A Fast, Stable Variant on Quicksort

Lilith Hafner[1] and Kobina Thatcher[1]

[1]Independent Researchers

ABSTRACT

We present a Quicksort partitioning method that is stable and faster than existing partitioning methods on modern architectures at the cost of requiring auxiliary memory.

Keywords

Julia, Performance, Sorting

1. Preliminaries

See [5] for a description of Quicksort and [1] for generalizations to alternative partitioning schemes. This paper assumes familiarity with both. A video presentation of the proposed algorithm is available at youtu.be/RIhCBTx5TYA

2. Partitioning scheme

The proposed partitioning scheme copies data from a source array to a destination array, partitioning it in the process. Compare each element of the source array with the pivot and fill the destination array from both ends, placing elements less than the pivot at the beginning and elements greater than the pivot at the end, as described in Algorithm 1

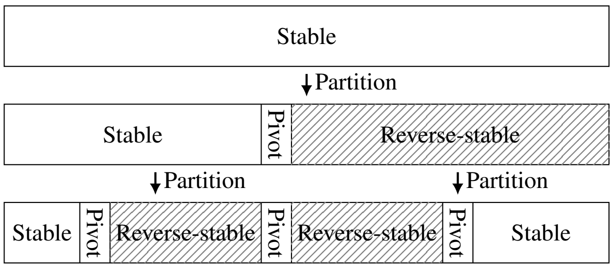
Algorithm 1 Partitioning scheme

```
1:  procedure PARTITION($source$, $destination$, $pivot$)
2:      initialize $start \leftarrow$ pointer to first element of $destination$
3:      initialize $end \leftarrow$ pointer to last element of $destination$
4:      for each $element$ in $source$ do
5:          if $element$ belongs before $pivot$ then
6:              place $element$ at $start$
7:              increment $start$
8:          else
9:              place $element$ at $end$
10:             decrement $end$
11:         end if
12:     end for
13:     return $start$
14: end procedure
```

3. Stability

Stability requires that elements which compare equivalently—neither is less than the other—appear in the output in the same order as in the input. We extend this to intermediary results and say that a block of memory is stable if every pair of equivalent elements in that block appears in the same order as in the input. We also define a notion of reverse-stability and say that a block of memory is reverse-stable if every pair of equivalent elements is in the opposite order as in the input.

After partitioning, elements less than the pivot appear at the start of the destination array in the same order they originally appeared and elements greater than the pivot appear at the end of destination array in reverse order. The reverse-stability of one sub-array is much more conducive to restoring stability than the complex mixing created by Hoare [5] and Lomuto [1] partitioning. We model stability in the Quicksort algorithm formed from the proposed partitioning scheme like so



Each partition splits the input into a stable sub-array and a reverse-stable sub-array where the first sub-array is of the same stability as the partition's input and the second sub-array is the opposite stability. Further, any pair of equivalent elements separated by a pivot always appear in the same order as they do in the input.
Elements less than and greater than the pivot are handled well by Algorithm 1. However, elements equal to the pivot must be placed after the pivot iff they occur after the pivot in the input. This can be handled by a complex "belongs before" comparison which accounts for both position and value but it is more efficient to split the partitioning loop in two. First iterate over elements before the pivot, branching on whether they are less than or equivalent to the pivot element; and then iterate over elements after the pivot, branching on whether they are strictly less than the pivot. Concretely, this may be implemented as Algorithm 2
To stabilize the final result we use a stable sorting algorithm as the base case for stable base cases and a reverse-stable sorting algorithm for reverse-stable base cases. Reverse-stable sorting algorithms are not the focus of this paper and can easily be formed by reversing an input and then applying a stable sorting algorithm such as insertion sort.

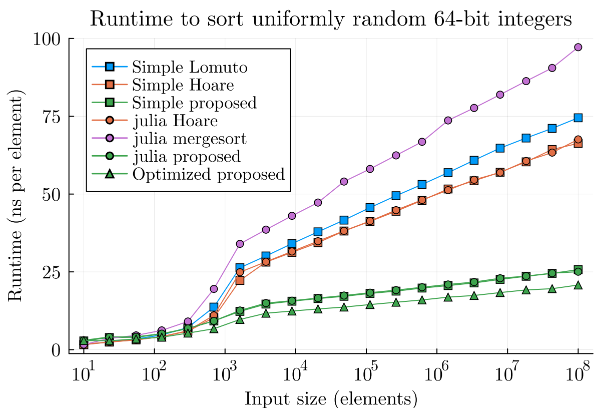
4. Empirical performance

To measure the practical efficiency of this partitioning scheme we implement it and existing partitioning schemes in the Julia [2] programming language.
We benchmark simple Quicksort implementations using the Hoare, Lomuto, and proposed partitioning schemes, as well as algorithms from Julia's standard library. All benchmarked algorithms use Insertionsort as a base case for 20 or fewer elements. Only the proposed algorithms and Mergesort are stable.

