A faster and adaptive partition function for Quicksort*

Victor Campos[1], Israel Mendes[1], Rodrigo Nogueira[1], Wladimir Tavares[1]

[1]Grupo ParGO, Universidade Federal do Ceará (UFC)

victoitor@ufc.br, israelmendes@alu.ufc.br,

nogrrodrigo@alu.ufc.br, wladimir@dc.ufc.br

Abstract. *The standard implementations of the Quicksort partitioning function are due to Hoare and Bentley–McIlroy. We discuss how the best performing algorithm is related to the number of repeated keys on the input and verify this experimentally. We extend our analysis to a third partitioning algorithm, which we call the Double Hoare algorithm. It has a similar behaviour to Bentley–McIlroy in comparison to the Hoare algorithm, but it performs better than the usual performance metrics indicate. Using this information, we modify it into an algorithm that dynamically adapts to the number of repeated keys and beats library implementations of the Hoare and Bentley–McIlroy independent of the key repetition density.*

1. Introduction

Sorting sequences of elements is a fundamental problem in computer science and plays a central role in a wide range of applications. Among comparison-based algorithms, *Quicksort* [Hoa62] has long been regarded as the method of choice due to its excellent practical performance.

A crucial factor in the efficiency of any Quicksort implementation is the choice of an effective partitioning algorithm. Hoare's original partitioning scheme [Hoa62] performs particularly well when all keys are distinct and remains widely used, for example in the `glibc` implementation of `qsort`. In the presence of repeated keys, this scheme can be adapted to avoid expected quadratic behaviour [Sed77]. However, this guarantee comes at the cost of wasting comparisons, since equal elements are repeatedly compared across recursive calls rather than being grouped and placed in their final positions.

Bentley and McIlroy [BM93] proposed a practical *fat partitioning* strategy that places all elements equal to the pivot into their final positions within a single partitioning step, in contrast to the classical *thin partition*, which isolates only one pivot copy per iteration. Their algorithm provided a novel and efficient solution to the Dutch national flag problem [Dij76]. Although earlier solutions for this problem had been proposed [McM78], they were not sufficiently efficient to be used as partitioning methods within Quicksort [Sed77]. As a result, Bentley–McIlroy became a cornerstone of several widely used sorting libraries, including the one used by FreeBSD.

Our contributions. Our first contribution is to reintroduce and analyze a fat partitioning algorithm originally proposed as a solution to the Dutch national flag problem, but never analyzed in the context of Quicksort [Che05], which we refer to as the *Double Hoare*

*Partially supported by CNPq Proc 404479-2023-5 and CAPES - Código 001.

partition. This algorithm incurs more comparisons and exhibits worse cache behaviour, but may perform fewer swaps than the Bentley–McIlroy partition. Although this suggests limited practical appeal, we observe it is competitive in practice, a fact that reopens the discussion on the parameters required for an accurate cost model for Quicksort.

In addition, we propose an adaptive version of the Double Hoare partition that dynamically changes its behaviour depending on the sequence of sampled pivots, similar to pattern-defeating Quicksort [Pet21]. When considering single-pivot Quicksort implementations, the choice of an optimal partitioning strategy depends strongly on the number of repeated keys in the input: Bentley–McIlroy's partition is preferable when there are many duplicates, whereas Hoare's partition performs better when most keys are distinct. Our adaptive approach bridges this gap and performs better across all input distributions.

2. Partitioning algorithms

For each algorithm, suppose a pivot element has already been chosen.

Hoare thin partition. The procedure works by keeping two pointers b and c that start from the left and the right of the array, moving towards each other until they cross. Pointer b moves right until it finds a key with value at least that of the pivot. Then, c moves left until it finds a key with value at most that of the pivot. When both pointers stop their keys are swapped and the loop continues until they cross. Fig. 1a illustrates this invariant.

Bentley–McIlroy fat partition. It follows Hoare's partition but, as pointers b and c move closer to each other, they reallocate copies of the pivot to the ends of the array. By keeping two other pointers a and d that mark the contiguous segments of pivot copies, they are able to move them into feasible positions after b and c have crossed. Fig. 1b illustrates this invariant.

