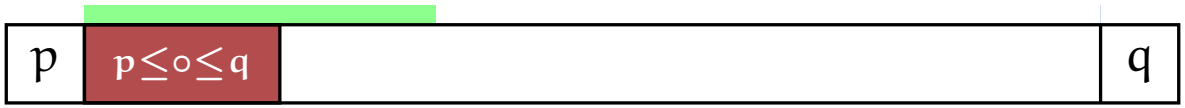
Yaroslavskiy's Algorithm



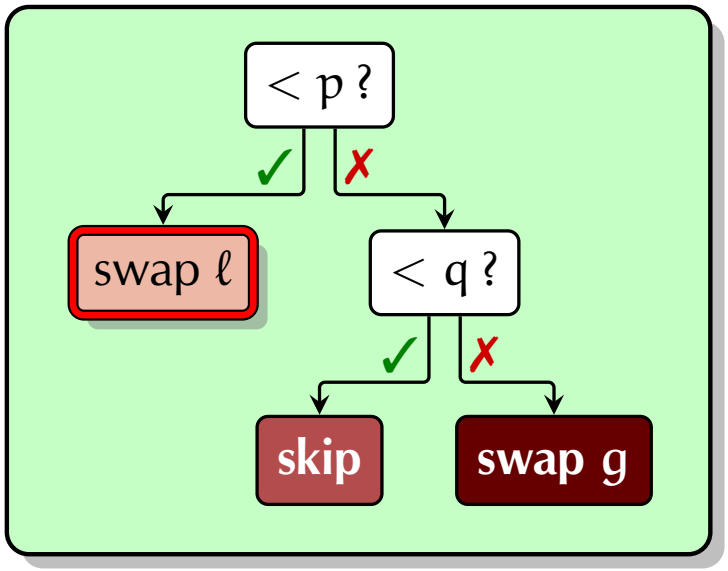


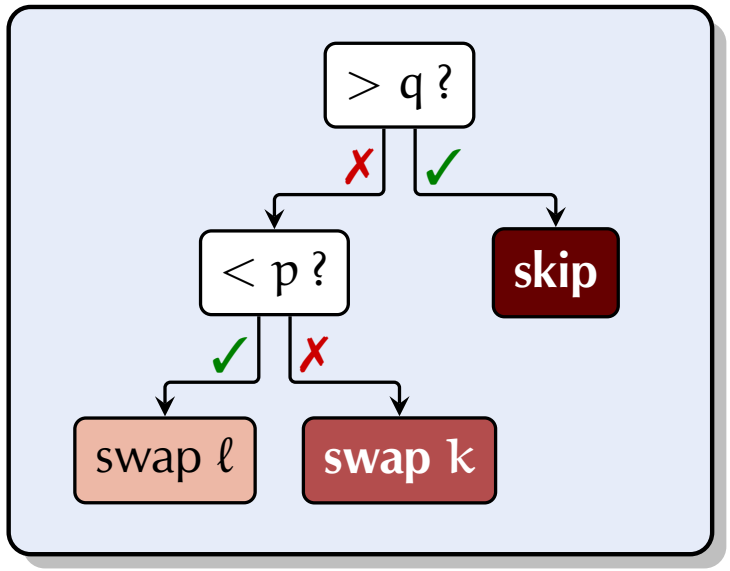


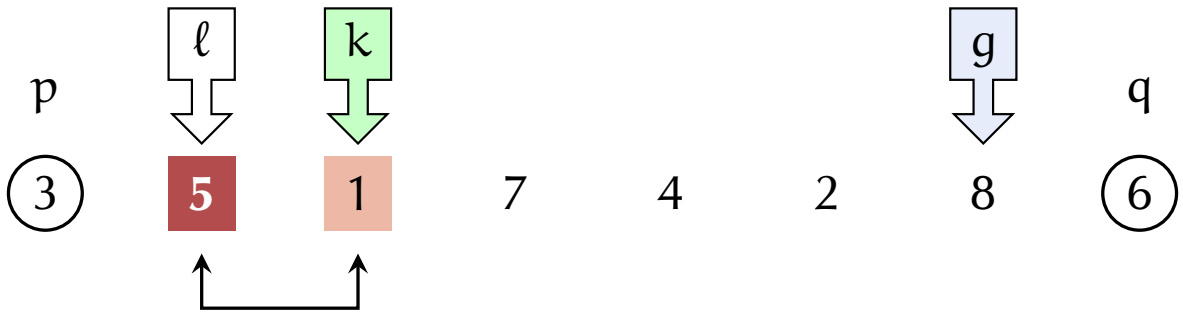
Invariant:



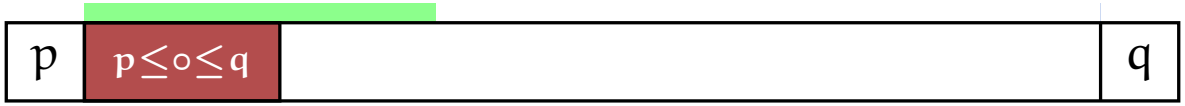
Yaroslavskiy's Algorithm



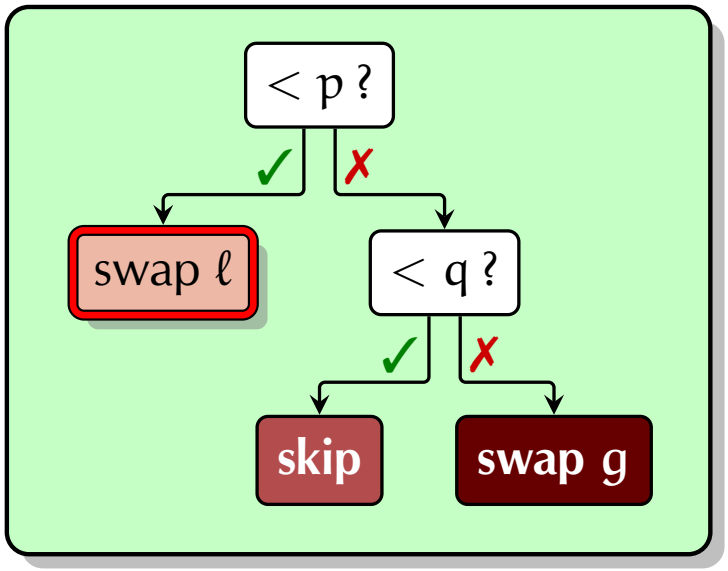


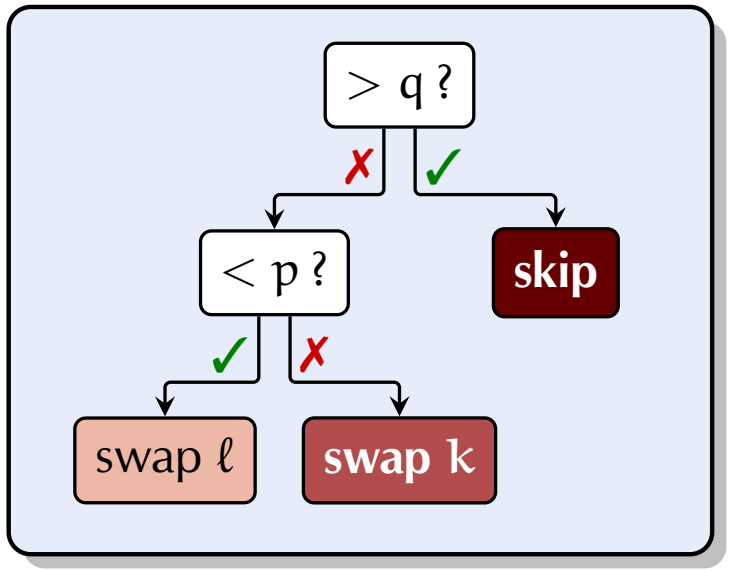


Invariant:



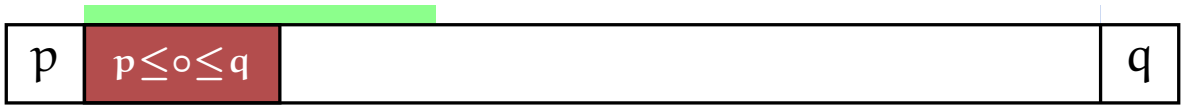
Yaroslavskiy's Algorithm



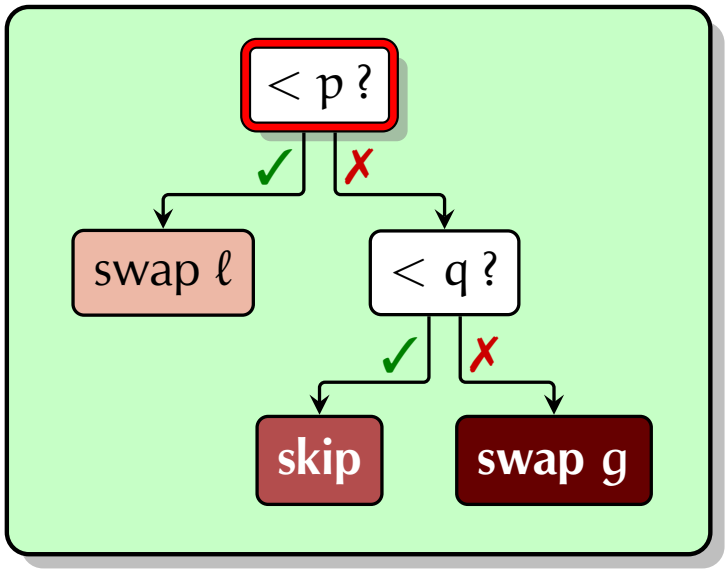


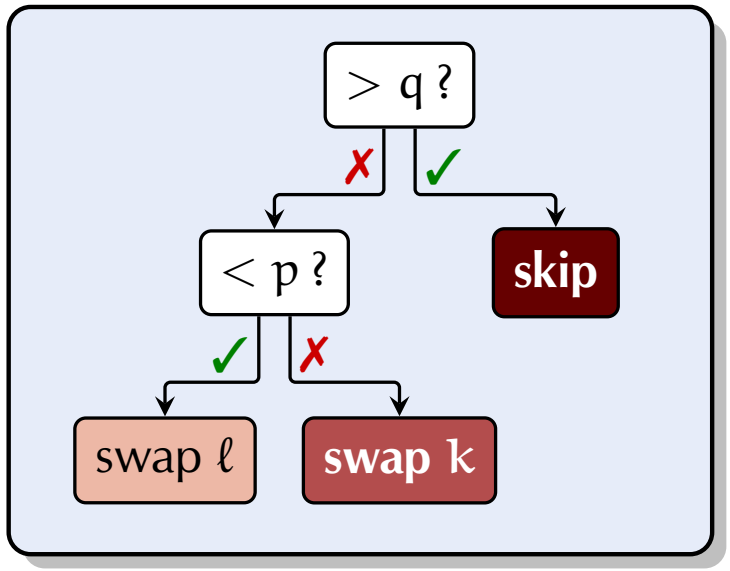


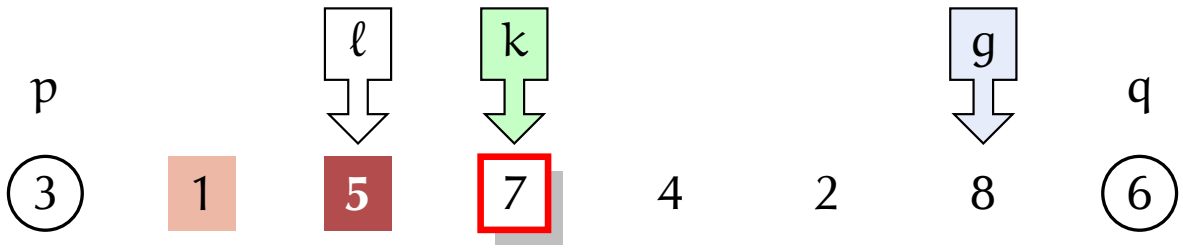
Invariant:



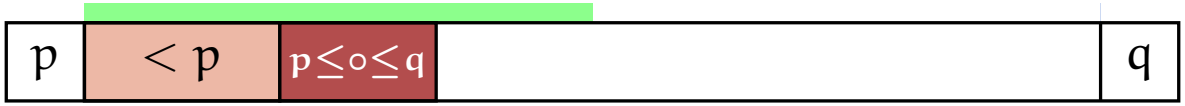
Yaroslavskiy's Algorithm



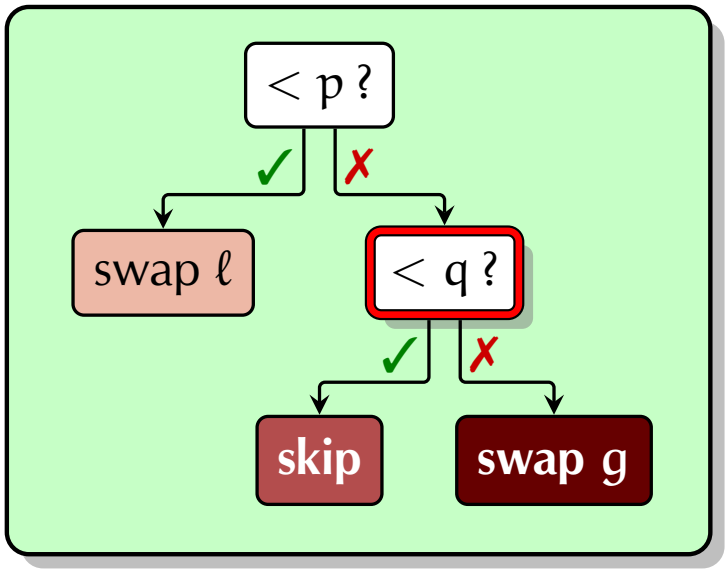


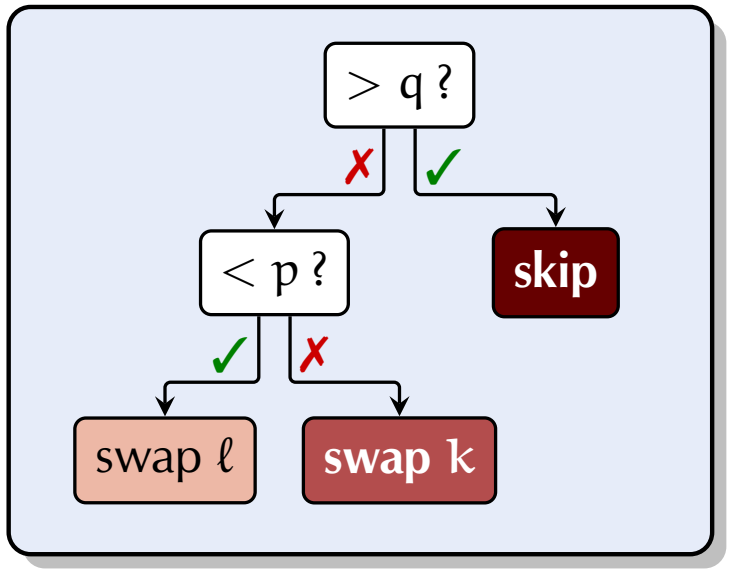


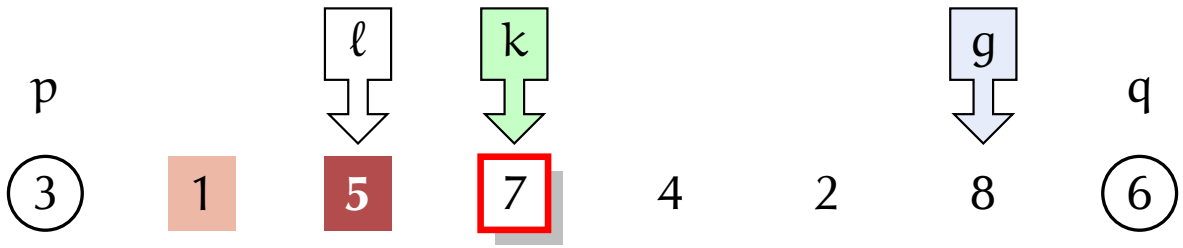
Invariant:



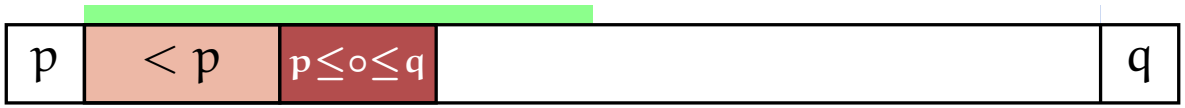
Yaroslavskiy's Algorithm



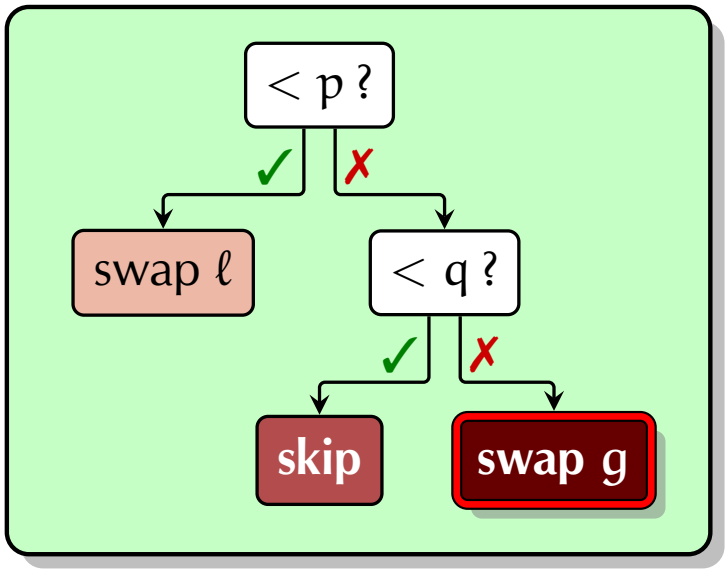


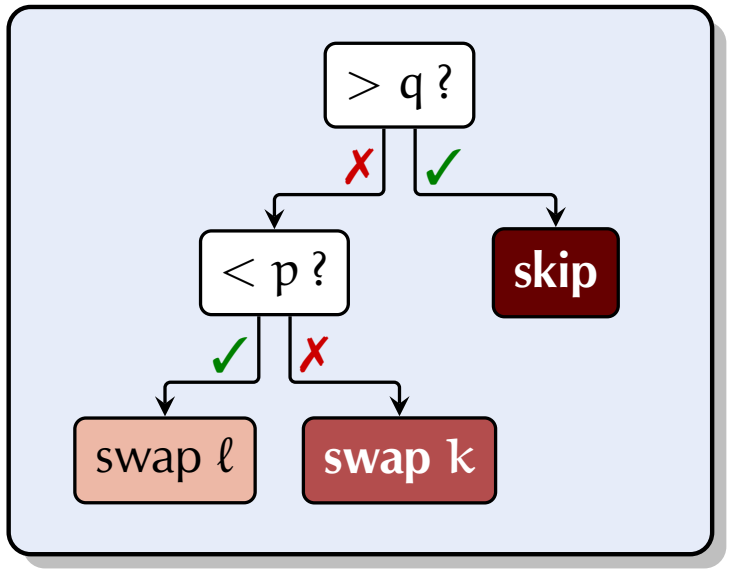


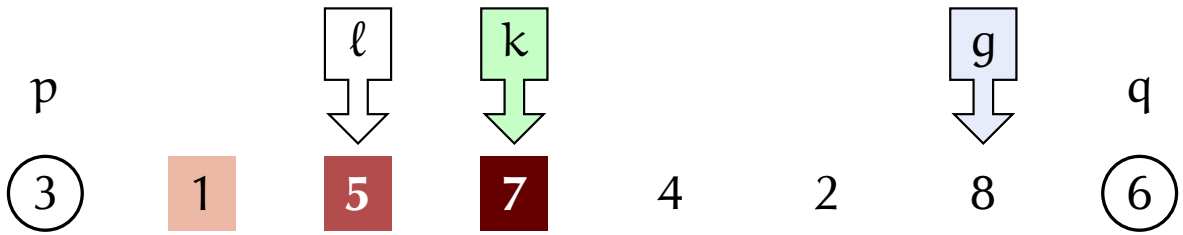
Invariant:



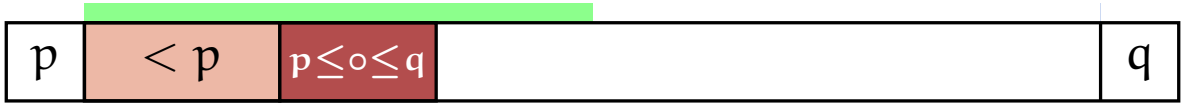
Yaroslavskiy's Algorithm



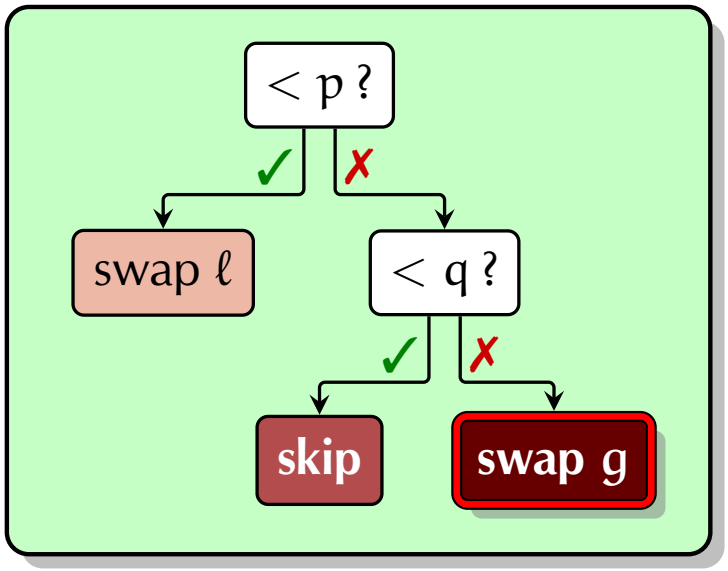


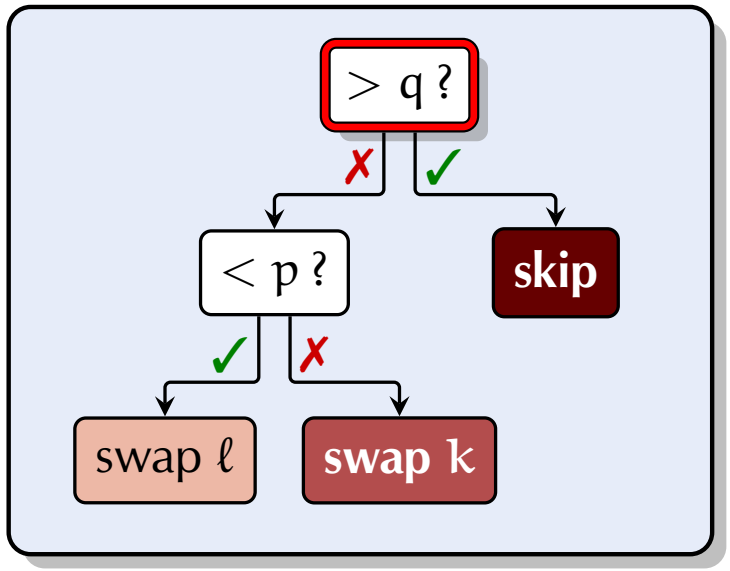


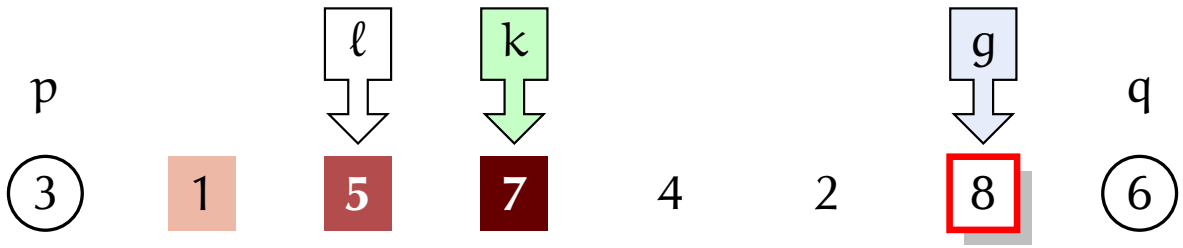
Invariant:



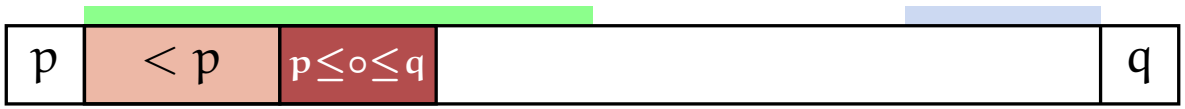
Yaroslavskiy's Algorithm



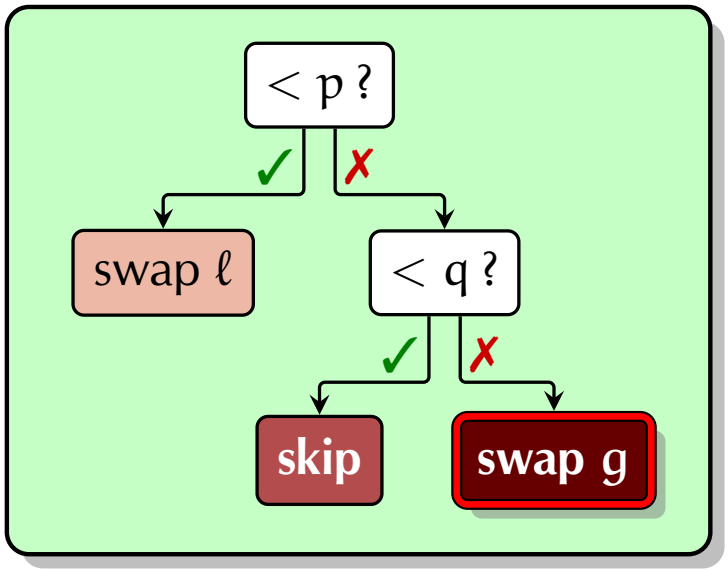


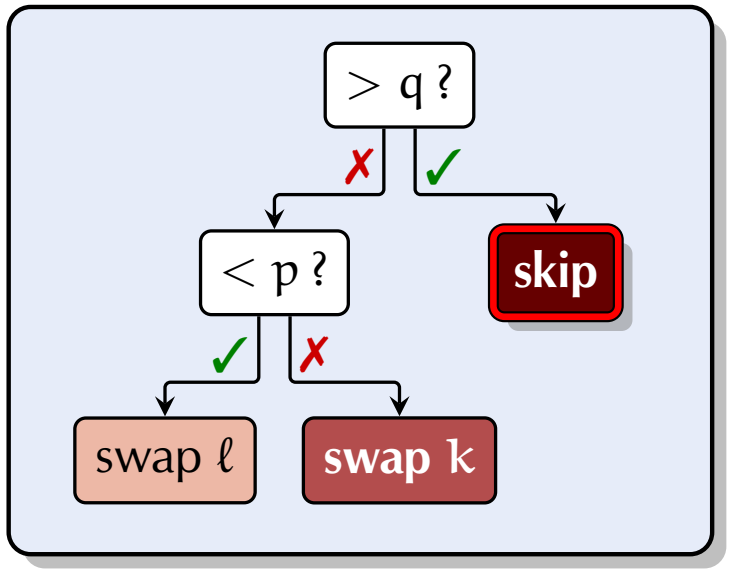


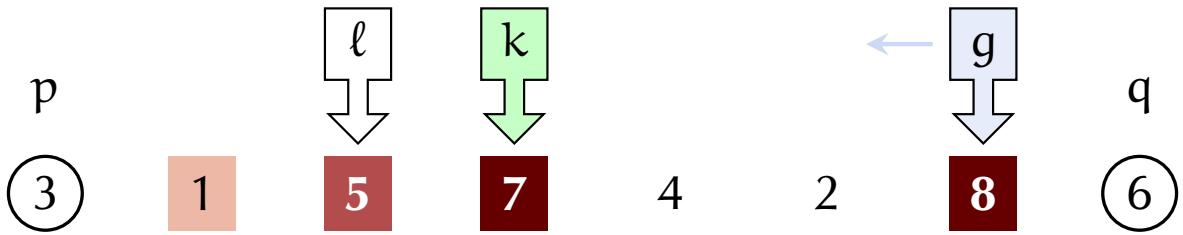
Invariant:



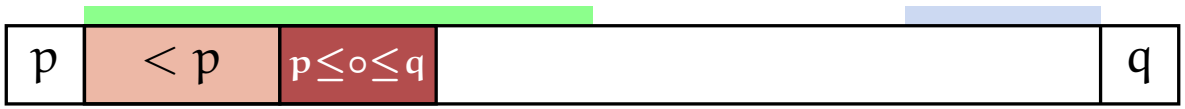
Yaroslavskiy's Algorithm



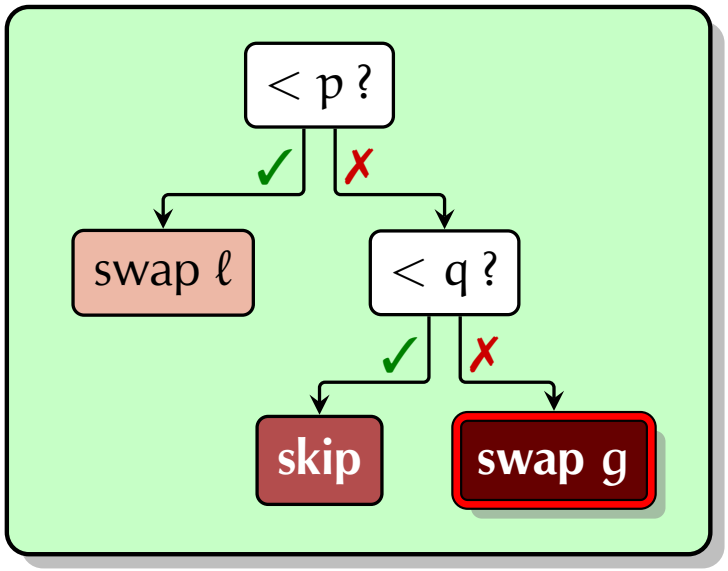


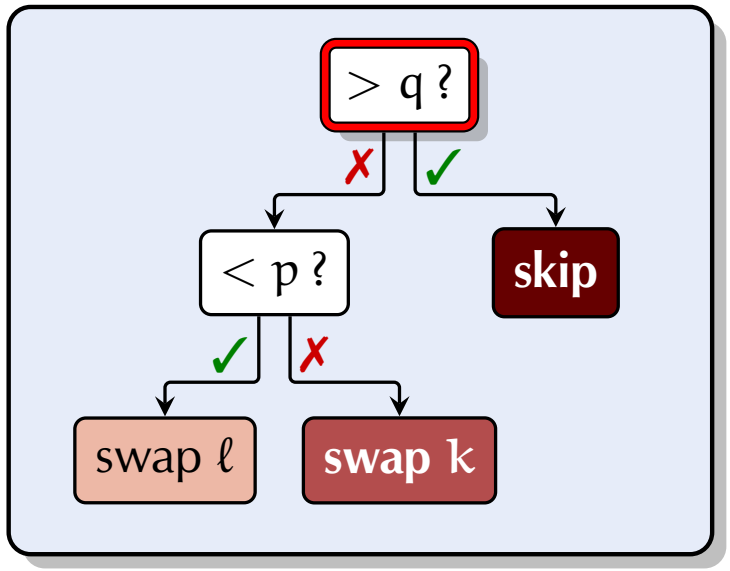


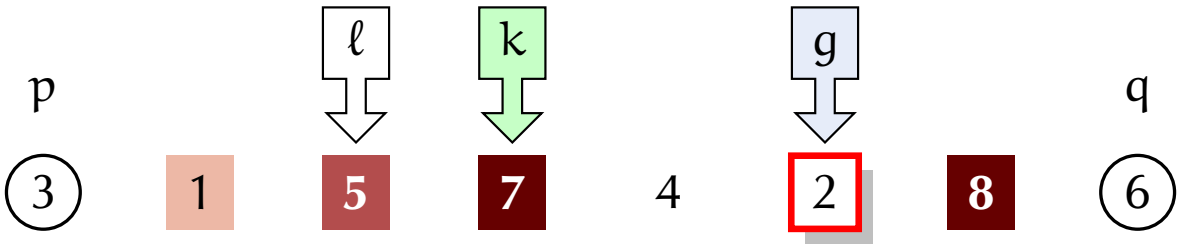
Invariant:



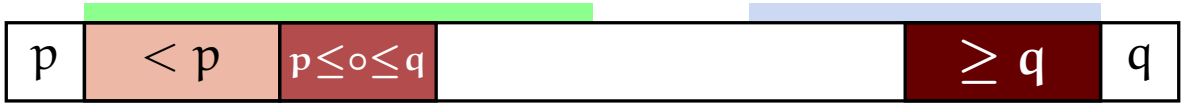
Yaroslavskiy's Algorithm



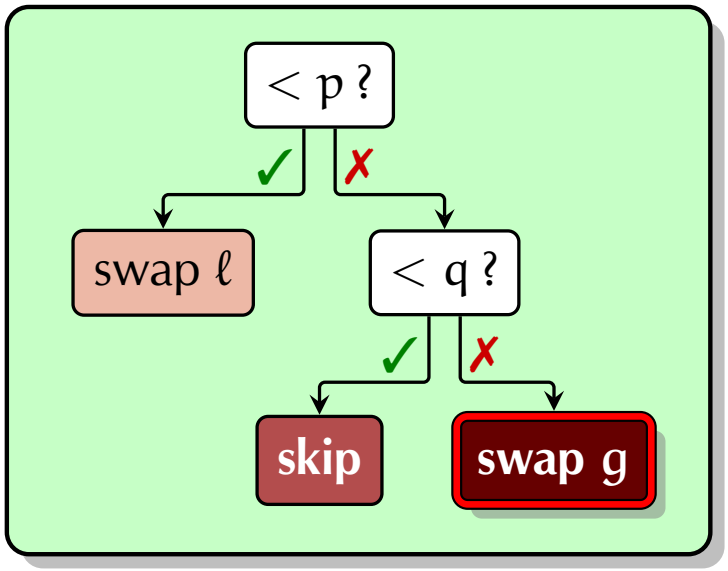


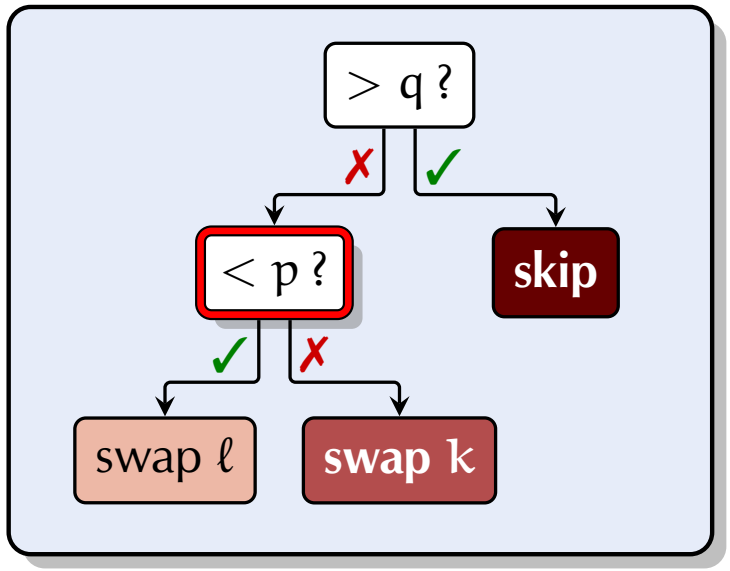


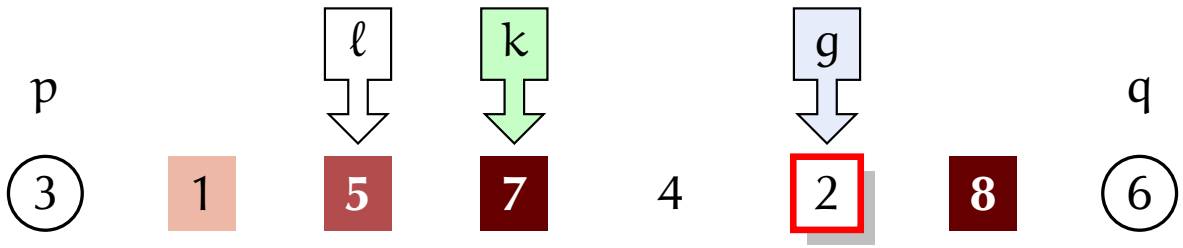
Invariant:



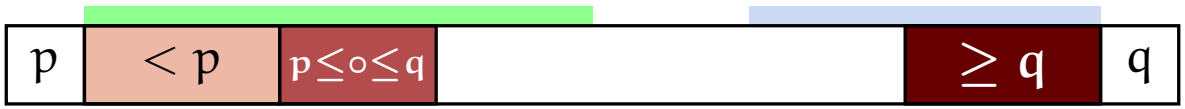
Yaroslavskiy's Algorithm



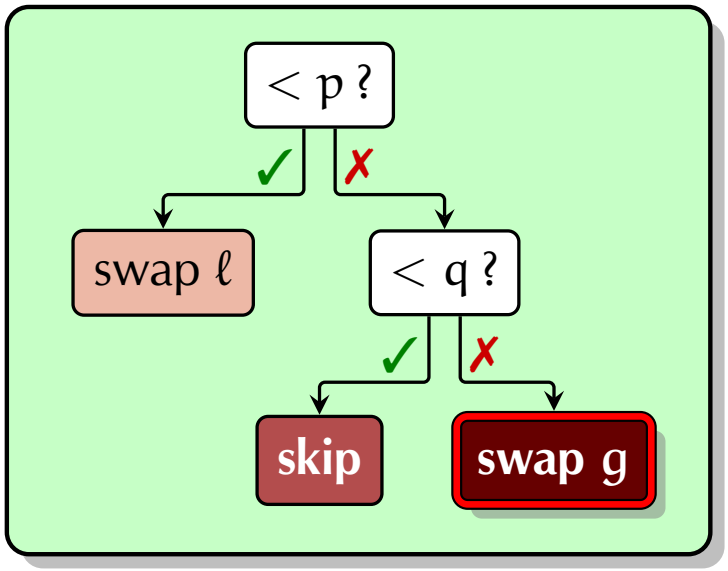


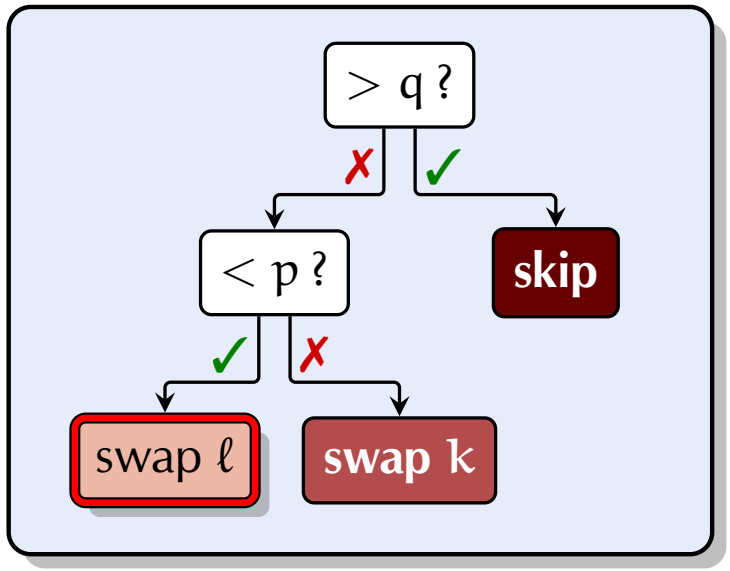


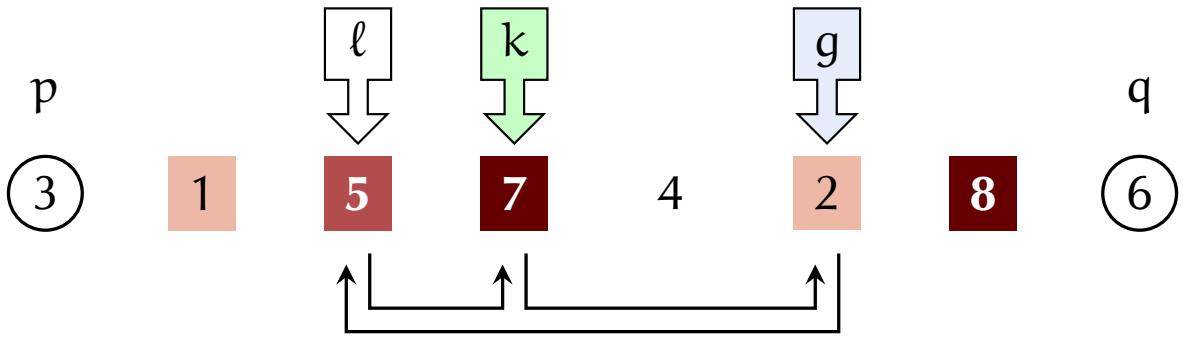
Invariant:



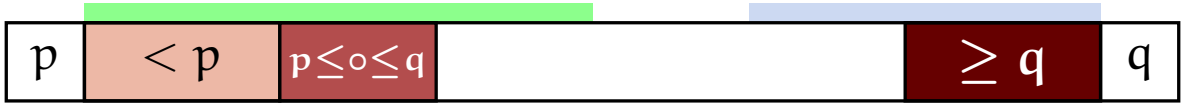
Yaroslavskiy's Algorithm



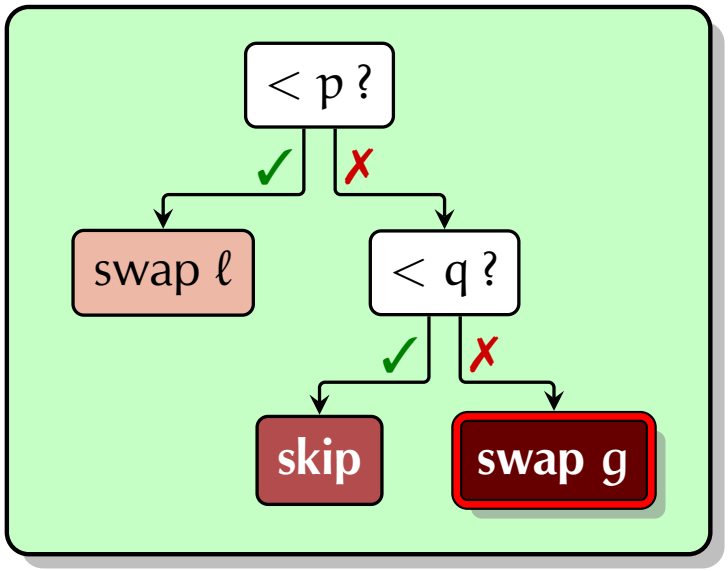


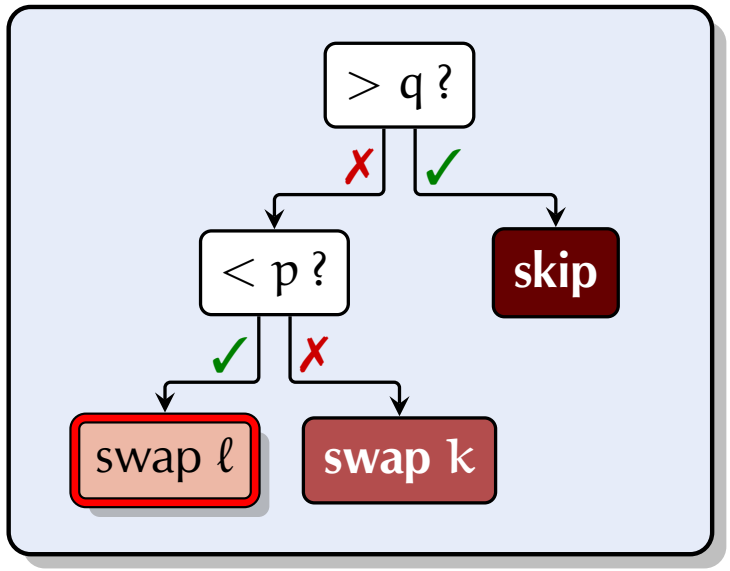


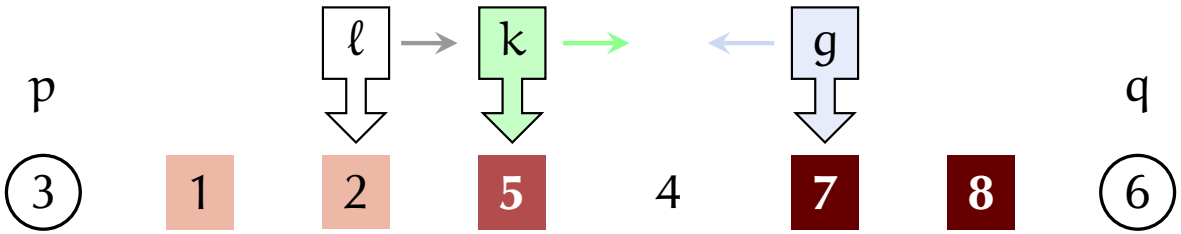
Invariant:



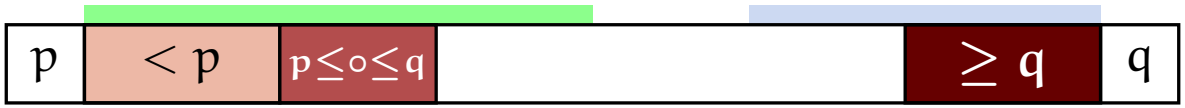
Yaroslavskiy's Algorithm



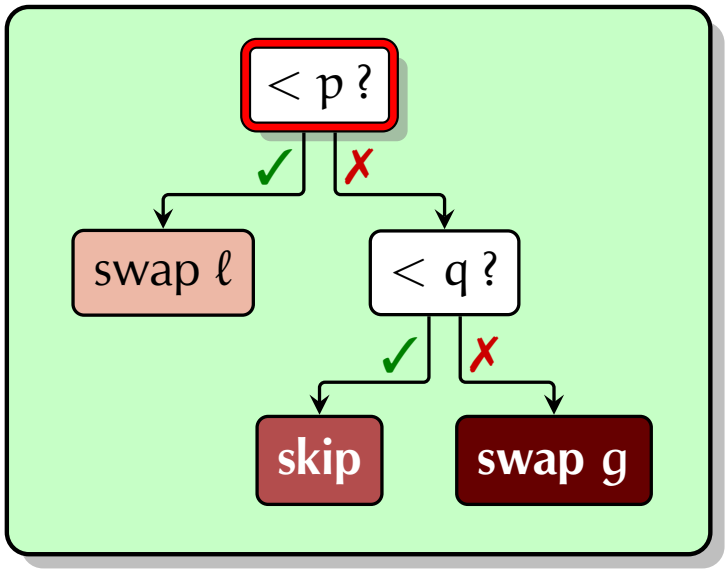


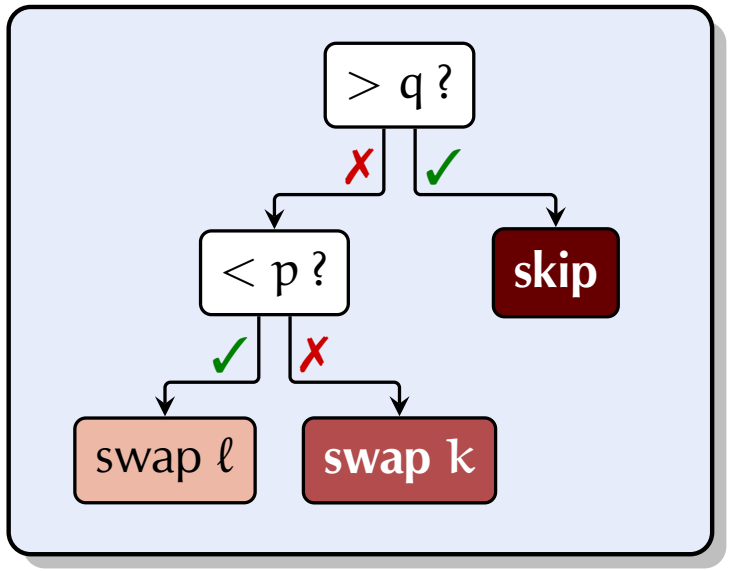


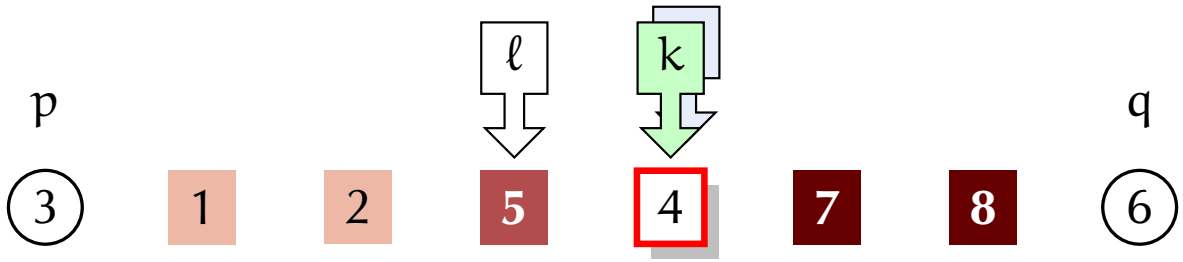
Invariant:



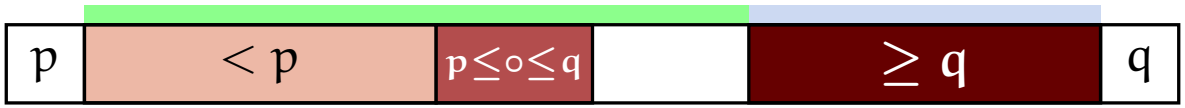
Yaroslavskiy's Algorithm



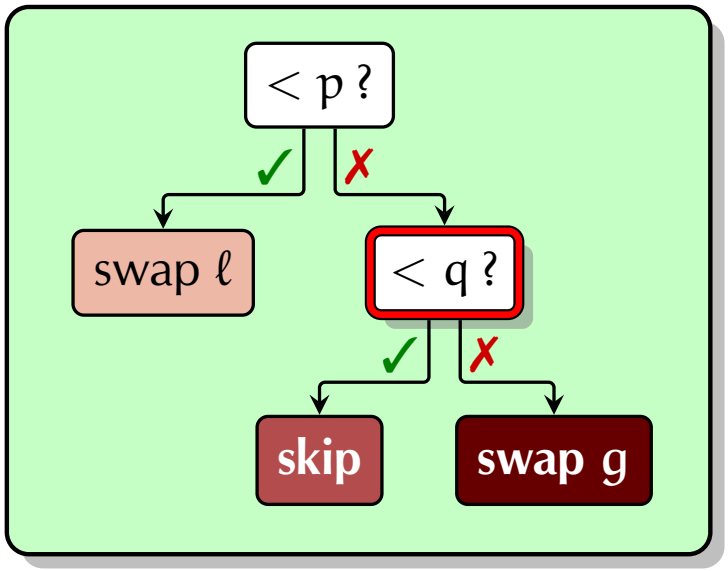


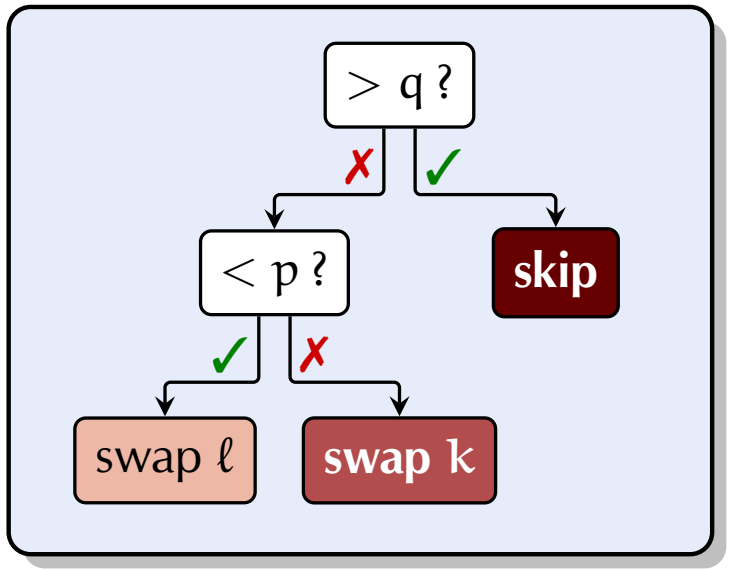


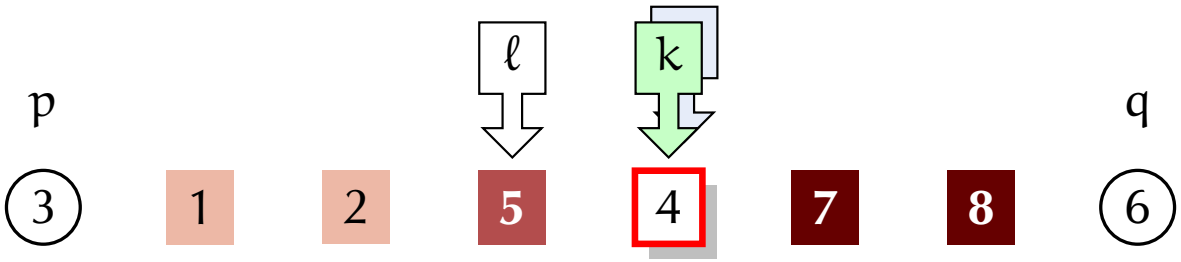
Invariant:



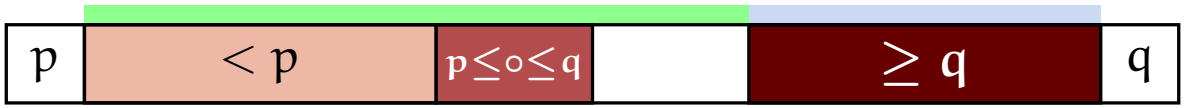
Yaroslavskiy's Algorithm



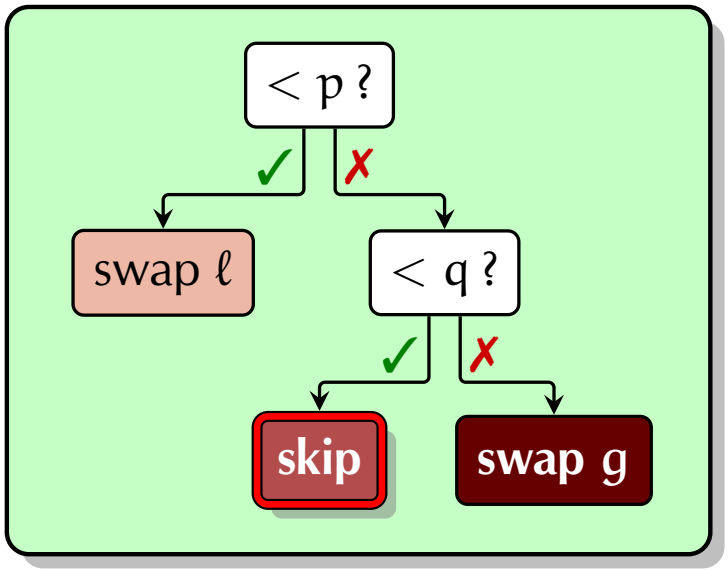


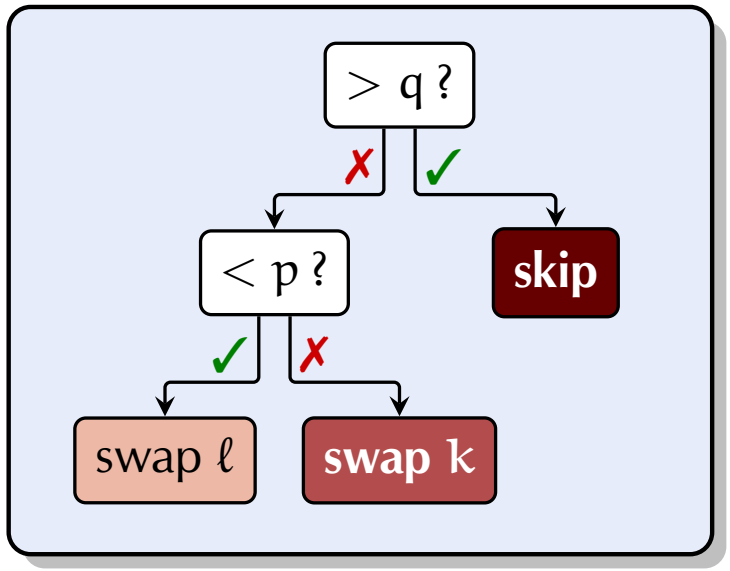


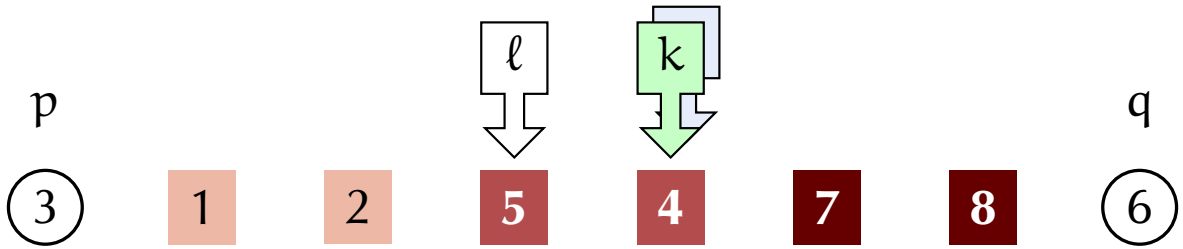
Invariant:



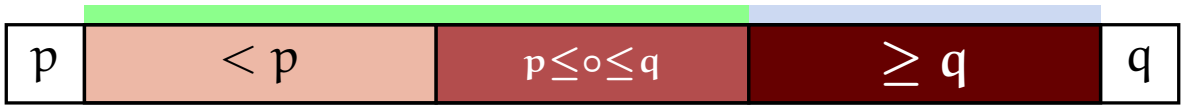
Yaroslavskiy's Algorithm



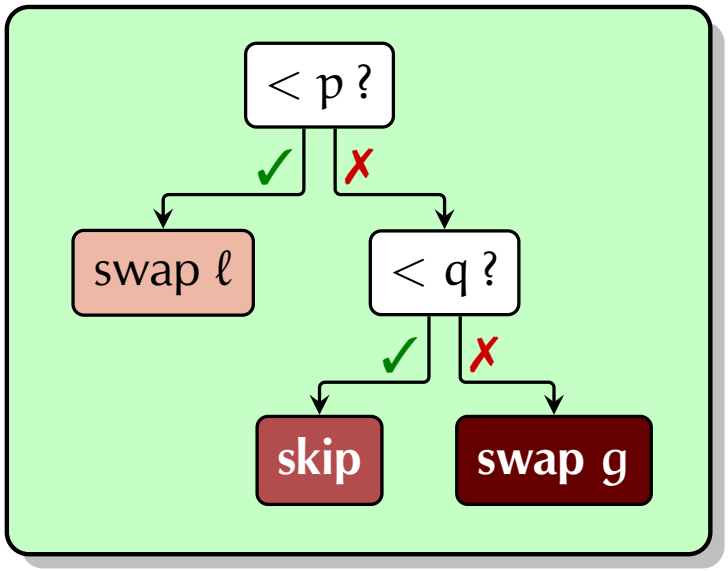


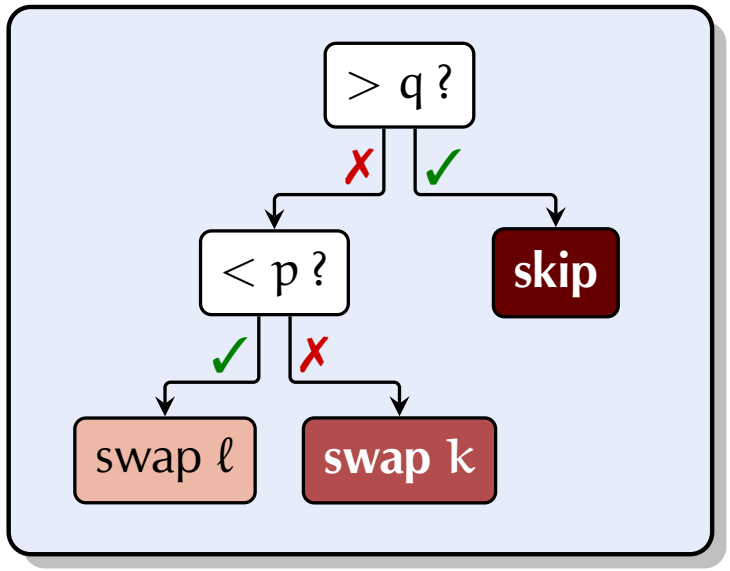


Invariant:



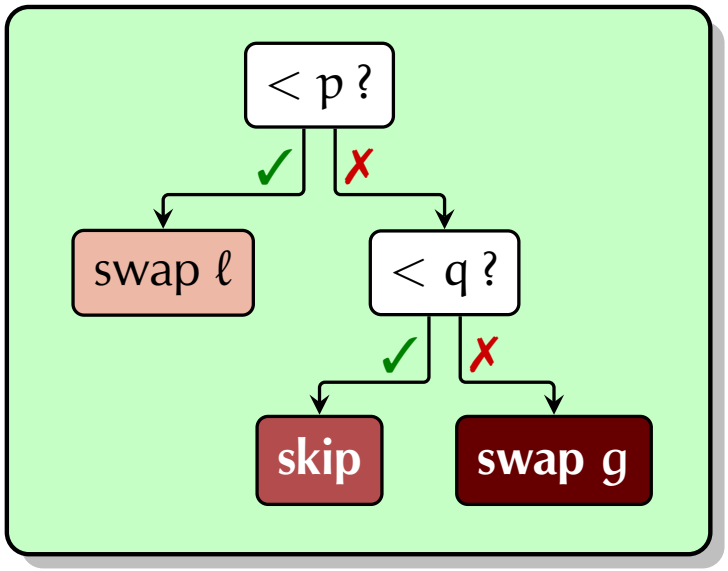
Yaroslavskiy's Algorithm

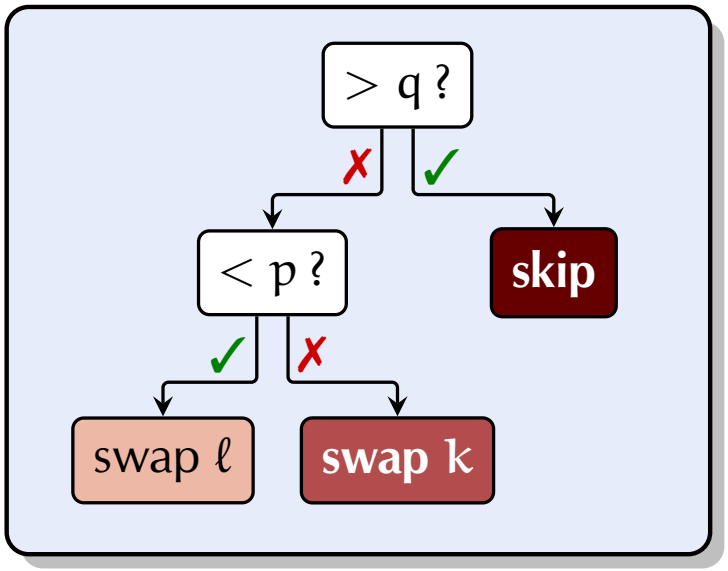


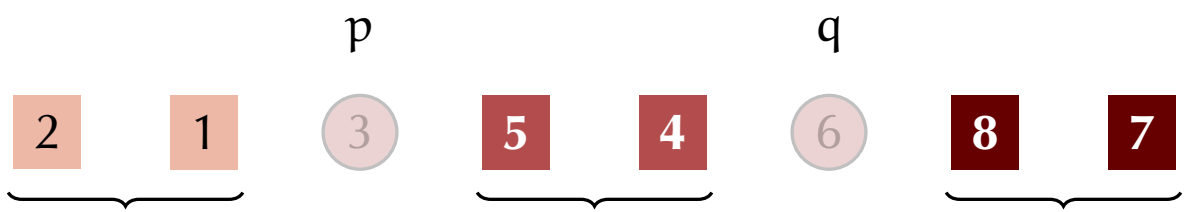




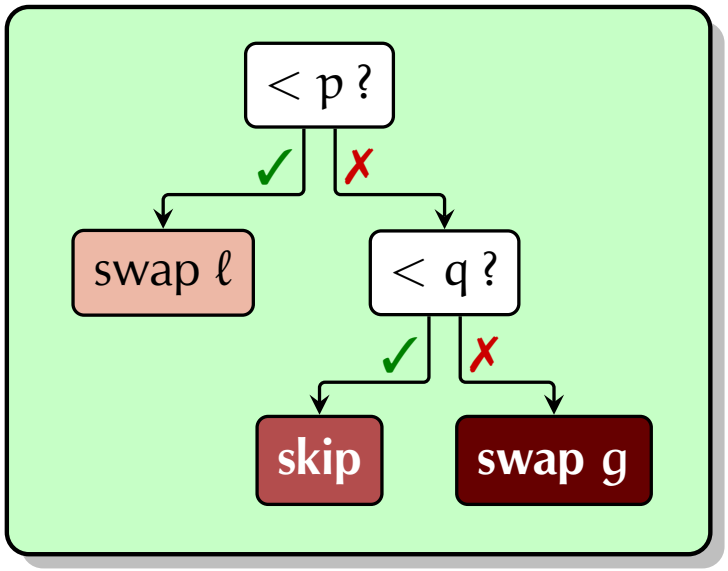
Yaroslavskiy's Algorithm

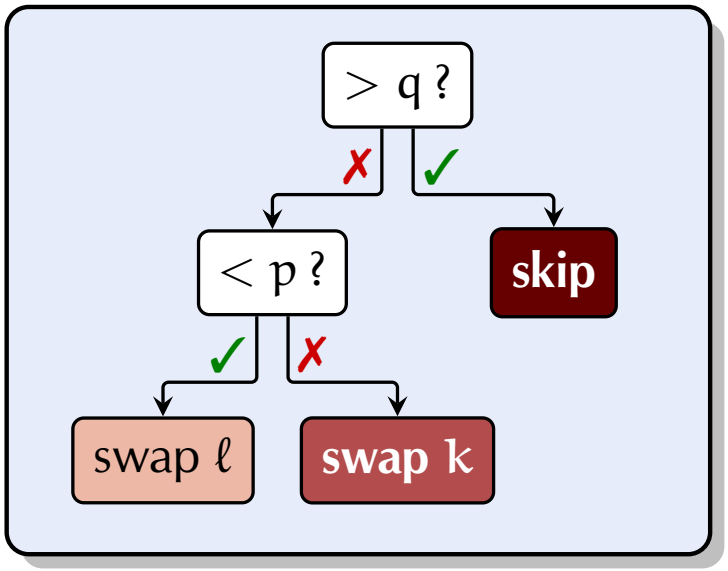






Yaroslavskiy's Algorithm

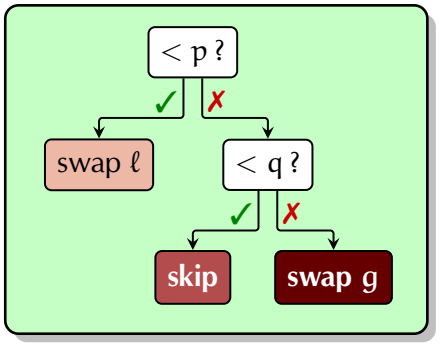


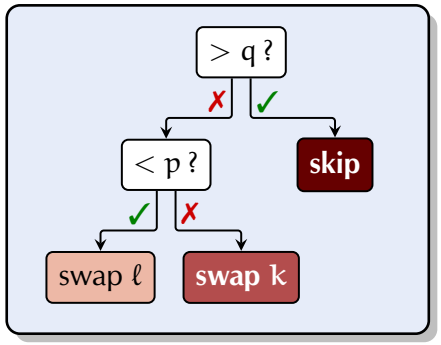




Asymmetries in #Comparisons

How many comparisons used?





$\rightsquigarrow$ Cheap elements (**1 cmp.**):

- small in k's range ($s @ \mathcal{K}$)
- large in g's range ($l @ \mathcal{G}$)

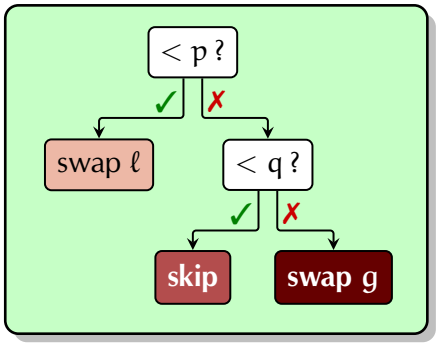
Key Observation: sizes of classes and ranges are correlated:

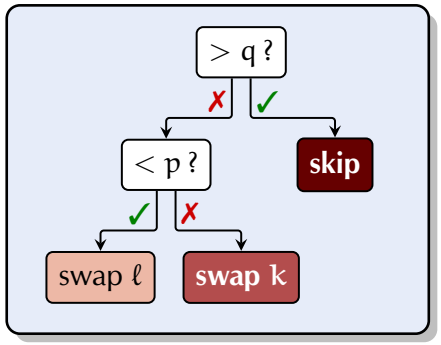
- many small elements $\rightsquigarrow$ k's range big
- many large elements $\rightsquigarrow$ g's range big

$\Big\}$ $> \frac{1}{3}$ cheap elements (on average)

Asymmetries in #Comparisons

How many comparisons used?

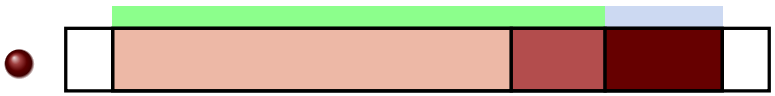




$\rightsquigarrow$ Cheap elements (**1 cmp.**):

- small in k's range ($s @ \mathcal{K}$)
- large in g's range ($l @ \mathcal{G}$)

Key Observation: sizes of classes and ranges are correlated:



many small elements $\rightsquigarrow$ k's range big



many large elements $\rightsquigarrow$ g's range big

$> \frac{1}{3}$ cheap elements (on average)

Comparisons per Partitioning Step

- Can give full distribution of s@$\mathcal{K}$ and l@$\mathcal{G}$ (hypergeometric conditional on p and q)

$\rightsquigarrow$ Expected #cmps:

$$
\begin{aligned}
c_n \quad &\sim \quad 2n && \text{two cmps per element} \\
&\quad -\tfrac{3}{12}n && \text{small in } k\text{'s range} \\
&\quad -\tfrac{1}{6}n && \text{large in } g\text{'s range} \\
\hline
&= \quad \tfrac{19}{12}n
\end{aligned}
$$

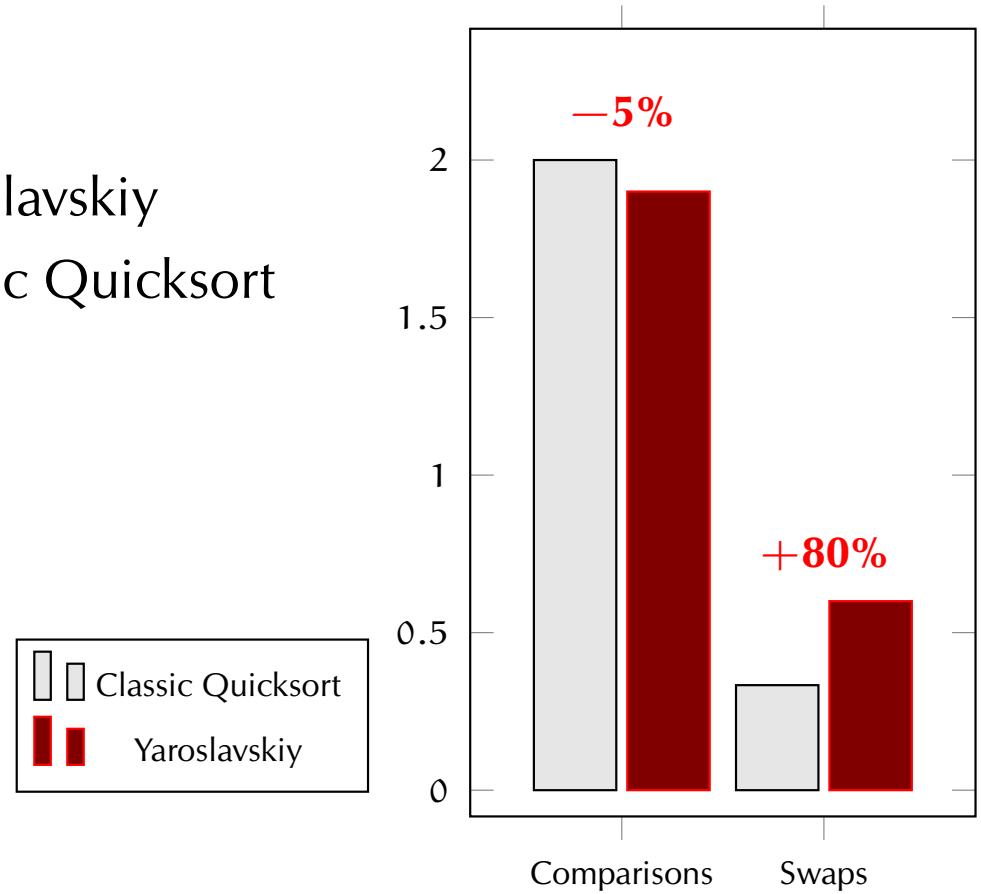
Recall: "lower bound" was $\frac{20}{12}n$.

Results — Basic Operations

Solving dual-pivot Quicksort recurrence gives:

- **Comparisons:**
  - Yaroslavskiy needs only $\sim \mathbf{1.9}\, n \ln n$ comparisons on average.
  - Classic Quicksort: $\sim \mathbf{2}\, n \ln n$
- **Swaps:**
  - $\sim \mathbf{0.6}\, n \ln n$ swaps for Yaroslavskiy
  - $\sim \mathbf{0.\overline{3}}\, n \ln n$ swaps for classic Quicksort

$\rightsquigarrow$ *Which algorithm is faster?*
Result remains inconclusive!

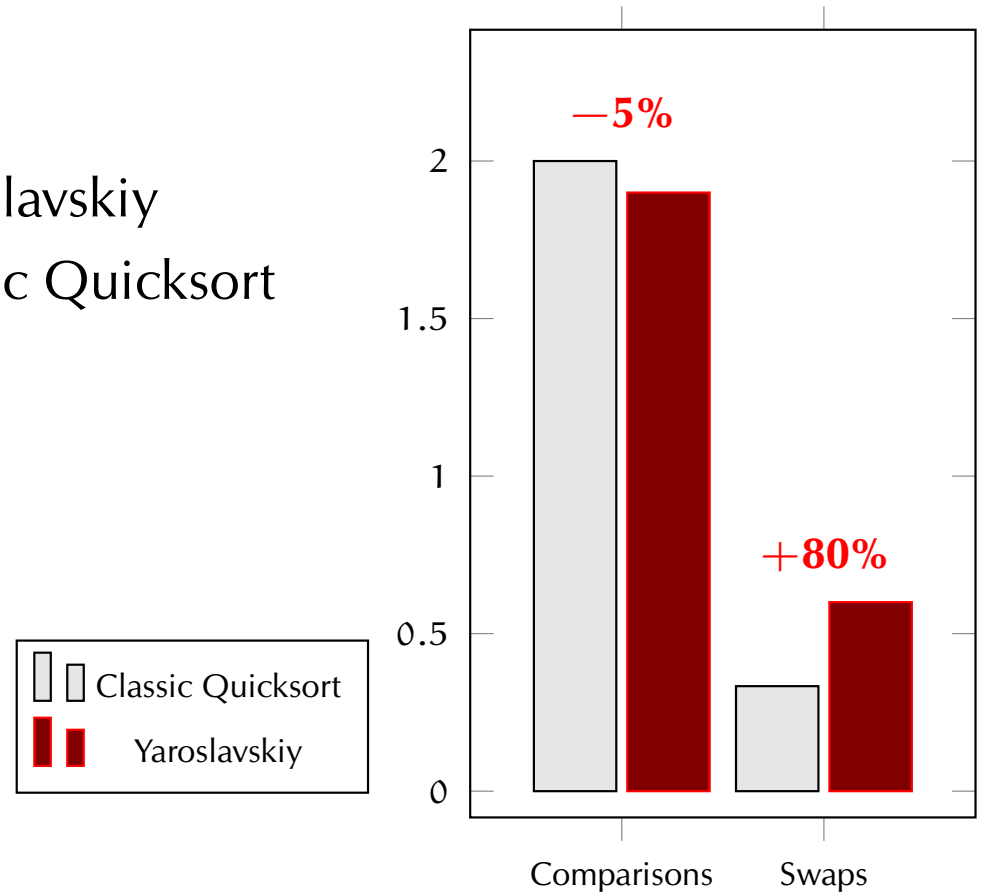


Results — Basic Operations

Solving dual-pivot Quicksort recurrence gives:

- **Comparisons:**
  - Yaroslavskiy needs only $\sim \mathbf{1.9}\, n \ln n$ comparisons on average.
  - Classic Quicksort: $\sim \mathbf{2}\, n \ln n$
- **Swaps:**
  - $\sim \mathbf{0.6}\, n \ln n$ swaps for Yaroslavskiy
  - $\sim \mathbf{0.\overline{3}}\, n \ln n$ swaps for classic Quicksort

$\rightsquigarrow$ *Which algorithm is faster?*
Result remains inconclusive!