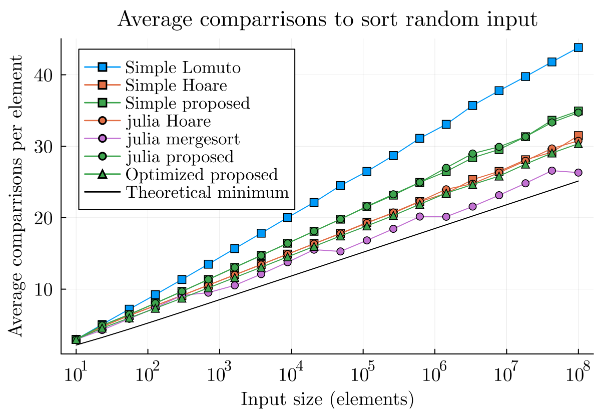
Algorithm 2 Stable partitioning scheme

```
1:  procedure PARTITION($source$, $destination$, $pivot\_index$)
2:      initialize $pivot \leftarrow source[pivot\_index]$
3:      initialize $start \leftarrow$ pointer to first element of $destination$
4:      initialize $end \leftarrow$ pointer to last element of $destination$
5:      for each $element$ in $source$ before $pivot\_index$ do
6:          if $element \leq pivot$ then
7:              place $element$ at $start$
8:              increment $start$
9:          else
10:             place $element$ at $end$
11:             decrement $end$
12:         end if
13:     end for
14:     for each $element$ in $source$ after $pivot\_index$ do
15:         if $element < pivot$ then
16:             place $element$ at $start$
17:             increment $start$
18:         else
19:             place $element$ at $end$
20:             decrement $end$
21:         end if
22:     end for
23:     place $pivot$ at $start$ (which is now equal to $end$)
24:     return $start$
25: end procedure
```

Additionally, we benchmark an optimized implementation of Quicksort using the proposed partitioning scheme. These optimizations include manual stack—or nest, as Hoare calls it—management, stable median-of-three pivot selection, and a base case that integrates reversing and copying with insertion sort.



As input size increases, the proposed partitioning scheme significantly outperforms existing systems, despite the additional memory usage and time taken to allocate scratch space. This improvement over prior art dwarfs additional improvements such as median of three partitioning, nest management, and some base-case optimization.
The relative performance of the discussed algorithms are similar for various bitwidth integers and floating point numbers, as well as records containing a key that does not occupy the entire record. However, there are cases with different trends. Notably, when the data are already sorted, reverse sorted, or all keys are identical. In these cases, Hoare partitioning reliably but marginally outperforms the proposed partitioning scheme.



Additionally, Mergesort performs fewer comparisons than any of the studied partitioning schemes. While Mergesort is significantly slower than the proposed partition scheme when comparisons are cheap, when comparisons are substantially more resource-intensive Mergesort will outperform all studied Quicksorts.

4.1 Reproducibility

The data were generated on a dedicated 2022 Apple M2 system with 8GB of memory and are qualitatively similar to results produced on a shared ProLiant DL325 Gen10 Plus v2. They can be reproduced on additional hardware by installing the Julia package hosted at github.com/LilithHafner/QuickerSort.jl and running `QuickerSort.reproduce_figures()`.

5. Deployment and reference implementations

The proposed partitioning scheme is used in Julia's the default sorting algorithm due to superior performance over the prior Hoare partitioning method[4]. The implementations used for benchmarking are available at github.com/LilithHafner/QuickerSort.jl.

6. Role in an in-place sorting algorithm

All algorithms require $\Omega(1)$ auxiliary memory. This memory can be used as scratch space for the proposed partitioning scheme provided the length of inputs provided to that partitioning scheme is bounded. This enables the use of the proposed algorithm as a base case for stable, in-place, divide and conquer sorting algorithms such as paged mergesort [3]

7. References

[1] Jon Bentley. *Programming Pearls (2nd Edition)*. Addison-Wesley Professional, 2 edition, 1999.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[3] S. Dvořák and B. Ďurian. Towards an efficient merging. In Jozef Gruska, Branislav Rovan, and Juraj Wiedermann, editors, *Mathematical Foundations of Computer Science 1986*, pages 290–298, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[4] Lilith Hafner, Oscar Smith, Stefan Karpinski, Petr Vana, and Gaurav Arya. Stabilize, optimize, and increase robustness of quicksort. *GitHub*, 2022.

[5] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 01 1962.