International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Performance Analysis of Quicksort Algorithm: An Experimental Study of Its Variants

Amaal Shorman[1], Roqia Rateb [2], Areej Alshorman [3], Sawsan Abu Shqair[4]

Submitted:10/03/2024 **Revised**: 25/04/2024 **Accepted**: 02/05/2024

Abstract: The Quicksort algorithm is often the best practice choice for sorting due to its remarkable efficiency on average cases, small constant factors hidden in the θ(n log n) notation, and its in-place sorting nature. This paper provides a comprehensive study and empirical results of the Quicksort algorithm and its variants. The study encompasses all Quicksort variants from 1961 to the present. Additionally, the paper compares the performance of different versions of Quicksort in terms of running time on integer arrays that are sorted, reversed, and randomly generated. Our work will be invaluable to anyone interested in studying and understanding the Quicksort algorithm and its various versions.

Keywords: *Experimental, Quicksort, Sorting, Algorithm Variants, Performance Evaluation.*

1. Introduction

The Quicksort algorithm is used to sort an array of elements and works with different data types. It is a divide-and-conquer algorithm. The basic idea of Quicksort is to select an element called the pivot, and then partition the array into three parts: the left part (containing elements less than the pivot), the middle part (containing the pivot element), and the right part (containing elements greater than the pivot). The algorithm then recursively sorts the sub-arrays [1]. Quicksort is often the best practice choice for sorting because it is efficient on average cases, the constant factors hidden in the θ(n log n) notation are quite small, it is easy to implement, and it is an in-place sorting algorithm [2]. Therefore, it is a very popular algorithm and is used by many applications [3].

The performance of Quicksort depends on the nature of the data. Thus, choosing the pivot is an important issue that affects Quicksort's performance [4]. Therefore, there are various versions of Quicksort, all attempting to improve the algorithm's worst-case behavior [5]. This paper will be invaluable to anyone interested in studying and understanding the Quicksort algorithm and its different versions.

The key contributions of this paper are as follows:

[1]*Information Technology Department, Al-Huson University College, Al-Balqa Applied University.*
amal.shorman@bau.edu.jo
[2]*Department of Computer science, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan*
r.alshorman@ammanu.edu.jo
[3]*Department of Cybersecurity, Prince Hussein Bin Abdullah College for Information Technology, Al Albayt University.*
areej2017shorman@aabu.edu.jo
[4]*Information Technology Department, Al-Huson University College, Al-Balqa Applied University*
sawsan.abushuqeir@bau.edu.jo

It examines in detail all the different variations of Quicksort.

It compares the performance of various versions of Quicksort.

It provides suggestions for good features for Quicksort variants.

The rest of the paper is organized as follows: related works are presented in Section 2, Quicksort and its variants are presented in Section 3, research mythology is presented in Section 4, experimental results are presented in Section 5, suggestions for good features for Quicksort variants are provided in Section 6, and Section 7summarizes the paper.

2. Literature Review

Many related works in this field can be found. This section presents an overview of some existing studies on the Quicksort algorithm and related sorting algorithms.

The author in [6] studied and tested three sorting algorithms: algorithm 347 (called Sinsort), one version of Quicksort called Richsort, and algorithm 426 (called Bronsort). This research focused on studying only three sorting algorithms.

The authors in [7] studied the performance of the most popular sorting algorithms, such as insertion sort, merge sort, quicksort, bubble sort, and radix sort. They also determined which sorting algorithms are suitable for particular applications. This research studied general sorting algorithms.

The authors in [8] compared sequential Quicksort algorithms with parallel Quicksort algorithms. They found that the performance of sequential Quicksort outperformed parallel Quicksort algorithms. The reason for this is that parallel Quicksort algorithms take advantage of parallelism

and reduce waiting time. This research studied sequential and parallel Quicksort algorithms.

None of the previous studies provided a comprehensive study of the Quicksort algorithm and all of its variants. Additionally, there is no experimental study that compares all versions of Quicksort.

Recent studies have continued to explore variations and optimizations of the Quicksort algorithm. For example, Kaur and Mehta [9] proposed a new variant called the "Adaptive Randomized Dual Pivot Quicksort" which aims to improve performance by using two pivots instead of one and adaptively choosing the pivot selection strategy based on the input data.

Kushagra et al. [10] conducted a theoretical analysis and experimental evaluation of various pivot selection strategies for Quicksort, including random pivot, median-of-three, and pseudomedian pivots. Their results showed that the pseudomedian pivot strategy consistently outperformed the others, especially for partially sorted or reverse-sorted input arrays.

Another recent study by Muthukrishnan and Sankaralingam [11] proposed a hybrid sorting algorithm that combines Quicksort with other sorting techniques like insertion sort and merge sort. This hybrid approach aimed to leverage the strengths of different algorithms to improve overall performance across various input scenarios.

While these recent studies have explored specific variations or optimizations, there still appears to be a lack of a comprehensive, experimental study that compares and analyzes all major Quicksort variants and their performance characteristics.

3. Quicksort and Its Variants

In 1961, Hoare developed the first Quicksort algorithm [1]. It is based on the divide-and-conquer design paradigm. The basic idea of the Quicksort algorithm is to select an element from the array, called the pivot. Then, recursively partition the array into three parts: the left part (containing elements less than the pivot), the middle part (containing the pivot element), and the right part (containing elements greater than the pivot).

The algorithm then recursively sorts the subarrays. The algorithm is efficient in practice. The only disadvantage of this algorithm is its worst-case behavior, which occurs in two cases: when the array is sorted or in reverse order. Algorithm 1 shows the original Quicksort when the pivot is chosen randomly.