Instruction Counts à la Knuth

*How to force **conclusive** analysis?*

- More **detailed** cost model!
- old days: Knuth's MIX instructions
- in the Java universe: **Java Bytecode** instructions
- #Bytecodes highly correlated with running time

Instruction Counts à la Knuth

*How to force **conclusive** analysis?*

- More **detailed** cost model!
- old days: Knuth's MIX instructions
- in the Java universe: **Java Bytecode** instructions
- #Bytecodes highly correlated with running time

Results — Primitive Instructions

Results:

- Yaroslavskiy's algorithm: $\sim \mathbf{21.7}\, n \ln n$ Bytecodes
- Classic Quicksort: $\sim \mathbf{18}\, n \ln n$ Bytecodes

Similar for Knuth's new mythical machine **MMIX**

- Yaroslavskiy: $\sim (\mathbf{13.1}\upsilon + \mathbf{2.8}\mu)\, n \ln n$
- Classic Quicksort: $\sim (\mathbf{11}\upsilon + \mathbf{2.}\overline{\mathbf{6}}\mu)\, n \ln n$

υ ("oops"): CPU cycles; μ ("mems"): memory accesses

$\rightsquigarrow$ *Classic Quicksort significantly* ***better*** *in all these measures . . .*

$\rightarrow$ *These models ignore memory hierarchies; see Alex López-Ortiz' talk!*

Results — Primitive Instructions

Results:

- Yaroslavskiy's algorithm: $\sim \mathbf{21.7}\, n \ln n$ Bytecodes
- Classic Quicksort: $\sim \mathbf{18}\, n \ln n$ Bytecodes

Similar for Knuth's new mythical machine **MMIX**

- Yaroslavskiy: $\sim (\mathbf{13.1}\upsilon + \mathbf{2.8}\mu)\, n \ln n$
- Classic Quicksort: $\sim (\mathbf{11}\upsilon + \mathbf{2.}\overline{\mathbf{6}}\mu)\, n \ln n$

υ ("oops"): CPU cycles; μ ("mems"): memory accesses

$\rightsquigarrow$ *Classic Quicksort significantly* ***better*** *in all these measures . . .*

$\rightarrow$ *These models ignore memory hierarchies; see Alex López-Ortiz' talk!*

Results — Primitive Instructions

Results:

- Yaroslavskiy's algorithm: $\sim \mathbf{21.7}\, n \ln n$ Bytecodes
- Classic Quicksort: $\sim \mathbf{18}\, n \ln n$ Bytecodes

Similar for Knuth's new mythical machine **MMIX**

- Yaroslavskiy: $\sim (\mathbf{13.1}\upsilon + \mathbf{2.8}\mu)\, n \ln n$
- Classic Quicksort: $\sim (\mathbf{11}\upsilon + \mathbf{2.}\overline{\mathbf{6}}\mu)\, n \ln n$

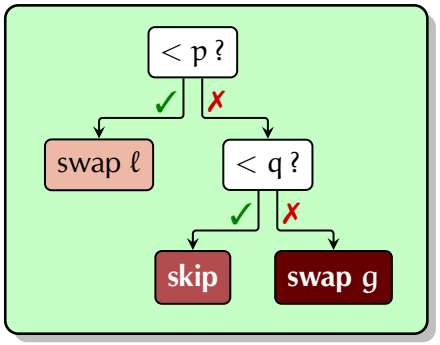
υ ("oops"): CPU cycles; μ ("mems"): memory accesses

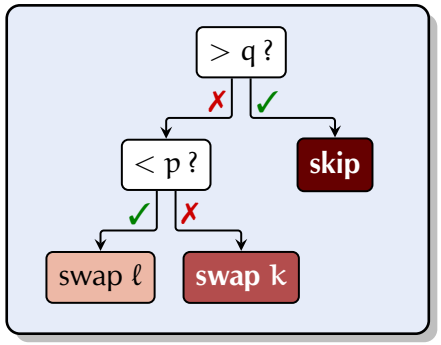
$\rightsquigarrow$ *Classic Quicksort significantly* ***better*** *in all these measures . . .*

$\rightarrow$ *These models ignore memory hierarchies; see Alex López-Ortiz' talk!*

Asymmetries in #Comparisons — Revisited

How many comparisons used?

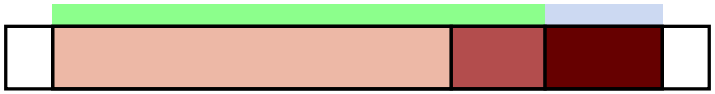




⇝ Cheap elements (**1 cmp.**):

- small in k's range ($s @ \mathcal{K}$)
- large in g's range ($l @ \mathcal{G}$)

Key Observation: sizes of classes and ranges are correlated:

- 

  many small elements ⇝ k's range big
- 

  many large elements ⇝ g's range big

Exploiting Asymmetries in #Comparisons

Motivation: medium elements always cause 2 cmps

$\rightsquigarrow$ Should avoid medium elements if possible!

Thought Experiment:

Assume always $\tau_1 \cdot n$ / $\tau_2 \cdot n$ / $\tau_3 \cdot n$ elements small / medium / large.

$\rightsquigarrow$ Comparisons per paritioning step:

$$2n - \tau_1 (\tau_1 + \tau_2) n - \tau_3 \tau_3 n$$

two per element

small in k's range

large in g's range

Minimal for $\tau = (1, 0, 0)$ or $\tau = (0, 0, 1)$.

Exploiting Asymmetries in #Comparisons

Motivation: medium elements always cause 2 cmps

$\rightsquigarrow$ Should avoid medium elements if possible!

Thought Experiment:

Assume always $\tau_1 \cdot n$ / $\tau_2 \cdot n$ / $\tau_3 \cdot n$ elements small / medium / large.

$\rightsquigarrow$ Comparisons per paritioning step:

$$2n - \tau_1 (\tau_1 + \tau_2) n - \tau_3 \tau_3 n$$

two per element

small in k's range

large in g's range

Minimal for $\tau = (1, 0, 0)$ or $\tau = (0, 0, 1)$.

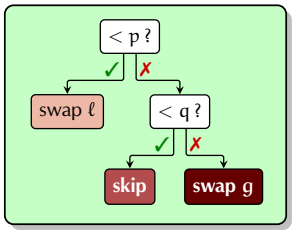
Exploiting Asymmetries in #Comparisons

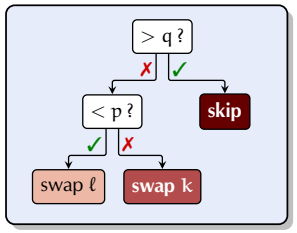
Motivation: medium elements always cause 2 cmps

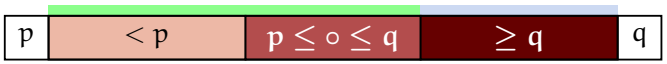
$\rightsquigarrow$ Should avoid medium elements if possible!

Thought Experiment:

Assume always $\tau_1 \cdot n$ / $\tau_2 \cdot n$ / $\tau_3 \cdot n$ elements small / medium / large.







$\rightsquigarrow$ Comparisons per paritioning step:

$$2n - \tau_1 (\tau_1 + \tau_2) n - \tau_3 \tau_3 n$$

two per element

small in k's range

large in g's range

Minimal for $\tau = (1, 0, 0)$ or $\tau = (0, 0, 1)$.

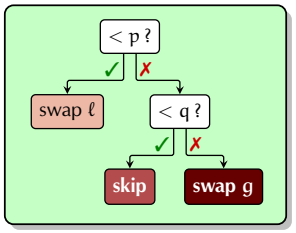
Exploiting Asymmetries in #Comparisons

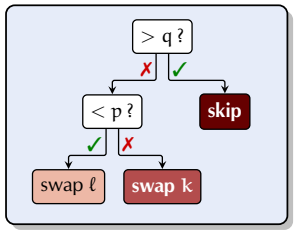
Motivation: medium elements always cause 2 cmps

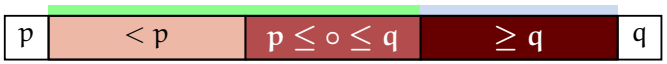
$\rightsquigarrow$ Should avoid medium elements if possible!

Thought Experiment:

Assume always $\tau_1 \cdot n$ / $\tau_2 \cdot n$ / $\tau_3 \cdot n$ elements small / medium / large.







$\rightsquigarrow$ Comparisons per paritioning step:

$$2n - \tau_1 (\tau_1 + \tau_2) n - \tau_3 \tau_3 n$$

two per element

small in k's range

large in g's range

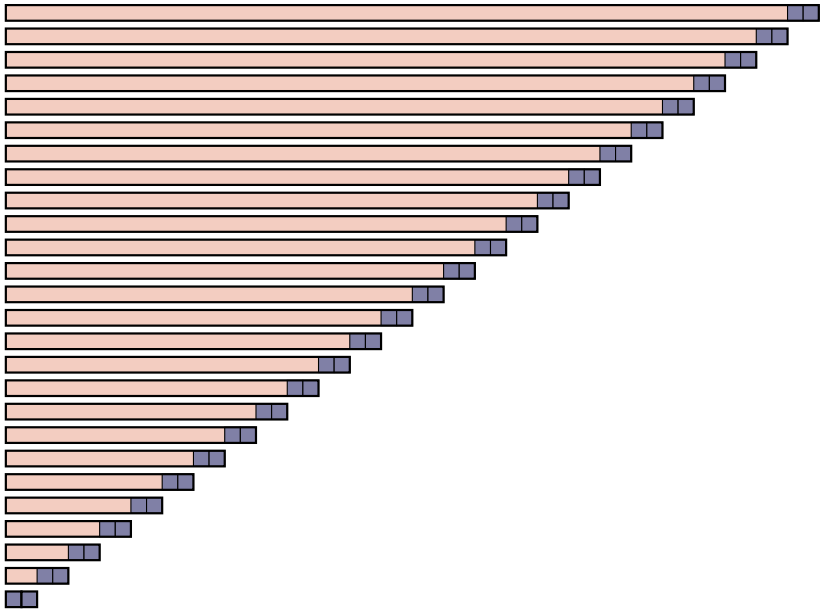
Minimal for $\tau = (1, 0, 0)$ or $\tau = (0, 0, 1)$.

Exploiting Asymmetries in #Comparisons

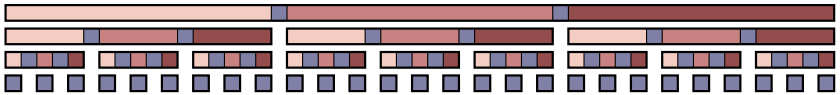
- $\tau = (1, 0, 0)$ optimal w. r. t. **one** partitioning step

But: worst case for partitioning:

$\tau = (1, 0, 0)$



Symmetric case $\tau = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$



Is $\tau = (\frac{1}{2}, 0, \frac{1}{2})$ better?

$\rightsquigarrow$ trade-off between partitioning costs and equality of split

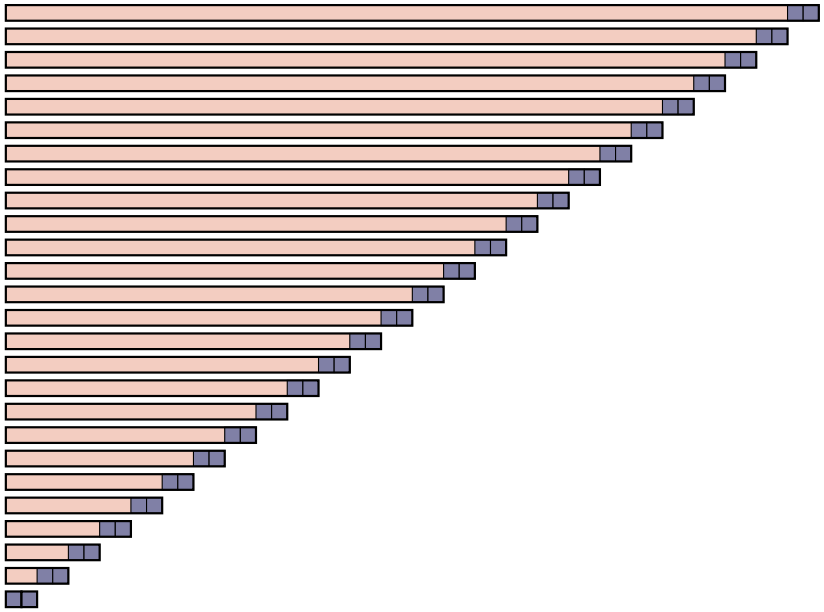
*What is the **optimal** τ?*

Exploiting Asymmetries in #Comparisons

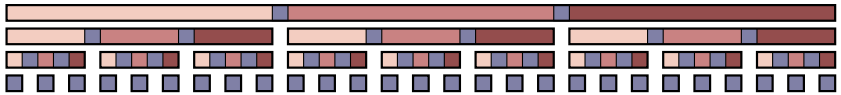
- $\tau = (1, 0, 0)$ optimal w. r. t. **one** partitioning step

But: worst case for partitioning:

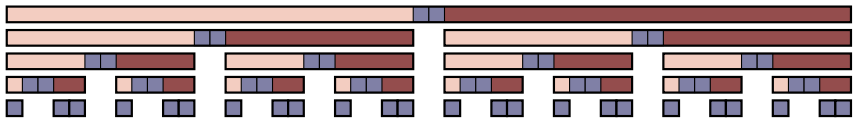
$\tau = (1, 0, 0)$



Symmetric case $\tau = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$



Is $\tau = (\frac{1}{2}, 0, \frac{1}{2})$ better?



$\rightsquigarrow$ trade-off between partitioning costs and equality of split

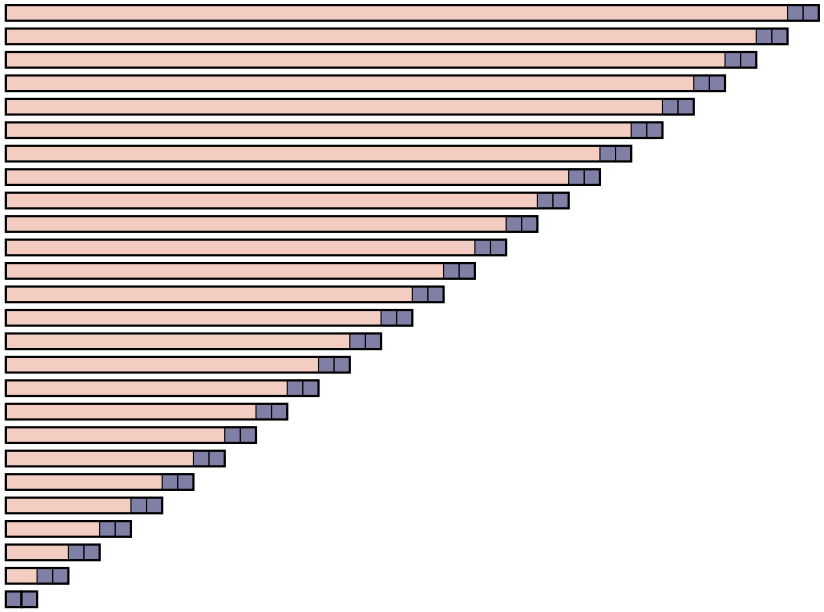
*What is the **optimal** τ?*

Exploiting Asymmetries in #Comparisons

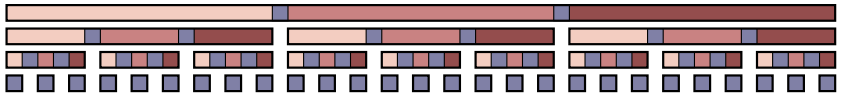
- $\tau = (1, 0, 0)$ optimal w. r. t. **one** partitioning step

But: worst case for partitioning:

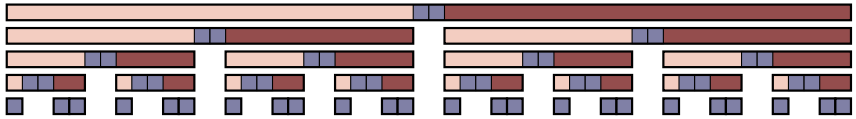
$\tau = (1, 0, 0)$



Symmetric case $\tau = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$



Is $\tau = (\frac{1}{2}, 0, \frac{1}{2})$ better?



$\rightsquigarrow$ trade-off between partitioning costs and equality of split

*What is the **optimal** τ?*

Results — Optimal τ

We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$

- quite different from $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$

- Similar analysis for #Bytecodes:
  $\tau^*_{BC} \approx (0.2068, 0.3485, 0.4447)$

- Optimal choice of pivots heavily depends on τ_1 and τ_2

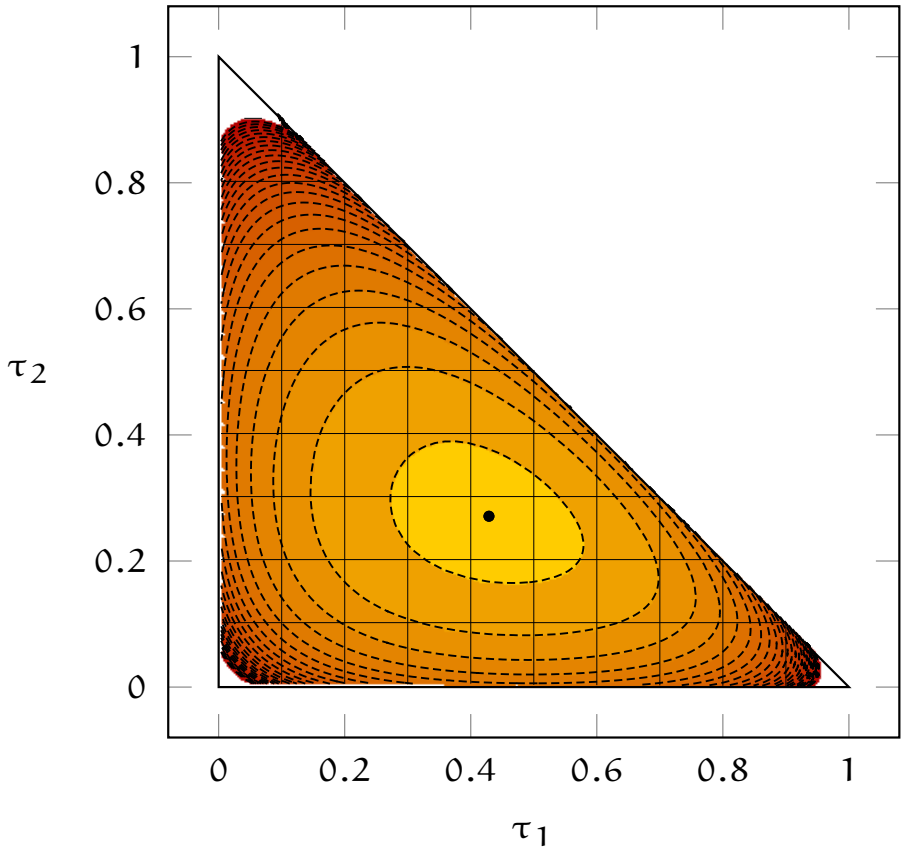
Results — Optimal τ

We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$



- quite different from $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- similar analysis for #Bytecodes:
  $\tau^*_{BC} \approx (0.2068, 0.3485, 0.4447)$



$\rightsquigarrow$ Optimal **skew** of pivots heavily depends on **cost measure**!