Double Hoare fat partition. This algorithm consists of two modified iterations of Hoare's algorithm. On the first iteration, all pivot copies are put into the left subarray. Once b and c cross, b is used to mark the crossing point and the second iteration is restricted to the left subarray, which contain all elements smaller than or equal to the pivot. On the second iteration, all pivot copies are put into the right subarray. Fig. 1c illustrates this invariant.

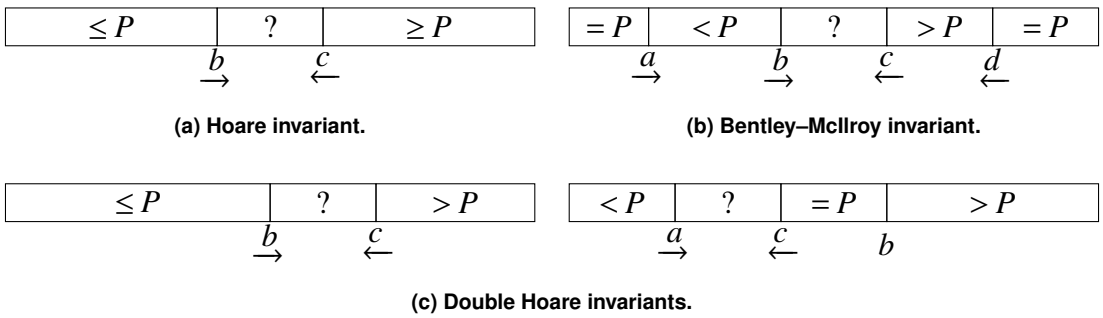

Figure 1. Sketch of partition invariants.

3. Theoretical analysis of partitioning algorithms

The results in this section consider sorting an array on n keys chosen in the random u-ary model. The *random u-ary model* consists of choosing each key independently and

uniformly at random from a universe of u distinct keys, with $u = O(n)$. This model allows for repeated keys, which does not happen on the classic *random permutation model*. The array is sorted using Quicksort where each pivot is chosen uniformly at random from among the indices of each subarray. We consider the variations of using the three partitioning algorithms of Sec. 2.

Hoare is the only thin partition, and it performs fewer operations compared to the other two algorithms. When all keys are distinct, there is no difference between a fat and a thin partition, so Hoare is expected to be faster in practice. When there are repeated keys, the expected cost of both fat partitioning algorithms is $\Theta(n \log u)$ which is strictly better than the expected cost $\Theta(n \log n)$ of Hoare [Sed77]. These fat partitioning algorithms are *instance optimal*, as they have a running time cost of $O(n\mathcal{H})$ and $n\mathcal{H} - n$ is a lower bound for sorting a random permutation of an array of size n with *entropy of repetition* $\mathcal{H}$ [SB02].

To compare the two fat partitioning algorithms, we consider the three main metrics studied for Quicksort, which are the number of comparisons, the number of swaps and the number of scanned elements. We remark that the number of scanned elements has only gained importance in the last decade as the other metrics could not describe the cache behaviour observed in practice [NWM16]. For comparisons, we consider ternary comparisons as it's the most natural for library implementations of sorting algorithms.

Theorem 1. *The expected number of ternary comparisons and of scanned elements of Quicksort using Bentley–McIlroy is* $2n \ln u + O(n)$ *and using Double Hoare is* $3n \ln u + O(n)$.

Theorem 2. *If two executions of Quicksort using Bentley–McIlroy or Double Hoare sample the same sequence of pivots, then the expected number of swaps performed by Double Hoare is at most that of Bentley–McIlroy and their difference is at most* $2n$.

The theoretical analysis indicates Bentley–McIlroy is better than the Double Hoare partition both in the number of comparisons and scanned elements while being worse on the number of swaps, but only by a linear term. In contrast to the theoretical analysis, empirical tests suggest that the Double Hoare partition has a similar running time. Motivated by this observation, we propose modifications to the Double Hoare partition to produce an algorithm which adapts to the presence of repeated keys.

