Dual Pivot Quicksort: Verification and Proof using KeY

Jonas Schiffl

Karlsruher Institut für Technologie

July 27th, 2016

Introduction

Introduction

Why verify Dual Pivot Quicksort?

Introduction

Why verify Dual Pivot Quicksort?

- Inspired by discovery of Timsort Bug

Introduction

Why verify Dual Pivot Quicksort?

- Inspired by discovery of Timsort Bug
- Widely used standard library algorithm

Introduction

Why verify Dual Pivot Quicksort?

- Inspired by discovery of Timsort Bug
- Widely used standard library algorithm
- Complex enough

Introduction

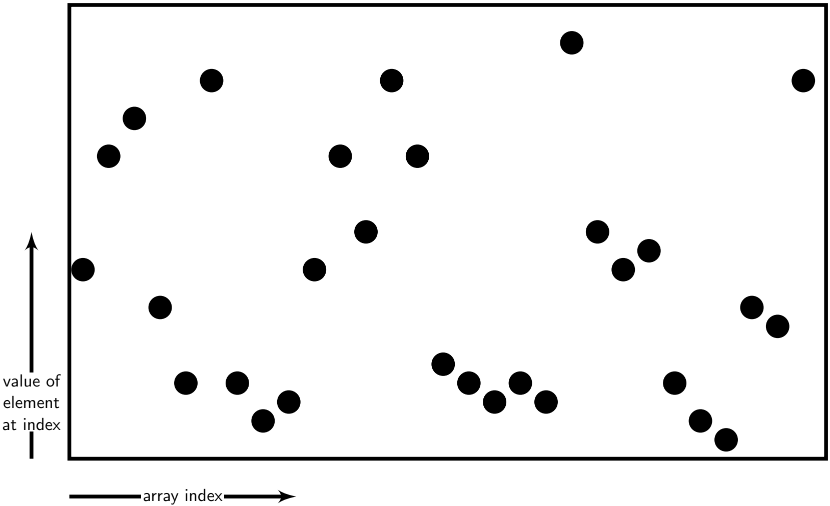
Why verify Dual Pivot Quicksort?

- Inspired by discovery of Timsort Bug
- Widely used standard library algorithm
- Complex enough
- Simple enough