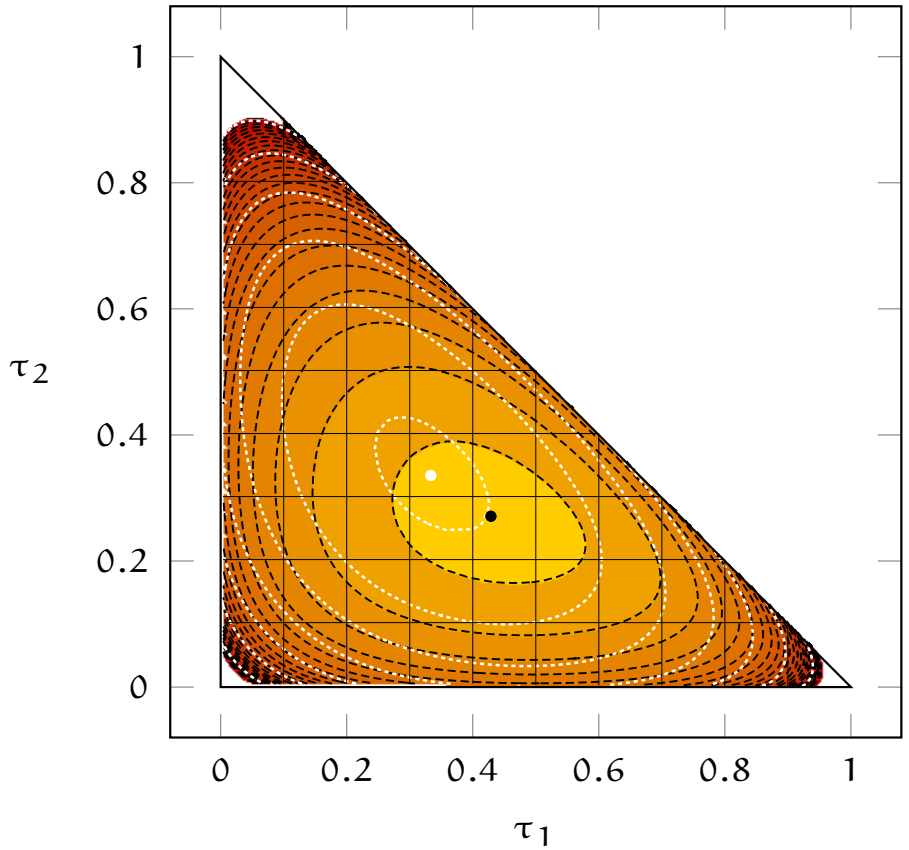
Results — Optimal τ

We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\,\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$



- quite different from $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- similar analysis for #Bytecodes:
  $\tau^*_{BC} \approx (0.2068, 0.3485, 0.4447)$



$\rightsquigarrow$ Optimal **skew** of pivots heavily depends on **cost measure**!

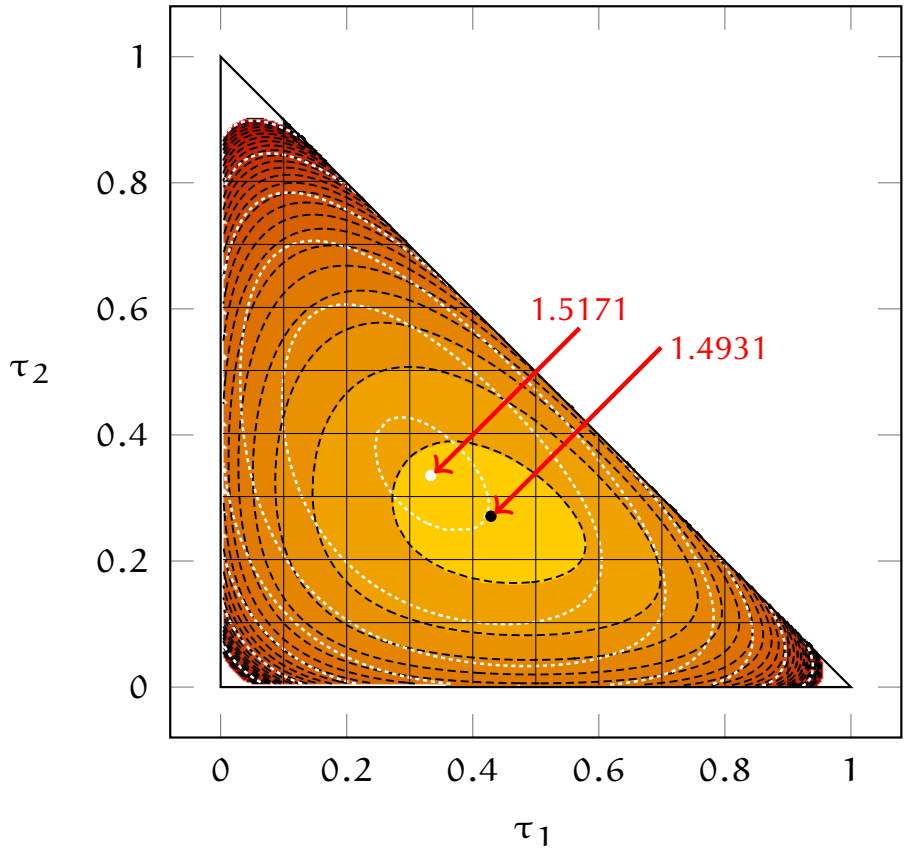
Results — Optimal τ

We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$



- quite different from $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- similar analysis for #Bytecodes:
  $\tau^*_{BC} \approx (0.2068, 0.3485, 0.4447)$



$\rightsquigarrow$ Optimal **skew** of pivots heavily depends on **cost measure**!

Results — Optimal τ

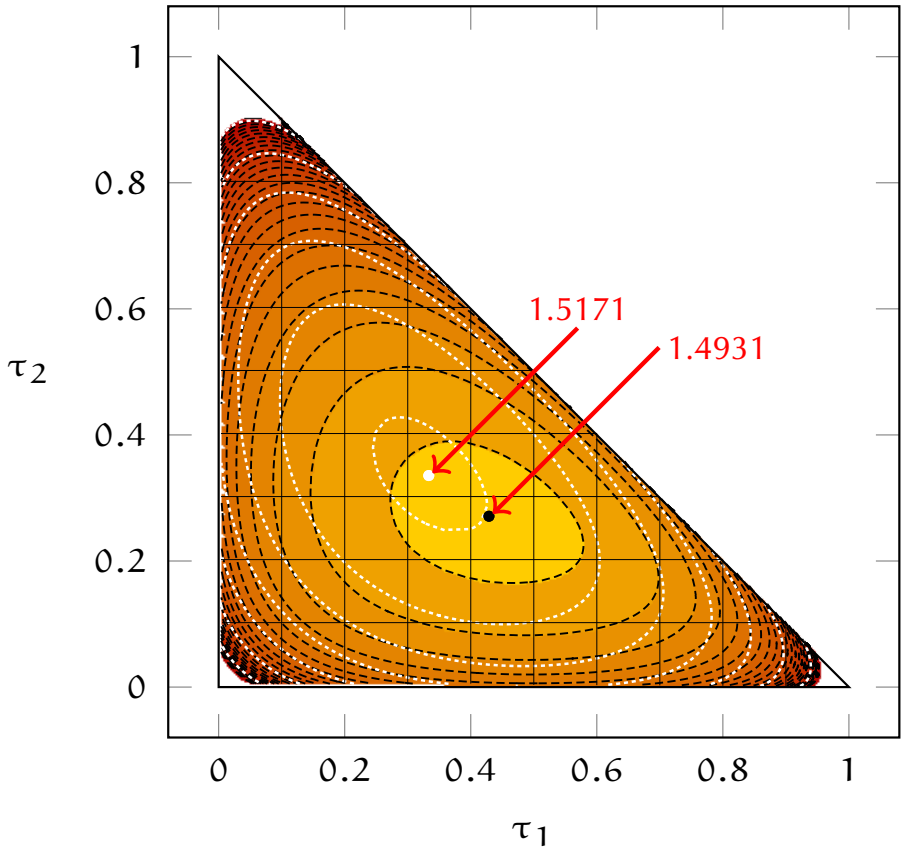
We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\,\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$



- quite different from $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- similar analysis for #Bytecodes:
  $\tau^*_{\mathrm{BC}} \approx (0.2068, 0.3485, 0.4447)$





$\rightsquigarrow$ Optimal **skew** of pivots heavily depends on **cost measure**!

Results — Optimal τ

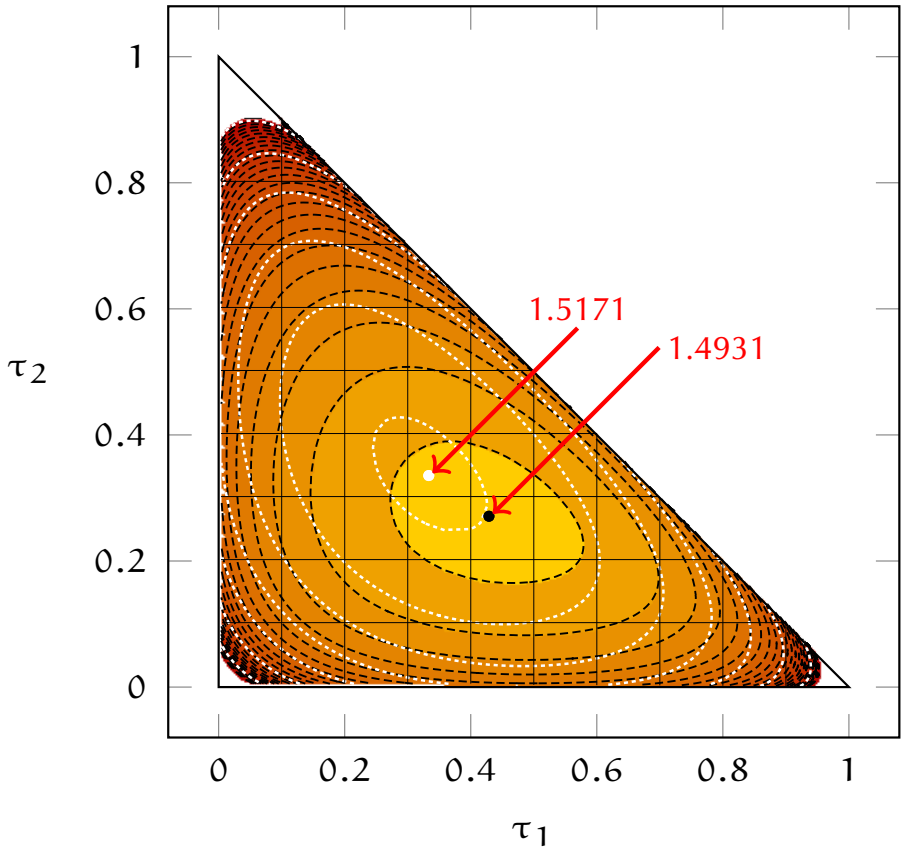
We could show: Overall $\sim \dfrac{2 - \tau_1(\tau_1 + \tau_2) - \tau_3\,\tau_3}{-\sum_{i=1}^{3} \tau_i \ln(\tau_i)}\, n \ln n$ cmps

- optimal choice
  $\tau^* \approx (0.4288, 0.2688, 0.3024)$



- quite different from $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$
- similar analysis for #Bytecodes:
  $\tau^*_{BC} \approx (0.2068, 0.3485, 0.4447)$





$\rightsquigarrow$ Optimal **skew** of pivots heavily depends on **cost measure**!

Conclusion

We've seen:

- **Breaking symmetry** in order of comparisons is **beneficial** in multi-pivot Quicksort
- skewed pivots can amplify savings
- **trade-off** between partitioning costs and balance in subproblem sizes

Open Questions:

- How about other input distributions?
  (up to now only random permutations)
- Is Yaroslavskiy's algorithm *special*?
  Are there better asymmetric Quicksorts (for practice)?

  → *Martin Aumüller's talk*

Conclusion

We've seen:

- **Breaking symmetry** in order of comparisons is **beneficial** in multi-pivot Quicksort
- skewed pivots can amplify savings
- **trade-off** between partitioning costs and balance in subproblem sizes

Open Questions:

- How about other input distributions?
  (up to now only random permutations)
- Is Yaroslavskiy's algorithm *special*?
  Are there better asymmetric Quicksorts (for practice)?

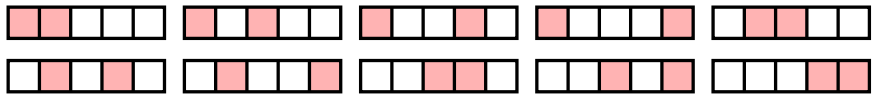
  → *Martin Aumüller's talk*

Choice of τ in Practice

Lesson for Practice?

- exact quantiles too expensive $\rightsquigarrow$ use **sampling**!
- sample size k controls deviation

$\rightsquigarrow$ limited to **discrete** quantiles



- **Java 7 library** uses **tertiles of five** elements

$\mathbb{E}[\tau^{2,4}] =$

Recall: Optimum (for #Bytecodes)

- $\tau^*_{\text{BC}} \approx (0.2068, 0.3485, 0.4447)$

$\rightsquigarrow$ probably better choice:

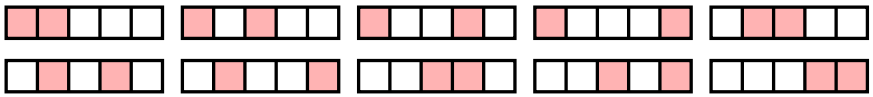
$\mathbb{E}[\tau^{1,3}] =$

Choice of τ in Practice

Lesson for Practice?

- exact quantiles too expensive $\rightsquigarrow$ use **sampling**!
- sample size k controls deviation

$\rightsquigarrow$ limited to **discrete** quantiles



- **Java 7 library** uses **tertiles of five** elements

$\mathbb{E}[\boldsymbol{\tau}^{2,4}] =$



Recall: Optimum (for #Bytecodes)

- $\boldsymbol{\tau}^*_{\text{BC}} \approx (0.2068, 0.3485, 0.4447)$

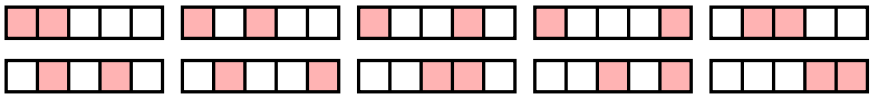
$\rightsquigarrow$ probably better choice:

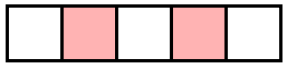
$\mathbb{E}[\boldsymbol{\tau}^{1,3}] =$

Choice of τ in Practice

Lesson for Practice?