4. Cearense partition and experimental analysis

The main idea in this new adaptive variant, which we call the *Cearense partitioning algorithm*, is to perform only the first iteration of the Double Hoare method whenever the second iteration does not seem necessary. When all elements smaller than or equal to the pivot are put in the left subarray, we sample another pivot (in this subarray) and the second iteration is only made when this new pivot has the same key as the original pivot.

This leads to savings when there are few repeated keys. There is a minimal overhead for identifying when both pivots are equal, as this is done using a single comparison, but we get a major improvement by using the pivots as sentinels, which allows for many optimizations. Furthermore, by using two symmetric versions of the partitioning algorithm, we produce inner loops which are as efficient as the Hoare algorithm.

For the experimental analysis, we implement several known optimizations for Quicksort. We use heapsort if the recursion depth reaches a logarithmic threshold, insertion sort for small subarrays, median-of-3 deterministic pivoting, and tail recursion elimination.

Each partitioning algorithm uses a version of insertion sort on small subarrays which is best suited for the produced partition. The usage of pivot sentinels allows our new version to use two symmetric versions of insertion sort which are more efficient in practice.

We run empirical tests using the random u-ary model to generate input arrays. Each input is sorted by several Quicksort implementations, identified by the partitioning algorithms used. We run these experiments on AMD and Intel CPUs to account for the difference in cache performance. The algorithms and input data can be found in [CMNT26].