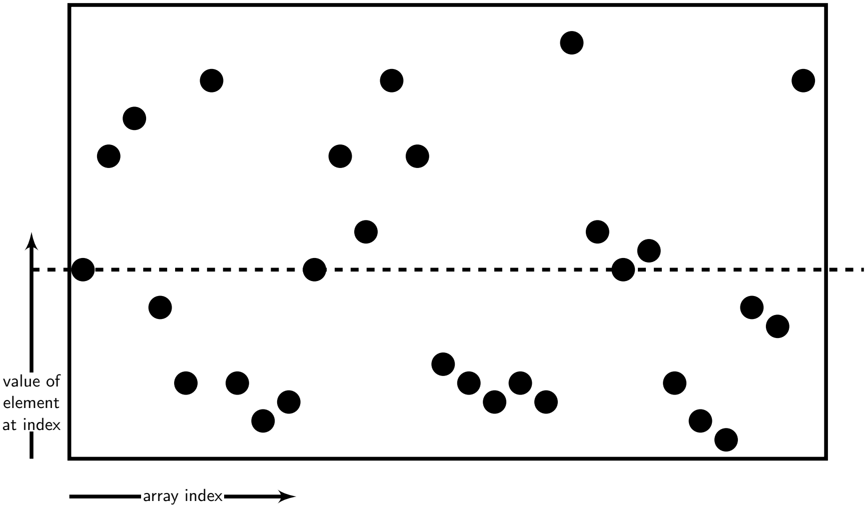
Section 1

Algorithm Description

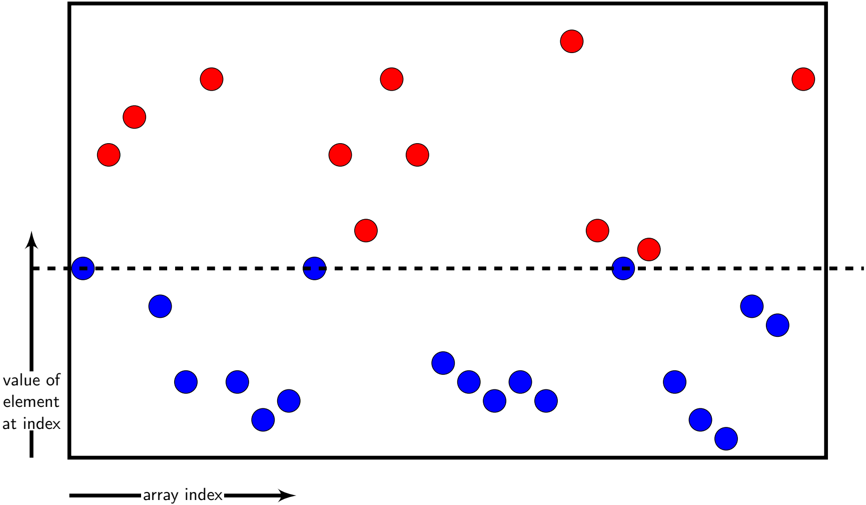
Quicksort



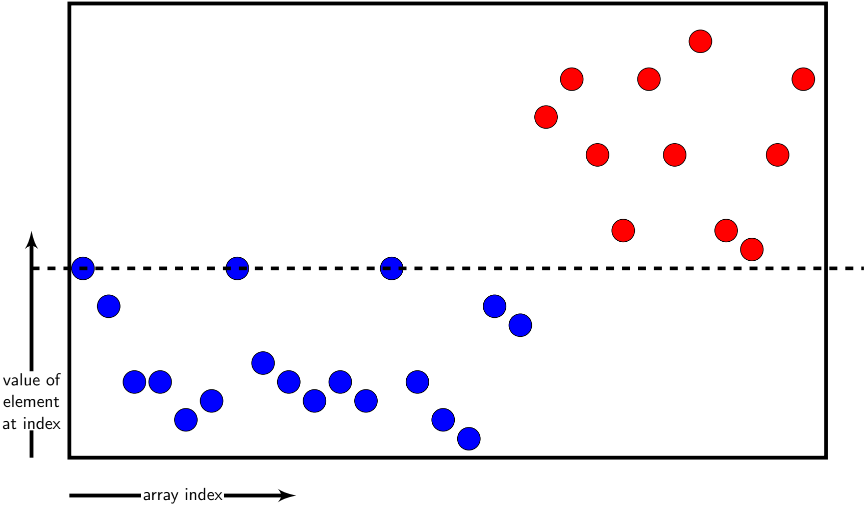
Quicksort



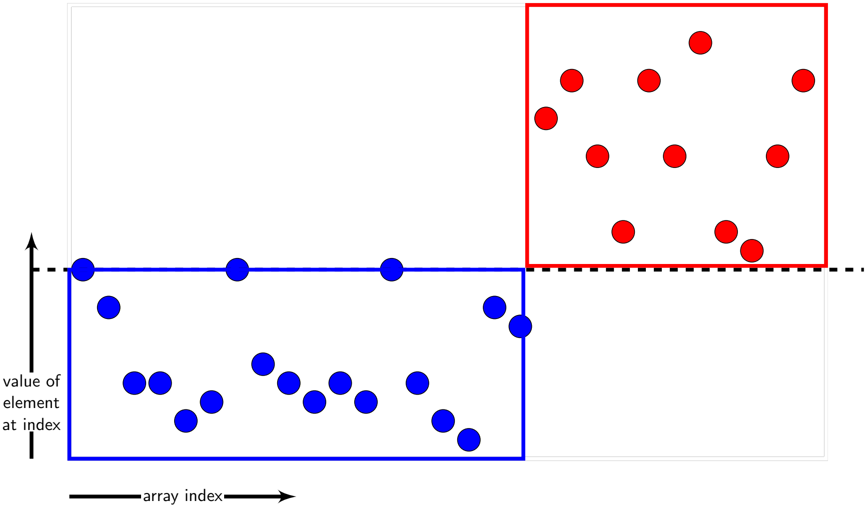
Quicksort



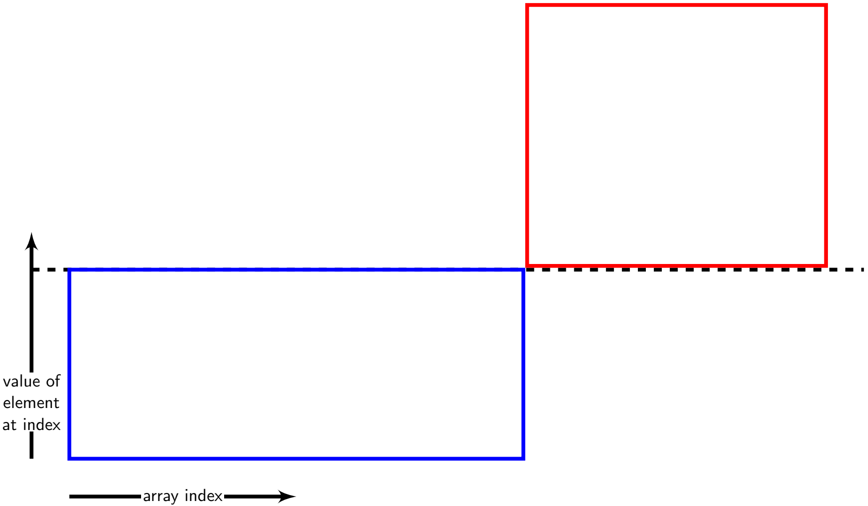
Quicksort



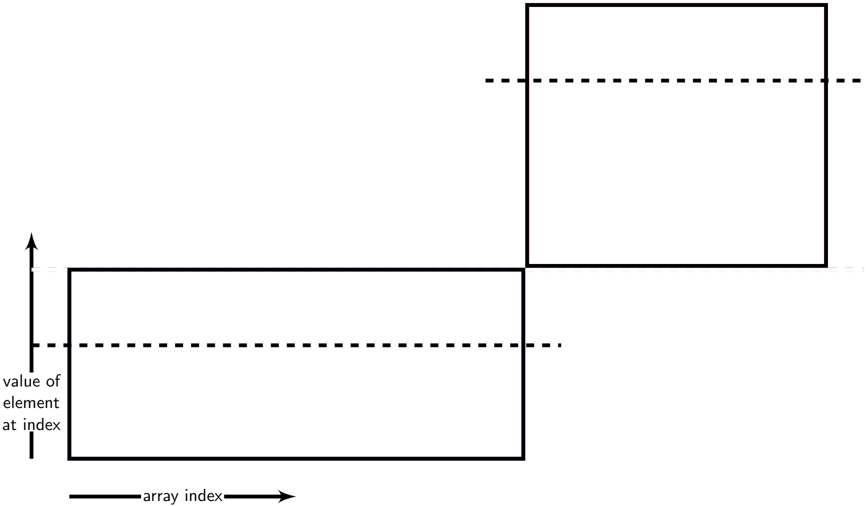
Quicksort



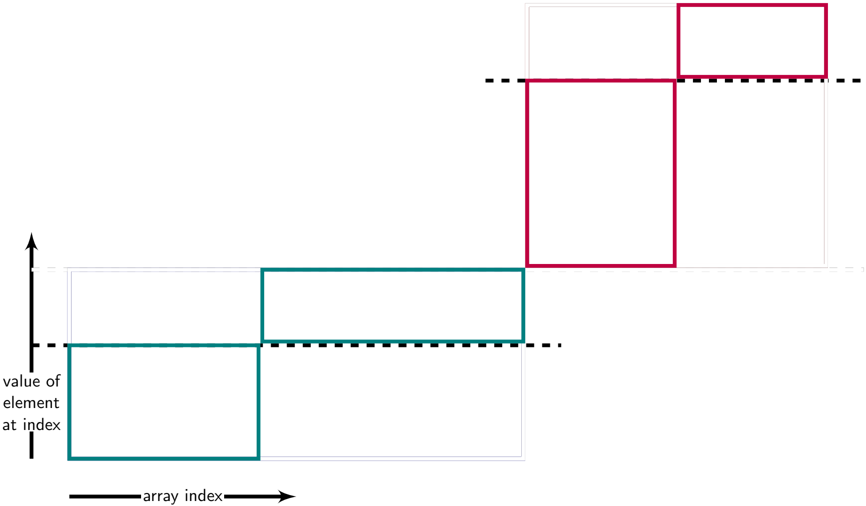
Quicksort



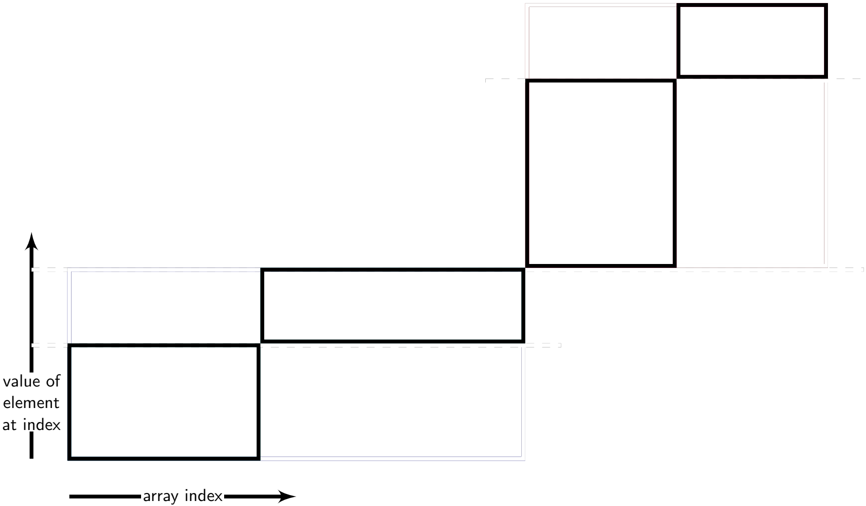
Quicksort



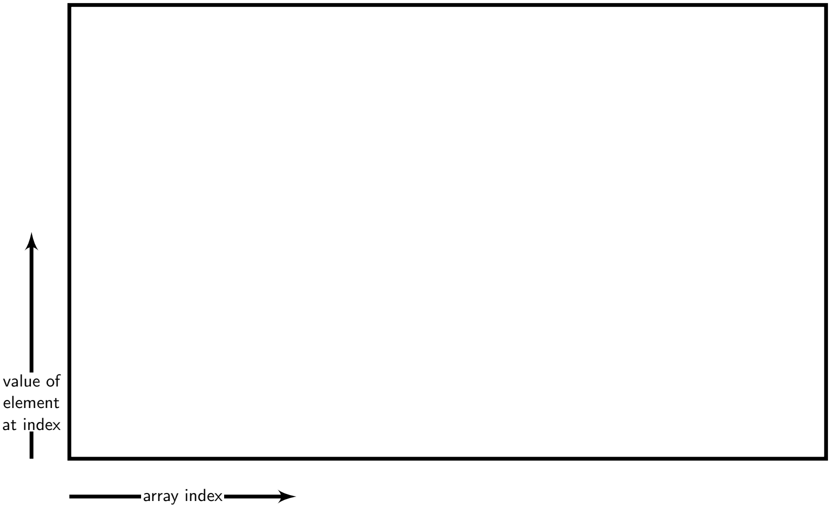
Quicksort



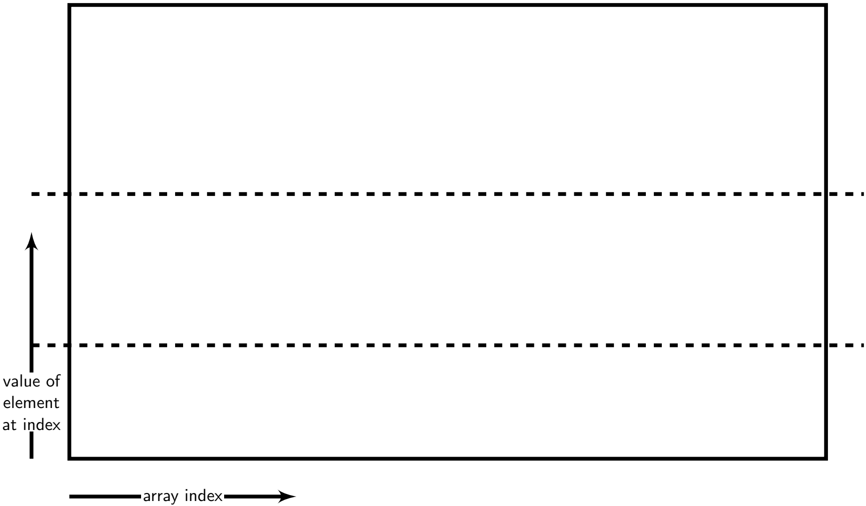
Quicksort



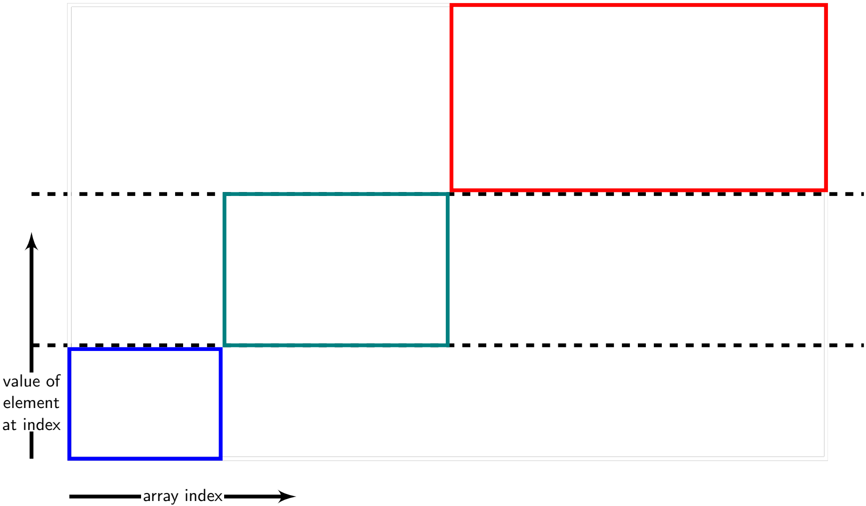
Dual Pivot Quicksort



Dual Pivot Quicksort



Dual Pivot Quicksort



Dual Pivot Quicksort

Why use Dual Pivot Quicksort?

Dual Pivot Quicksort

Why use Dual Pivot Quicksort?

- Theory: Average number of swaps reduced by 20% (Yaroslavskiy 2009)

Dual Pivot Quicksort

Why use Dual Pivot Quicksort?

- Theory: Average number of swaps reduced by 20% (Yaroslavskiy 2009)
