Dual Pivot Quicksort: Verification and Proof using KeY

Jonas Schiffl

Karlsruher Institut für Technologie

July 27th, 2016
Introduction

Why verify Dual Pivot Quicksort?

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

Why verify Dual Pivot Quicksort?
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

value of element at index

array index
Quicksort

array index

value of element at index

array index
Quicksort

value of element at index

array index
Quicksort

value of element at index

array index
Quicksort

value of element at index

array index
Quicksort

value of element at index

array index
Dual Pivot Quicksort

value of element at index

array index
Dual Pivot Quicksort

array index

value of element at index

array index
Dual Pivot Quicksort

value of element at index

array index
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)
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)
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

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counting Sort</td>
<td>short, char</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>&lt; 29</td>
</tr>
<tr>
<td>Quicksort</td>
<td>long, float, double</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>no</td>
</tr>
</tbody>
</table>

For data types like int, long, float, and double, the highly structured algorithms such as Quicksort or Merge Sort are generally recommended due to their efficiency in handling these types of data.
Java Implementation – Choosing a Sorting Algorithm

1. data type?
   - byte
   - int, long, float, double

2. length?
   - short, char
   - <=29
   - >29
   - >3200

3. length?
   - <=47
   - >47
   - <47

4. highly structured?
   - yes
   - no

5. Counting Sort
6. Insertion Sort
7. Quicksort
8. Merge Sort
Java Implementation – Quicksort

Quicksort

- Select 5 evenly spaced array elements
- Sort elements in their positions
- All 5 elements distinct?
- Single Pivot Partition: no
- Dual Pivot Partition: yes
- Central part large?
- Pivot Values Partition: yes
- Recursion: no
Java Implementation – Quicksort

- Quicksort
  - Select 5 evenly spaced array elements
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

Quicksort → Select 5 evenly spaced array elements

Sort elements in their positions → All 5 elements distinct?

yes → Dual Pivot Partition

no → Single Pivot Partition
Java Implementation – Quicksort

1. Quicksort
   - Select 5 evenly spaced array elements
   - Sort elements in their positions
   - All 5 elements distinct?
     - yes
       - Dual Pivot Partition
       - Central part large?
     - no
       - Single Pivot Partition
Java Implementation – Quicksort

Quicksort → Select 5 evenly spaced array elements

Sort elements in their positions → All 5 elements distinct?
  yes → Dual Pivot Partition
  no → Single Pivot Partition

Dual Pivot Partition → yes → Pivot Values Partition

Pivot Values Partition → yes → Central part large?

Central part large? → no → Single Pivot Partition
Java Implementation – Quicksort

1. Select 5 evenly spaced array elements
2. Sort elements in their positions
3. All 5 elements distinct?
   - yes: Pivot Values Partition
   - no: Single Pivot Partition
4. Dual Pivot Partition
5. Central part large?
   - yes: Pivot Values Partition
   - no: Recursion
Java Implementation – Single Pivot Partition

value of element at index

array index
Java Implementation – Dual Pivot Partition

value of element at index

array index
Java Implementation – Swap Pivot Values Partition
Java Implementation – Partitioning

less

k

great
Java Implementation – Partitioning

less

k

great
Java Implementation – Partitioning

less

k

great
Java Implementation – Partitioning
Java Implementation – Partitioning

less

k

great

←→
Java Implementation – Partitioning
Java Implementation – Partitioning

less

k
great
Section 2

Specification and Proof
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
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

less k great
Violation of Single Pivot Partition Invariant

```c
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

... less great k ...

< = > = >
Section 3

Conclusive Remarks
Conclusive Remarks

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

- Verifying a large, complex, real-world Java program with KeY is feasible, but not without challenges.
Conclusive Remarks

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

- Verifying a large, complex, real-world Java program with KeY is feasible, 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Nodes</th>
<th>Branches</th>
<th>Time [s]</th>
<th>Rule Apps</th>
<th>Interactive</th>
<th>SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>case_right</td>
<td>14784</td>
<td>114</td>
<td>17,7</td>
<td>18919</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>split</td>
<td>17609</td>
<td>90</td>
<td>23,8</td>
<td>24189</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sort(array, left, right)</td>
<td>18495</td>
<td>101</td>
<td>18,8</td>
<td>22839</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sort(array)</td>
<td>654</td>
<td>7</td>
<td>0,4</td>
<td>1342</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>51542</td>
<td>312</td>
<td>60.7</td>
<td>67289</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Method</td>
<td>Nodes</td>
<td>Branches</td>
<td>Time [s]</td>
<td>Rule Apps</td>
<td>Interactive</td>
<td>SMT</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----</td>
</tr>
<tr>
<td>move.great.left</td>
<td>1245</td>
<td>16</td>
<td>0,8</td>
<td>2346</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>move.less.right</td>
<td>2120</td>
<td>14</td>
<td>1,8</td>
<td>3224</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>swap.values</td>
<td>123636</td>
<td>407</td>
<td>246,6</td>
<td>138039</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>127001</td>
<td>437</td>
<td>249,2</td>
<td>143609</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Statistics – Dual Pivot Partition

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

**Entire Proof**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>297220</td>
<td>1892</td>
<td>1133,8</td>
<td>510439</td>
<td>132</td>
<td>186</td>
</tr>
</tbody>
</table>