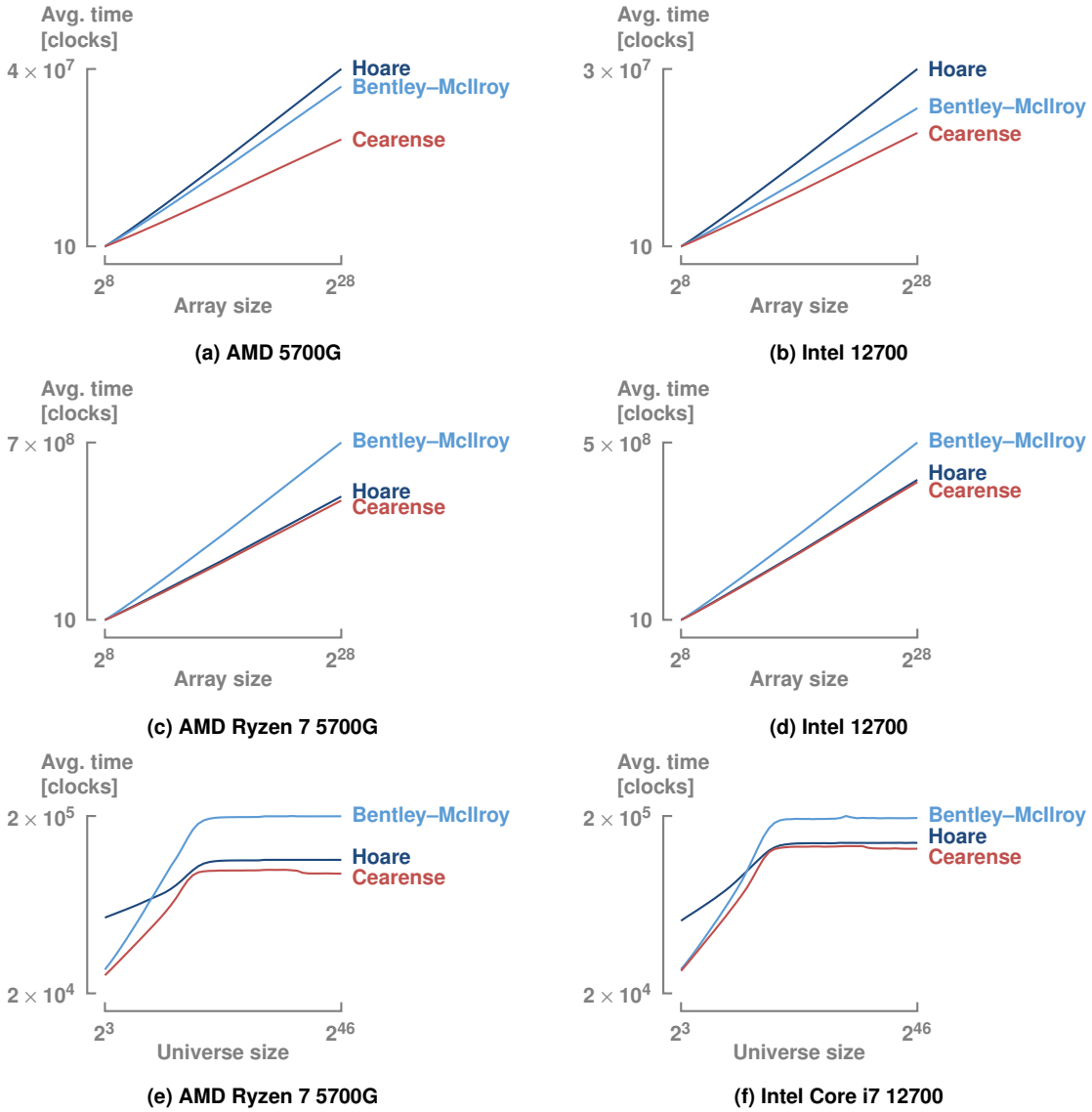

Figure 2. Experimental results

Figs. 2a and 2b have $u = \sqrt{n}$, Figs. 2c and 2d have $u = n$, and they all have $n \in \{2^8, 2^9, \ldots, 2^{28}\}$. Figs. 2e and 2f have $u \in \{2^3, 2^4, \ldots, 2^{46}\}$ and $n = 2^{20}$. We remark that Bentley–McIlroy partition is indeed faster than the Hoare partition on smaller values of u where more duplicates occur. This behaviour is inverted with fewer duplicates when u is larger. It can be observed that the Cearense partition indeed adapts to the number of repeated keys and is the fastest choice in all studied scenarios.

References

[BM93] J. Bentley and M. McIlroy. "Engineering a sort function". In: *Software: Practice and Experience* 23.11 (1993), pp. 1249–1265. DOI: 10.1002/spe.4380231105.

[Che05] W. Chen. "Probabilistic analysis of algorithms for the Dutch national flag problem". In: *Theoretical Computer Science* 341.1 (2005), pp. 398–410. DOI: 10.1016/j.tcs.2005.03.047.

[CMNT26] Victor Campos, Israel Mendes, Rodrigo Nogueira, and Wladimir Tavares. *A faster and adaptive partition function for Quicksort*. Online repository. 2026. URL: https://github.com/israelnicolas940/Variantes-Quick-Sort/releases/tag/ETC2026.

[Dij76] E. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976. ISBN: 978-0132158718.

[Hoa62] C. Hoare. "Quicksort". In: *The computer journal* 5.1 (1962), pp. 10–16. DOI: 10.1093/comjnl/5.1.10.

[McM78] C. McMaster. "An analysis of algorithms for the Dutch National Flag Problem". In: *Commun. ACM* 21.10 (1978), pp. 842–846. DOI: 10.1145/359619.359629.

[NWM16] M. Nebel, S. Wild, and C. Martínez. "Analysis of pivot sampling in dual-pivot quicksort: A holistic analysis of Yaroslavskiy's partitioning scheme". In: *Algorithmica* 75 (2016), pp. 632–683. DOI: 10.1007/s00453-015-0041-7.

[Pet21] O. Peters. *Pattern-defeating Quicksort*. 2021. arXiv: 2106.05123. URL: https://arxiv.org/abs/2106.05123.

[SB02] R. Sedgewick and J. Bentley. "Quicksort is Optimal". KnuthFest. 2002. URL: https://sedgewick.io/wp-content/uploads/2022/03/2002QuicksortIsOptimal.pdf (visited on 03/01/2026).

[Sed77] R. Sedgewick. "Quicksort with equal keys". In: *SIAM Journal on Computing* 6.2 (1977), pp. 240–267. DOI: 10.1137/0206018.