- Practice: Multi-pivot Quicksorts are more cache-efficient (Kushagra 2014)

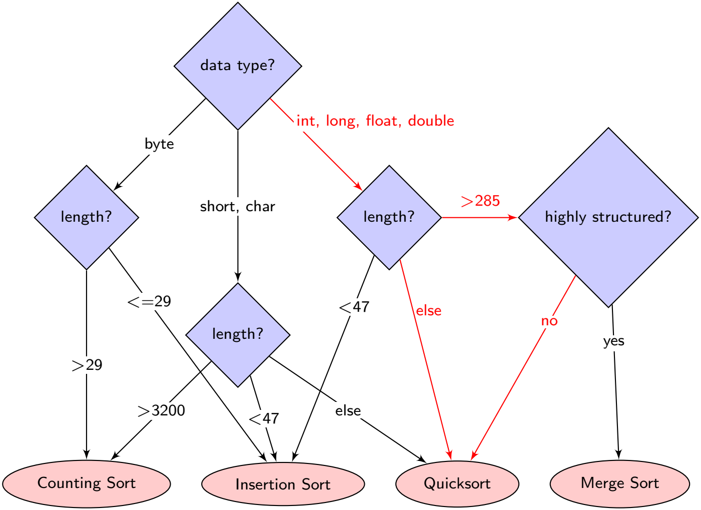
Dual Pivot Quicksort

Why use Dual Pivot Quicksort?

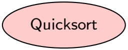
- Theory: Average number of swaps reduced by 20% (Yaroslavskiy 2009)
- Practice: Multi-pivot Quicksorts are more cache-efficient (Kushagra 2014)
- Benchmarking shows it is faster

Java Implementation – Choosing a Sorting Algorithm

Java Implementation – Choosing a Sorting Algorithm

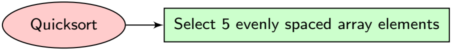


Java Implementation – Quicksort

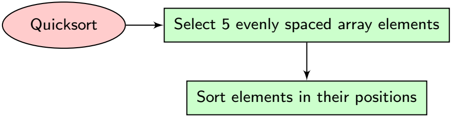


Quicksort

Java Implementation – Quicksort



Java Implementation – Quicksort

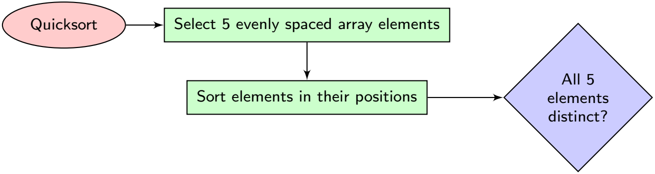


Quicksort

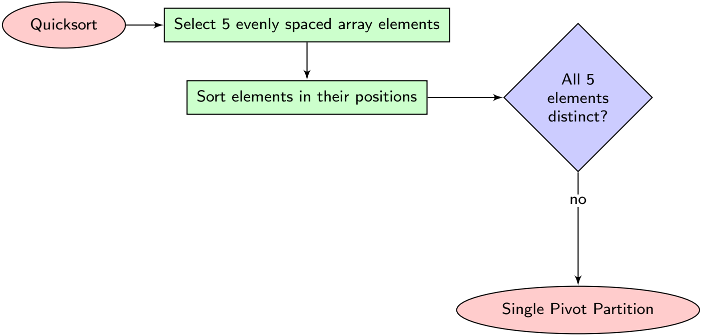
Select 5 evenly spaced array elements

Sort elements in their positions

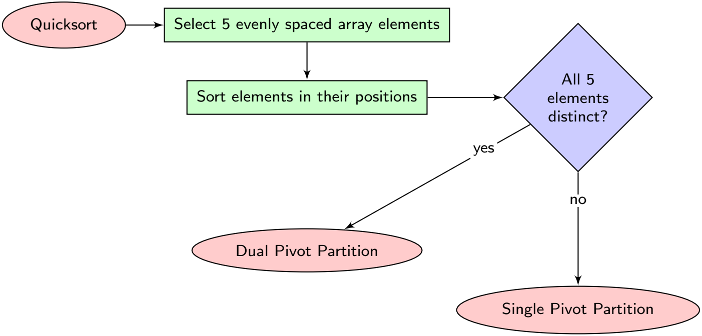
Java Implementation – Quicksort


Quicksort
Select 5 evenly spaced array elements
Sort elements in their positions
All 5 elements distinct?