- exact quantiles too expensive $\rightsquigarrow$ use **sampling**!
- sample size k controls deviation

$\rightsquigarrow$ limited to **discrete** quantiles 

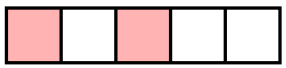
- **Java 7 library** uses **tertiles of five** elements 

  $\mathbb{E}[\tau^{2,4}] =$ 

Recall: Optimum (for #Bytecodes)

- $\tau^*_{\text{BC}} \approx (0.2068, 0.3485, 0.4447)$

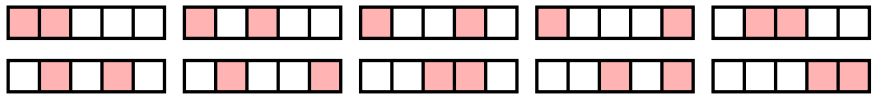
$\rightsquigarrow$ probably better choice: 

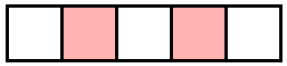
$\mathbb{E}[\tau^{1,3}] =$ 

Choice of τ in Practice

Lesson for Practice?

- exact quantiles too expensive $\rightsquigarrow$ use **sampling**!
- sample size k controls deviation

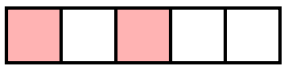
$\rightsquigarrow$ limited to **discrete** quantiles 

- **Java 7 library** uses **tertiles of five** elements 

$\mathbb{E}[\tau^{2,4}] =$  $\rightsquigarrow$ **19.298** $n \ln n$ Bytecodes

Recall: Optimum (for #Bytecodes)

- $\tau^*_{\mathrm{BC}} \approx (0.2068, 0.3485, 0.4447)$ 

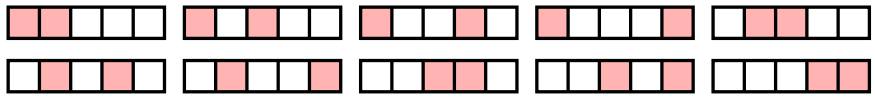
$\rightsquigarrow$ probably better choice: 

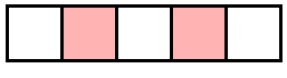
$\mathbb{E}[\tau^{1,3}] =$  $\rightsquigarrow$ **18.791** $n \ln n$ Bytecodes

Choice of τ in Practice

Lesson for Practice?

- exact quantiles too expensive $\rightsquigarrow$ use **sampling**!
- sample size k controls deviation

$\rightsquigarrow$ limited to **discrete** quantiles 

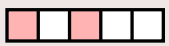
- **Java 7 library** uses **tertiles of five** elements 

$\mathbb{E}[\tau^{2,4}] =$  $\rightsquigarrow$ **19.298** $n \ln n$ Bytecodes

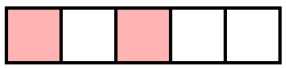
Recall: Optimum (for #Bytecodes)

- $\tau^*_{\text{BC}} \approx (0.2068, 0.3485, 0.4447)$ 

Running time:

C++:  0.5 % faster

Java: . . . depends on JIT

$\rightsquigarrow$ probably better choice: 

$\mathbb{E}[\tau^{1,3}] =$  $\rightsquigarrow$ **18.791** $n \ln n$ Bytecodes

Beating the "Lower Bound"

- $\sim \frac{20}{12} n$ comparisons only needed,
  if there is **one** comparison **location**,
  then checks for x and y **independent**

- **But:** Can have **several** comparison locations!

  Here: Assume **two** locations c_1 and c_2 s. t.

  - c_1 first compares with p.
  - c_2 first compares with q.
  - c_1 executed often, *iff* p is **large**.
  - c_2 executed often, *iff* q is **small**.

- $\rightsquigarrow$ c_1 executed often
  iff many small elements
  iff good chance that c_1 needs only one comparison
  (c_2 similar)

- $\rightsquigarrow$ **less** comparisons than $\frac{5}{3}$ per elements on average

Yaroslavskiy's Quicksort

```
DualPivotQuicksortYaroslavskiy(A, left, right)
 1  if right − left ⩾ 1
 2      p := A[left];    q := A[right]
 3      if p > q then Swap p and q end if
 4      ℓ := left + 1;    g := right − 1;    k := ℓ
 5      while k ⩽ g
 6          if A[k] < p
 7              Swap A[k] and A[ℓ] ;   ℓ := ℓ + 1
 8          else if A[k] ⩾ q
 9              while A[g] > q and k < g do g := g − 1 end while
10              Swap A[k] and A[g] ;    g := g − 1
11              if A[k] < p
12                  Swap A[k] and A[ℓ] ;    ℓ := ℓ + 1
13              end if
14          end if
15          k := k + 1
16      end while
17      ℓ := ℓ − 1;    g := g + 1
18      Swap A[left] and A[ℓ] ;    Swap A[right] and A[g]
19      DualPivotQuicksortYaroslavskiy(A,  left  , ℓ − 1 )
20      DualPivotQuicksortYaroslavskiy(A, ℓ + 1 , g − 1 )
21      DualPivotQuicksortYaroslavskiy(A, g + 1,  right  )
22  end if
```

Yaroslavskiy's Quicksort

```
DualPivotQuicksortYaroslavskiy(A, left, right)
 1  if right − left ⩾ 1
 2      p := A[left];    q := A[right]
 3      if p > q then Swap p and q end if
 4      ℓ := left + 1;    g := right − 1;    k := ℓ
 5      while k ⩽ g
 6  C_k     if A[k] < p
 7              Swap A[k] and A[ℓ];    ℓ := ℓ + 1
 8  C'_k    else if A[k] ⩾ q
 9  C_g         while A[g] > q and k < g do g := g − 1 end while
10              Swap A[k] and A[g];    g := g − 1
11  C'_g        if A[k] < p
12                  Swap A[k] and A[ℓ];    ℓ := ℓ + 1
13              end if
14          end if
15          k := k + 1
16      end while
17      ℓ := ℓ − 1;    g := g + 1
18      Swap A[left] and A[ℓ];    Swap A[right] and A[g]
19      DualPivotQuicksortYaroslavskiy(A, left , ℓ − 1)
20      DualPivotQuicksortYaroslavskiy(A, ℓ + 1, g − 1)
21      DualPivotQuicksortYaroslavskiy(A, g + 1, right )
22  end if
```

- **2** comparison locations
- C_k handles pointer k
  C_g handles pointer g
- C_k first checks $< p$
  C'_k **if needed** $\geqslant q$
- C_g first checks $> q$
  C'_g **if needed** $< p$

Dual Pivot Quicksort Recurrence

- C_n expected #comparisons to sort random permutation of $\{1, \dots, n\}$
- C_n satisfies **recurrence relation**

$$C_n = T_n + \frac{2}{n(n-1)} \sum_{1 \leqslant p < q \leqslant n} \left(C_{p-1} + C_{q-p-1} + C_{n-q} \right),$$

  with T_n expected #comparisons in **first** partitioning step
- recurrence solvable by standard methods

linear $T_n \sim a \cdot n$ yields $C_n \sim \frac{6}{5} a \cdot n \ln n$.

$\rightsquigarrow$ *It remains to compute* T_n.

Analysis of Yaroslavskiy's Algorithm (1)

- **first** comparison for **all** elements (at C_k **or** C_g)
  $\rightsquigarrow \sim n$ comparisons

- **second** comparison for **some** elements at C'_k resp. C'_g
  ... but how often are C'_k resp. C'_g reached?

- C'_k: all **non-** small elements **reached** by pointer k.
  C'_g: all **non-** large elements **reached** by pointer g.

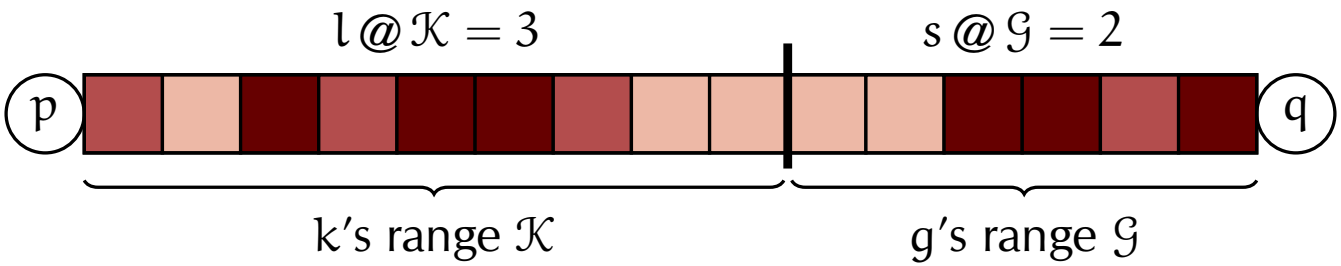
- second comparison for medium elements **not avoidable**
  $\rightsquigarrow \sim \frac{1}{3}n$ comparisons in expectation

- $\rightsquigarrow$ it remains to count:
  large elements reached by k and
  small elements reached by g.

Analysis of Yaroslavskiy's Algorithm (2)

- **Second** comparisons for small and large elements?
  Depends on **location**!

- $C'_k \rightsquigarrow$ number of **large** elements in k's range: $l @ \mathcal{K}$
  $C'_g \rightsquigarrow$ number of **small** elements in g's range: $s @ \mathcal{G}$

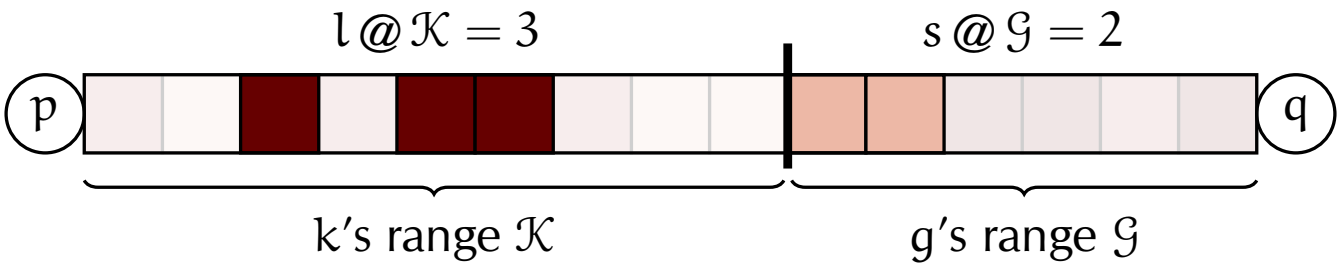


- $l @ \mathcal{K}$ and $s @ \mathcal{G}$ are **random** even for **fixed** p and q

Analysis of Yaroslavskiy's Algorithm (2)

- **Second** comparisons for small and large elements?
  Depends on **location**!

- $C'_k \rightsquigarrow$ number of **large** elements in k's range: $l @ \mathcal{K}$
  $C'_g \rightsquigarrow$ number of **small** elements in g's range: $s @ \mathcal{G}$

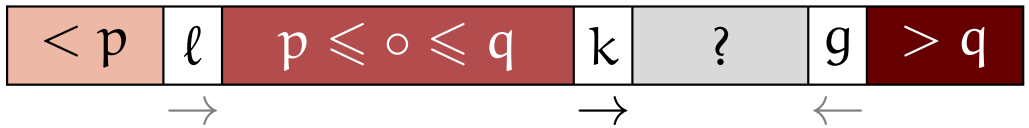


- $l @ \mathcal{K}$ and $s @ \mathcal{G}$ are **random** even for **fixed** p and q

Distribution of $\ell @ \mathcal{K}$ and $s @ \mathcal{G}$ (1)

Assume p and q are **fixed.**

1. **Where do k and g cross?**
   - Recall invariant:



   - $\rightsquigarrow$ k and g cross at (rank of) q

     $|\mathcal{K}| \sim q, \qquad |\mathcal{G}| \sim n - q$

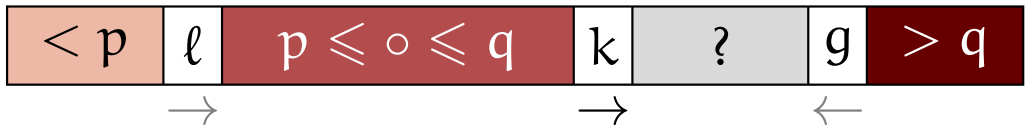
2. **How many small and large elements?**
   - $\#\text{small} = |\{1, \dots, p-1\}| = p - 1$
   - $\#\text{large} = |\{q+1, \dots, n\}| = n - q$

$\rightsquigarrow$ $|\mathcal{K}|$, $|\mathcal{G}|$, #small and #large are **constant** (for given p and q).

Distribution of ℓ@$\mathcal{K}$ and s@$\mathcal{G}$ (1)

Assume p and q are **fixed.**

1. **Where do k and g cross?**
   - Recall invariant:



   - $\rightsquigarrow$ k and g cross at (rank of) q

     $|\mathcal{K}| \sim q, \qquad |\mathcal{G}| \sim n - q$

2. **How many small and large elements?**
   - $\#\text{small} = \big|\{1, \dots, p-1\}\big| = p - 1$
   - $\#\text{large} = \big|\{q+1, \dots, n\}\big| = n - q$

$\rightsquigarrow$ $|\mathcal{K}|$, $|\mathcal{G}|$, #small and #large are **constant** (for given p and q).

Distribution of $l@\mathcal{K}$ and $s@\mathcal{G}$ (2)

3. ***Conditional* Distribution of $l@\mathcal{K}$**
   - We draw **positions** of **large** elements at random.
   - $n-2$ positions
   - draw $\#$**large** positions **without replacement**
   - $|\mathcal{K}|$ positions contribute to $l@\mathcal{K}$

$$\rightsquigarrow l@\mathcal{K} \overset{\mathcal{D}}{=} \text{Hypergeometric}(\#\text{large}, |\mathcal{K}|, n-2)$$

$$= \textit{Number of } \textbf{red} \textit{ balls obtained when drawing } \#\textbf{large} \textit{ balls } \textbf{without replacement} \textit{ form } \textbf{urn} \textit{ with } n-2 \textit{ balls, exactly } |\mathcal{K}| \textit{ of which are } \textbf{red}.$$

- $\mathbb{E}[l@\mathcal{K} \mid p, q] = \dfrac{\#\text{large} \cdot |\mathcal{K}|}{n-2} \underset{1,2}{\sim} \dfrac{(n-q)\cdot q}{n-2}$
- **law of total expectation:**

$$\mathbb{E}[l@\mathcal{K}] = \sum_{1 \leqslant p < q \leqslant n} 1/\binom{n}{2} \cdot \frac{(n-q)\cdot q}{n-2} \sim \frac{1}{6}n$$

Distribution of $l@\mathcal{K}$ and $s@\mathcal{G}$ (2)

3. ***Conditional* Distribution of $l@\mathcal{K}$**
   - We draw **positions** of **large** elements at random.
   - $n-2$ positions
   - draw $\#$**large** positions **without replacement**
   - $|\mathcal{K}|$ positions contribute to $l@\mathcal{K}$

$$\rightsquigarrow l@\mathcal{K} \overset{\mathcal{D}}{=} \text{Hypergeometric}(\#\text{large}, |\mathcal{K}|, n-2)$$

$$= \textit{Number of } \textbf{\textit{red}} \textit{ balls obtained when drawing } \#\textbf{\textit{large}} \textit{ balls } \textbf{\textit{without replacement}} \textit{ form } \textbf{\textit{urn}} \textit{ with } n-2 \textit{ balls, exactly } |\mathcal{K}| \textit{ of which are } \textbf{\textit{red}}.$$

- $\mathbb{E}[l@\mathcal{K} \mid p, q] = \dfrac{\#\text{large} \cdot |\mathcal{K}|}{n-2} \underset{1,2}{\sim} \dfrac{(n-q)\cdot q}{n-2}$

- **law of total expectation:**

$$\mathbb{E}[l@\mathcal{K}] = \sum_{1 \leqslant p < q \leqslant n} 1/\binom{n}{2} \cdot \frac{(n-q)\cdot q}{n-2} \sim \frac{1}{6}n$$