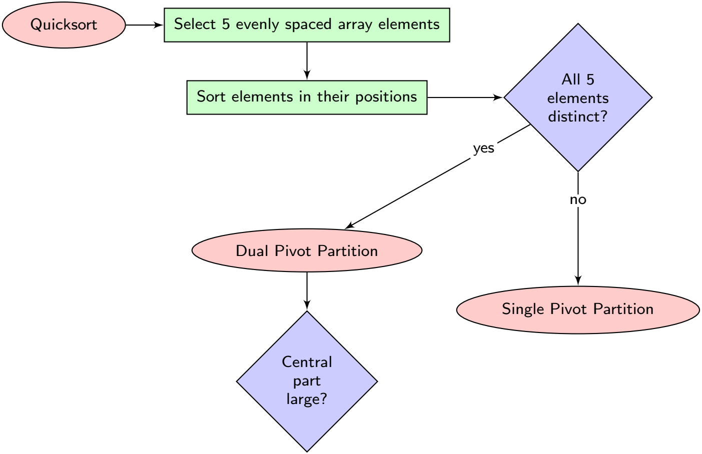
Java Implementation – Quicksort


Quicksort
Select 5 evenly spaced array elements
Sort elements in their positions
All 5 elements distinct?
no
Single Pivot Partition

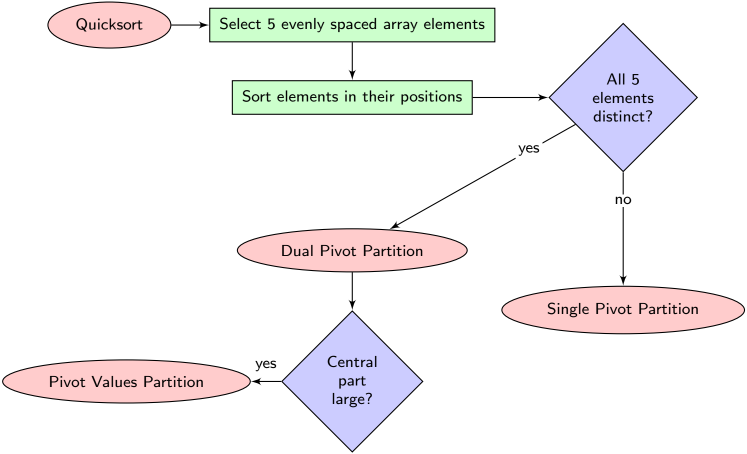
Java Implementation – Quicksort



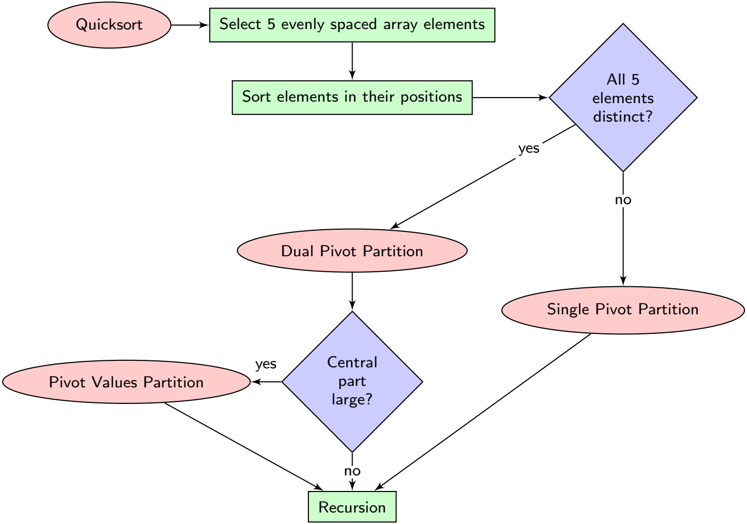
Java Implementation – Quicksort



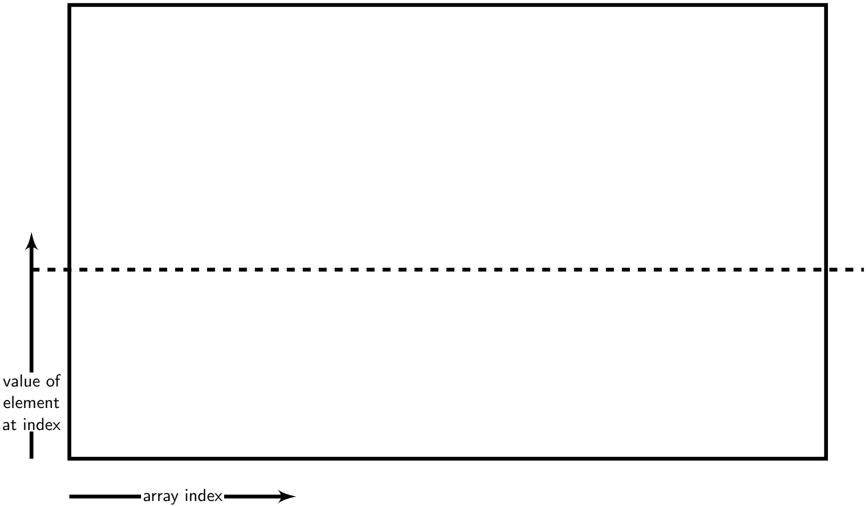
Java Implementation – Quicksort



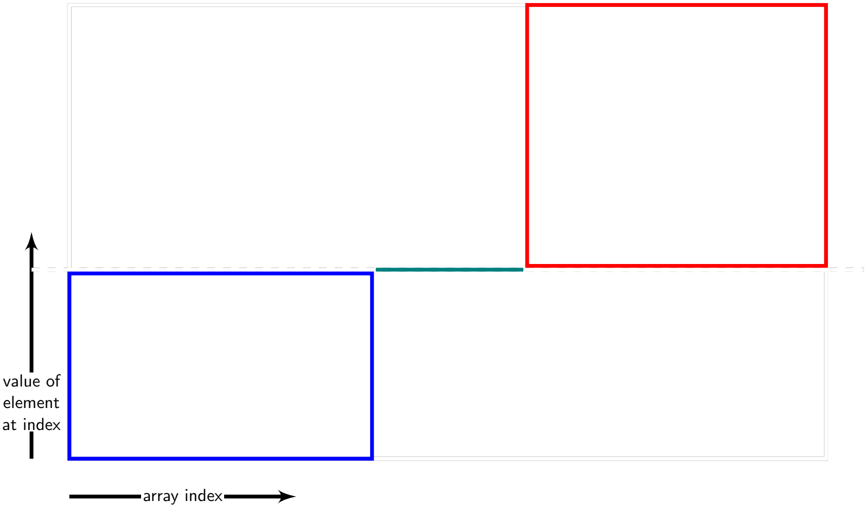
Java Implementation – Quicksort



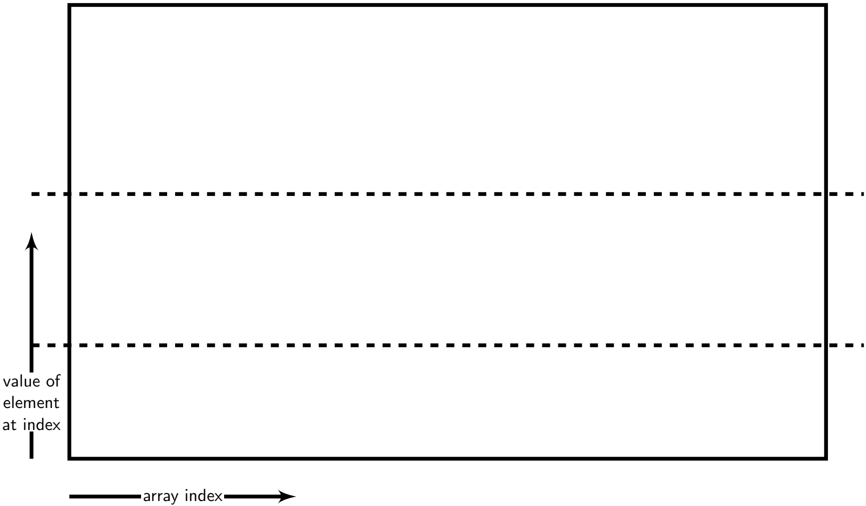
Java Implementation – Single Pivot Partition



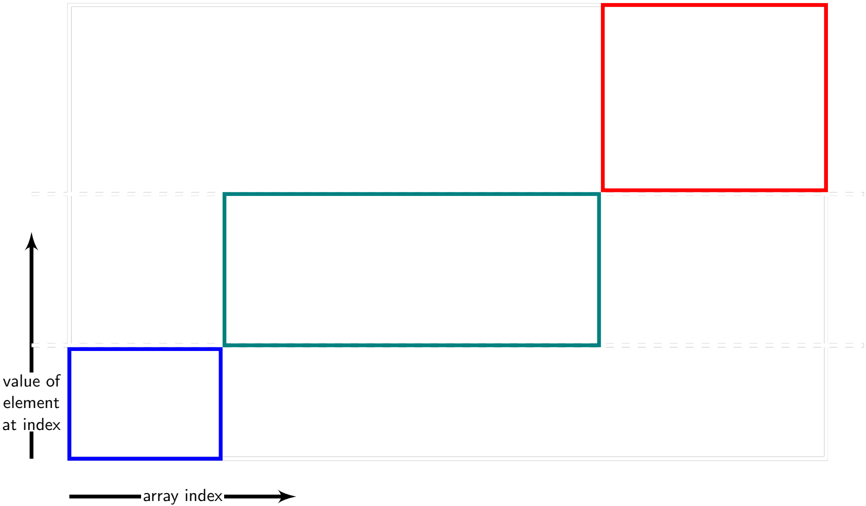
Java Implementation – Single Pivot Partition



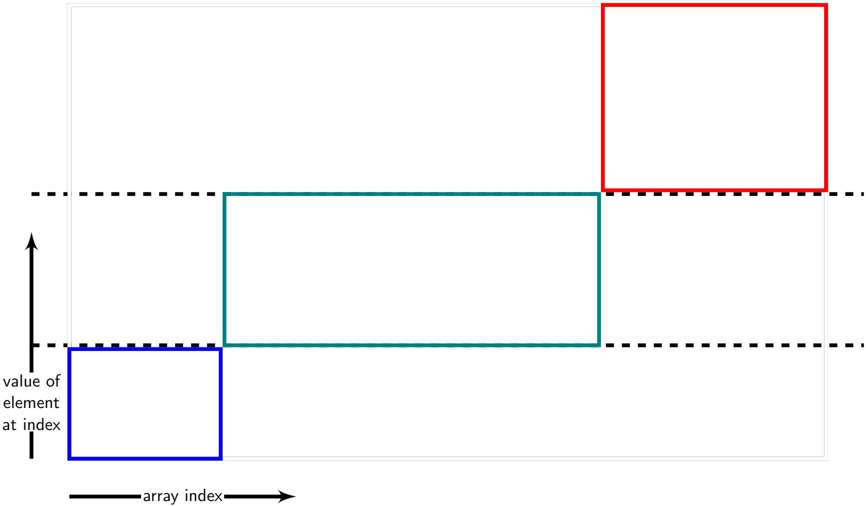
Java Implementation – Dual Pivot Partition



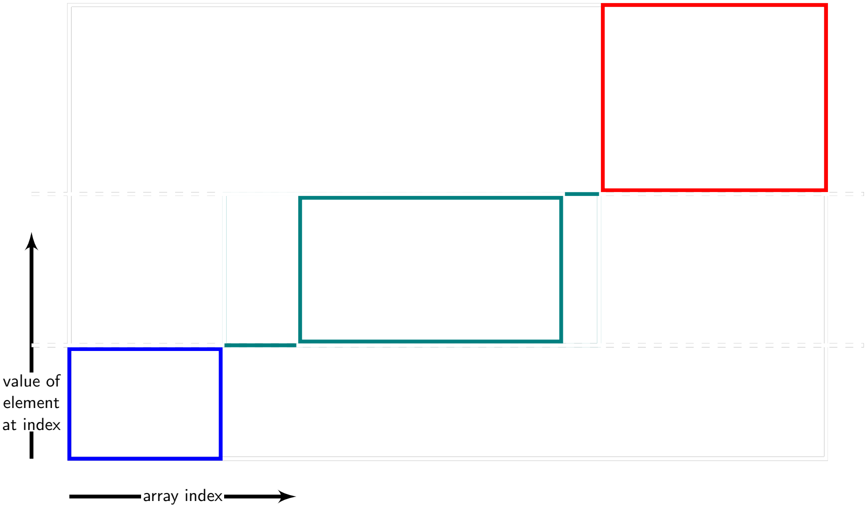
Java Implementation – Dual Pivot Partition



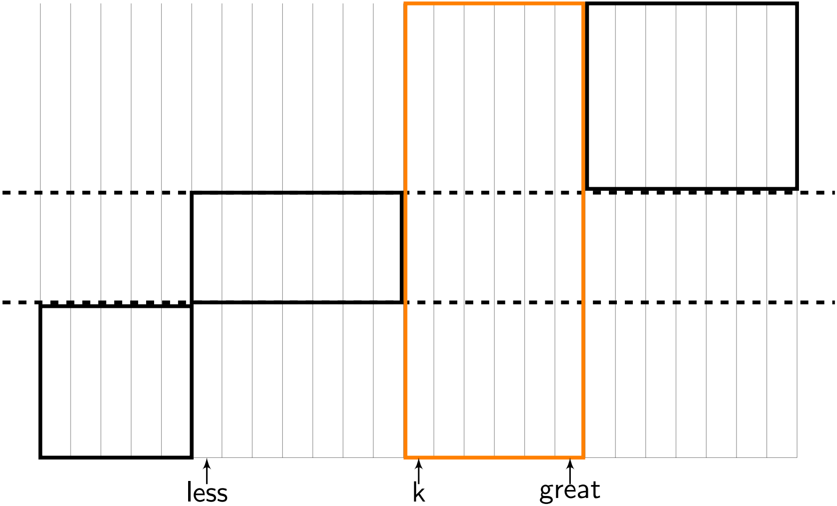
Java Implementation – Swap Pivot Values Partition



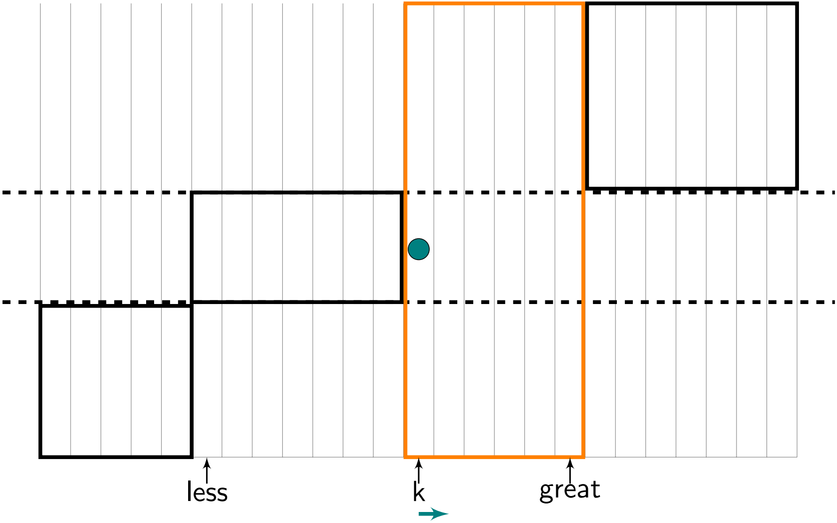
Java Implementation – Swap Pivot Values Partition



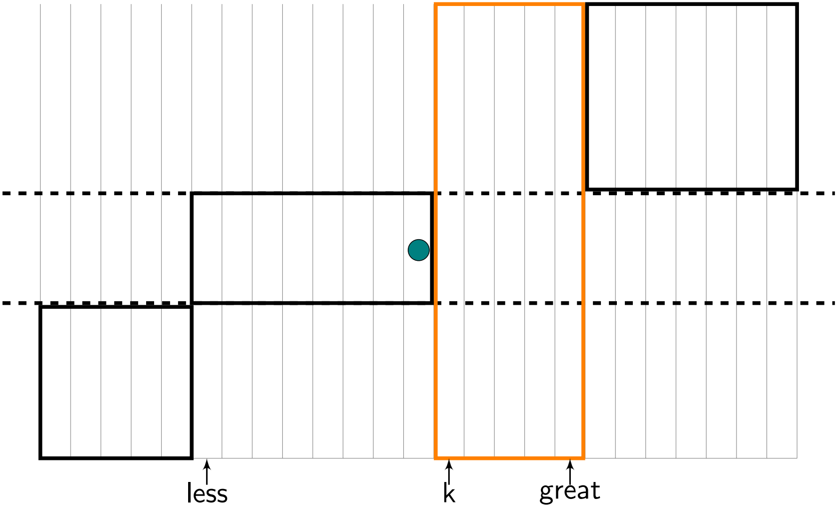
Java Implementation – Partitioning



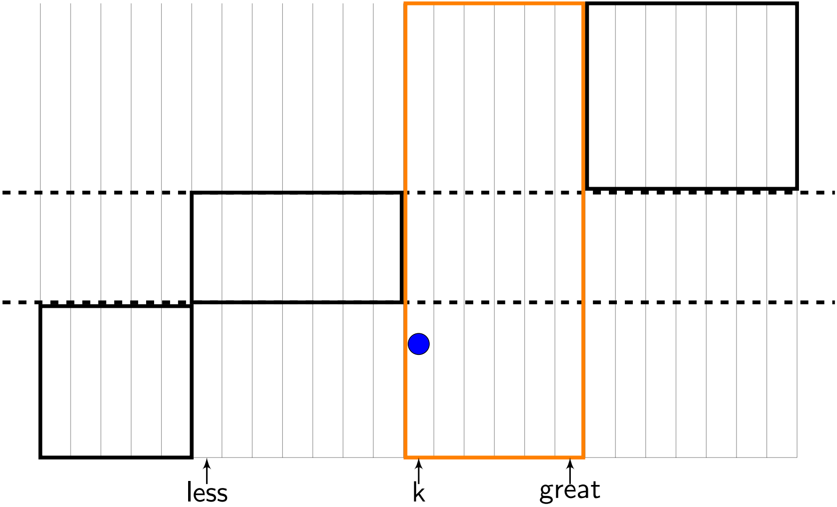
Java Implementation – Partitioning



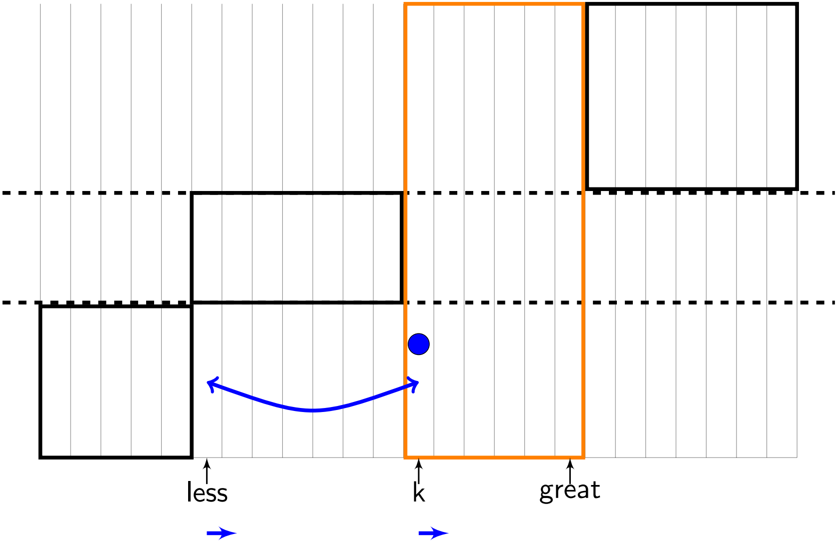
Java Implementation – Partitioning



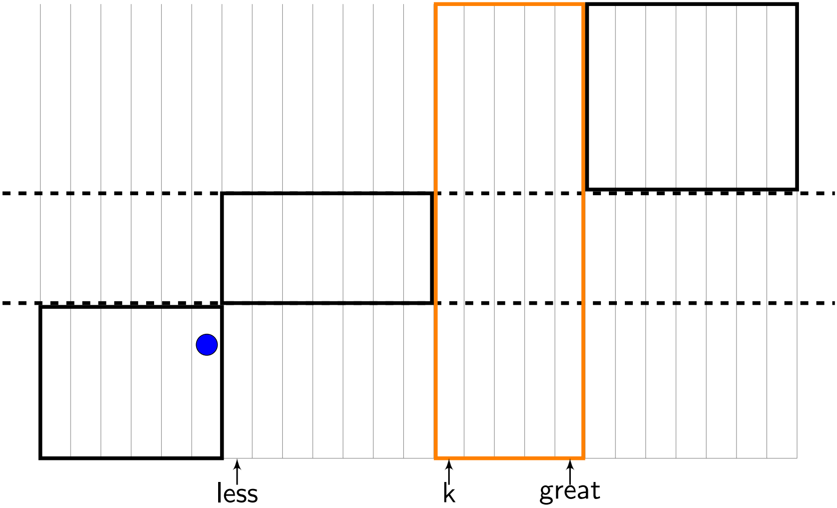
Java Implementation – Partitioning



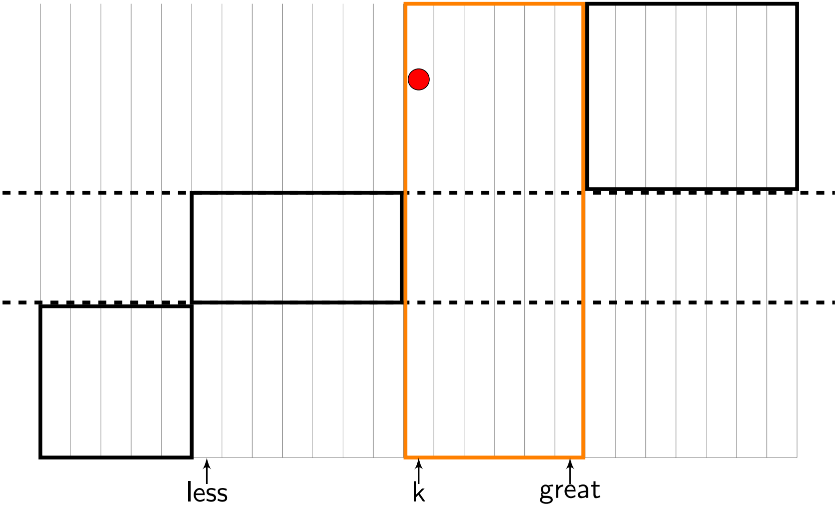
Java Implementation – Partitioning



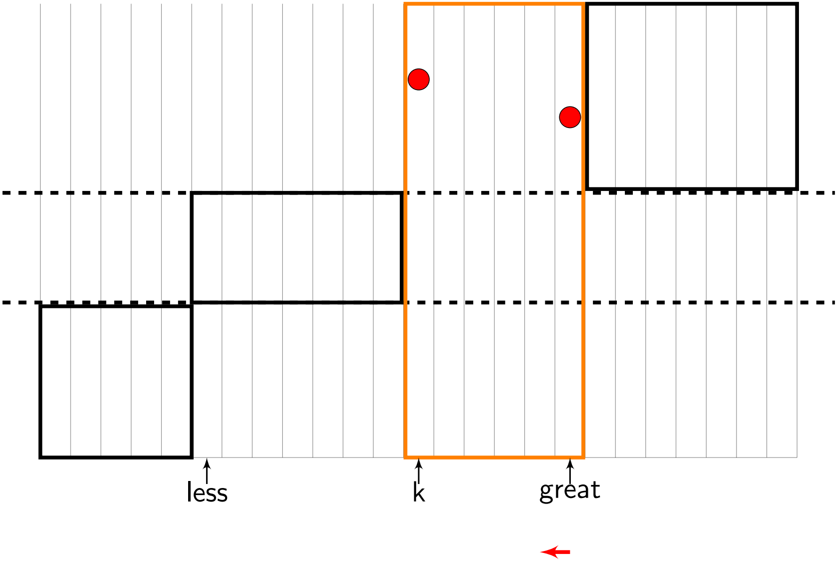
Java Implementation – Partitioning



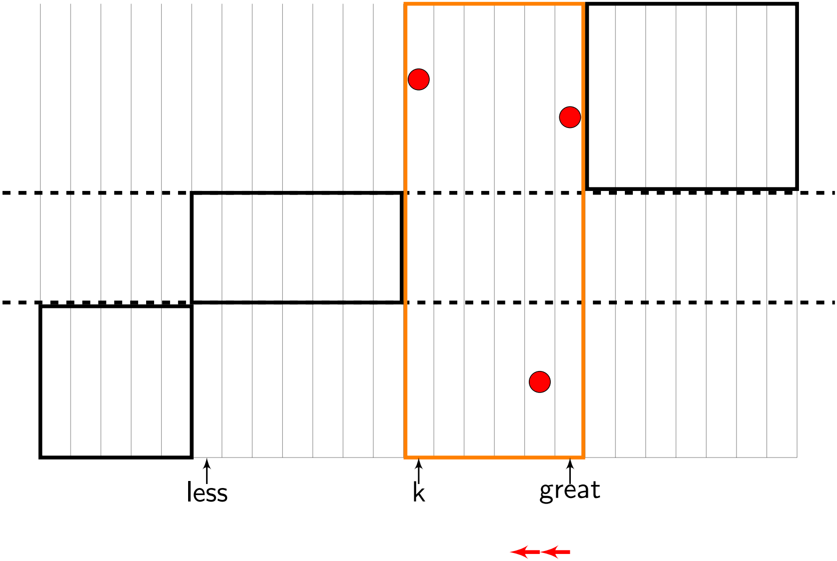
Java Implementation – Partitioning



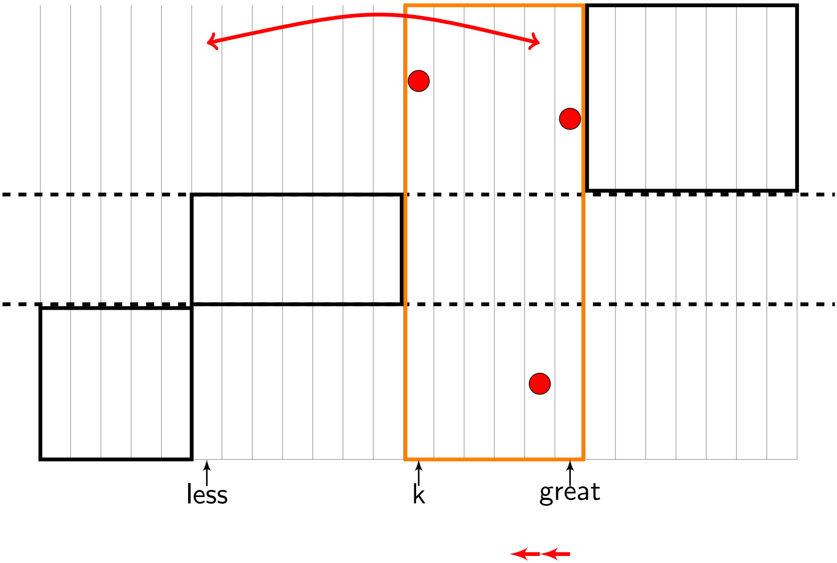
Java Implementation – Partitioning



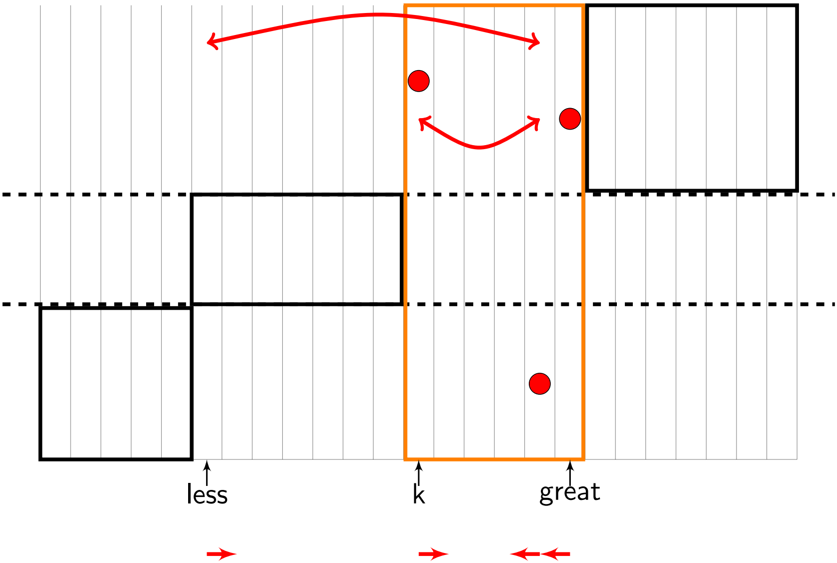
Java Implementation – Partitioning



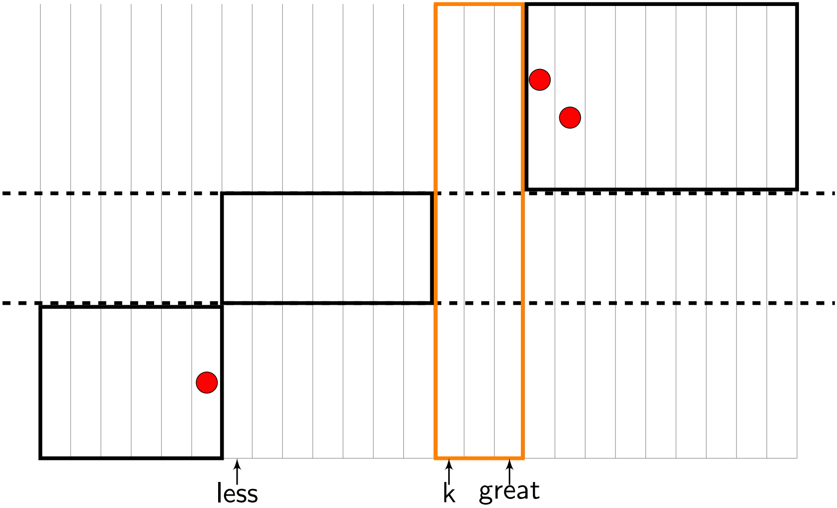
Java Implementation – Partitioning



Java Implementation – Partitioning



Java Implementation – Partitioning



Section 2

Specification and Proof

Work Flow

Work Flow

- Encapsulating source code in its own Java class

Work Flow

- Encapsulating source code in its own Java class
- Subdivision into three classes: One per partitioning style

Work Flow

- Encapsulating source code in its own Java class
- Subdivision into three classes: One per partitioning style
- Writing specification
  Running KeY
  Adapting specification or source code

General KeY Strategy

General KeY Strategy

- Autopilot Strategy Macro

General KeY Strategy

- Autopilot Strategy Macro
- If proof fails:
  - Confirm by generating counterexample
  - Find violated specification condition
  - Adapt specification (or source code)

General KeY Strategy

- Autopilot Strategy Macro
- If proof fails:
  - Confirm by generating counterexample
  - Find violated specification condition
  - Adapt specification (or source code)
- If no proof is found:
  - Increase number of steps (?)
  - Interactive Rule Apps (Quantifier Instantiation, if-then-else-split)
  - Heap Simplification + SMT Solver

Feasibility – Problems with KeY

Feasibility – Problems with KeY

- Computation time

Feasibility – Problems with KeY

- Computation time
  - Method extraction
  - Exact Localization
  - SMT Solver
  - Block Contracts

Feasibility – Problems with KeY

- Computation time
  - Method extraction
  - Exact Localization
  - SMT Solver
  - Block Contracts
- Error in specification or lack of resources?

Feasibility – Problems with KeY

- Computation time
    - Method extraction
    - Exact Localization
    - SMT Solver
    - Block Contracts
- Error in specification or lack of resources?
- Localizability

Feasibility – Problems with KeY

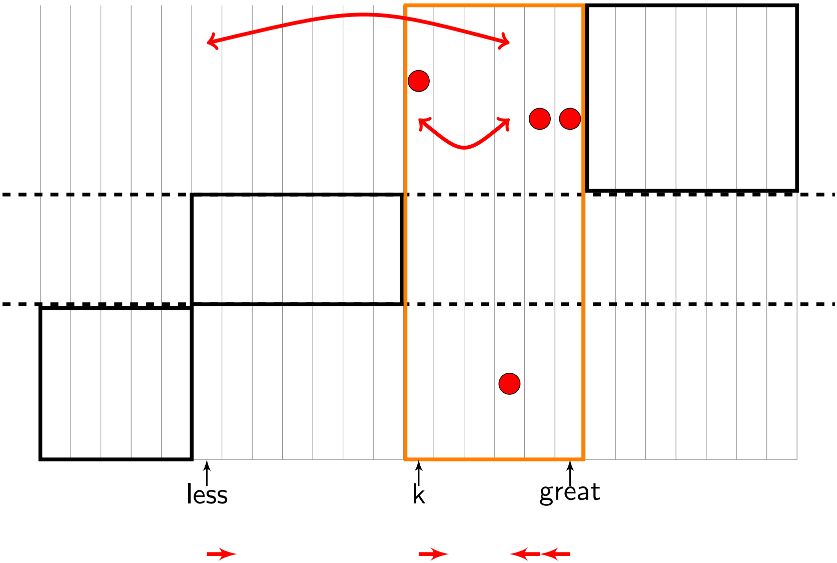
- Computation time
  - Method extraction
  - Exact Localization
  - SMT Solver
  - Block Contracts
- Error in specification or lack of resources?
- Localizability
- Stability

Feasibility – Problems with KeY

- Computation time
  - Method extraction
  - Exact Localization
  - SMT Solver
  - Block Contracts
- Error in specification or lack of resources?
- Localizability
- Stability
- Responsiveness

Violation of Single Pivot Partition Invariant

Violation of Single Pivot Partition Invariant



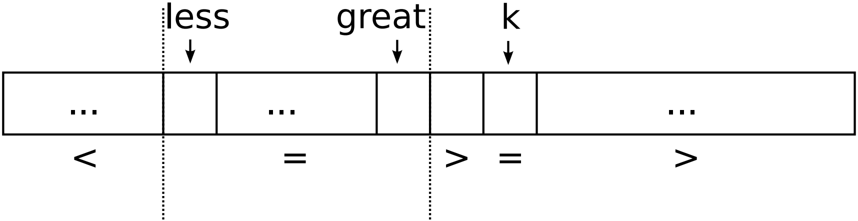
Violation of Single Pivot Partition Invariant

```
while (a[great] > pivot2) {
    if (great-- == k) {
        break outer;
    }
}

while (a[great] == pivot2) {
    if (great-- == k) {
        break outer;
    }
}

while (a[great] > pivot) {
    --great;
}
...
```

Violation of Single Pivot Partition Invariant



Section 3

Conclusive Remarks

Conclusive Remarks

Conclusive Remarks

- Verifying a large, complex, real-world Java program with KeY is feasable, but not without challenges

Conclusive Remarks

- Verifying a large, complex, real-world Java program with KeY is feasable, but not without challenges
- Correct sorting, but invariant is violated

Conclusive Remarks

- Verifying a large, complex, real-world Java program with KeY is feasable, but not without challenges
- Correct sorting, but invariant is violated

Further Work

Further Work

- Prove permutation property

Further Work

- Prove permutation property
- Prove method as-is

Further Work

- Prove permutation property
- Prove method as-is
- Prove entire `sort(int[])` method

Further Work

- Prove permutation property
- Prove method as-is
- Prove entire `sort(int[])` method
- Prove entire `sort` method

Further Work

- Prove permutation property
- Prove method as-is
- Prove entire `sort(int[])` method
- Prove entire `sort` method

Statistics – Single Pivot Partition

Method	Nodes	Branches	Time [s]	Rule Apps	Interactive	SMT
case_right	14784	114	17,7	18919	0	0
split	17609	90	23,8	24189	0	0
sort(array, left, right)	18495	101	18,8	22839	0	0
sort(array)	654	7	0,4	1342	0	0
Total	51542	312	60.7	67289	0	0

Statistics – Swap Pivot Values Partition

Method	Nodes	Branches	Time [s]	Rule Apps	Interactive	SMT
move_great_left	1245	16	0,8	2346	0	0
move_less_right	2120	14	1,8	3224	0	0
swap_values	123636	407	246,6	138039	0	0
Total	127001	437	249.2	143609	0	0

Statistics – Dual Pivot Partition

Method	Nodes	Branches	Time [s]	Rule Apps	Interactive	SMT
calc_indices	24533	8	49,6	24835	0	0
insertionsort_indices	50816	365	137,4	73056	0	34
prepare_indices	5332	28	6,4	7153	0	0
move_great_left	1650	15	1,1	2605	0	0
move_great_in_loop	1580	18	1,1	2787	0	0
move_less_right	1928	14	1,4	2967	0	0
loop_body	52134	287	57,3	56263	18	0
split	28751	98	109,6	51666	0	36
sort(int[],left,right)	51342	305	459,6	76973	114	116
sort(int[])	611	5	0,4	1236	0	0
Total	218677	1143	823,9	299541	132	186
Entire Proof	297220	1892	1133,8	510439	132	186