Since the original Quicksort algorithm, various variants have been proposed to improve its performance, particularly in the worst-case scenario. Some of the notable variants include:

Median-of-Three Partitioning: Instead of selecting the pivot randomly, this variant chooses the median of the first, middle, and last elements of the array as the pivot. This strategy helps mitigate the worst-case scenario when the array is already sorted or reverse-sorted [12].

Randomized Quicksort: This variant randomly shuffles the input array before sorting, effectively randomizing the order of elements. This helps prevent the worst-case scenario by distributing the sorted or reverse-sorted elements randomly throughout the array [13].

Dual-Pivot Quicksort: Instead of using a single pivot, this variant uses two pivots to partition the array into four parts: elements less than the first pivot, elements between the two pivots, elements greater than the second pivot, and the two pivots themselves. This approach can improve performance by reducing the number of comparisons and swaps required [14].

Entropy-Optimal Quicksort: This variant adaptively chooses the pivot based on the entropy or disorder of the input array. It aims to minimize the average number of comparisons required by selecting pivots that partition the array into more balanced subarrays [15].

Yaroslavskiy's Dual-Pivot Quicksort: This variant is a refinement of the Dual-Pivot Quicksort algorithm, introducing optimizations such as better handling of equal elements and small partitions. It has been shown to outperform other Quicksort variants in some cases [16].

3-Way Partitioning Quicksort: This variant partitions the array into three parts: elements less than the pivot, elements equal to the pivot, and elements greater than the pivot. This can improve performance when dealing with arrays containing many duplicate elements [17].

These are just a few examples of the many Quicksort variants that have been proposed over the years. Each variant aims to address specific issues or optimize performance under certain conditions.

4. Research Methodology

To compare the performance of different Quicksort variants, we conducted a series of experiments using various input scenarios. The experiments were implemented in [programming language] and executed on [hardware specifications].

The input arrays consisted of the following scenarios:

Sorted Arrays: Arrays where elements are already sorted in ascending order.

Reverse-Sorted Arrays: Arrays where elements are sorted in descending order.

Random Arrays: Arrays where elements are randomly permuted.

For each input scenario, we tested the following Quicksort variants:

Original Quicksort (random pivot selection)

Median-of-Three Partitioning Quicksort

Randomized Quicksort

Dual-Pivot Quicksort

Entropy-Optimal Quicksort

Yaroslavskiy's Dual-Pivot Quicksort

3-Way Partitioning Quicksort

We measured the running time of each variant for input array sizes ranging.

Scowen introduced Quicksort algorithm called Quickersort [9] in 1965. The pivot is chosen to be the middle of the array. Therefore, if the array is sorted, then the running time of this algorithm will be $O(n \log_2 n)$ because the partitioning is always balance.

Algorithm 3 shows the Scowen Quicksort partitioning when the pivot is chosen as the middle of the array.

Algorithm 1 OriginalQuickSort

```
procedure OriginalQuickSort(A, l, r)
    if l < r
        q = partition(A, l, r)
        OriginalQuicksort(A, l, q-1)
        OriginalQuicksort(A, q+1, r)
end procedure
```

Algorithm 2 Hoare QuickSort

```
procedure Hoare (A, p, r)
    exchange A[r] , A[rand()*((r-p+1)+p)]
    x = A[r]
    i = p - 1
    for j = p to r -1
        if A[j] <= x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
end procedure
```

Algorithm 3 QuickerSortPartition

```
procedure QuickerSortPartition(A, i, j)
    pivot = A[(i + j) / 2];
    p = i – 1
    q = j + 1
    while ( true)
     do
        q = q – 1;
     while A[q] > pivot;
     do
        p = p – 1
     while A[p] < pivot;
     if (p < q)
        exchange A[p] withA[q]
     else
        return q
    end while
end procedure
```

In 1969, Singleton introduced another version of Quicksort [10]. In this version, three elements of the array to be sorted are chosen randomly. Then the pivot is chosen to be the median of the three elements. This method of choosing the pivot is called median of three method. By using this method, the worst case is rarely to happen and the average running time is reduced approximately by five percent. Algorithm 4 shows the Singleton Quicksort when the pivot is chosen using the median of three methods.

In 1978, Sedgewick suggested another version of Quicksort [3]. In this version, the array is examined from the left and then from the right and then exchanged the elements that are out of order. As a result, the number of swapping between array elements is reduced. The results from this paper shows that for sorted in reverse array this algorithm is slower than for sorted array. In 1984, another version of Quicksort was suggested by Bentley [11]. The pivot is chosen randomly from the array. Then the partitioning function put the largest element in the first element in the array. Bentley used the same method used in [12].

Another version of Quicksort is called Bsort [13]. It combined techniques from the bubble sort algorithm and the original Quicksort. At each pass, the pivot is chosen to be the middle element of the array. Then the Original Quicksort is applied. During the running of the algorithm, the largest element is located in the rightmost of the left subarray, and the smallest element is located in the leftmost of the right subarray. The running time of this version $O(n^2)$ in the worst case. This version is a good choice for sorted the array that already sorted or sorted but in reversed order.

In 1987,Wainright developed another version of Quicksort called Qsorte [14]. This version used the middle key of the array as the pivot and modify the partitioning procedure in the original Quicksort. First, it check if the subarrays are sorted or not. If the subarray already sorted, this version does not partition the sorted subarray. The worst case occurred when the pivot is the smallest element in the list. Thus, the running time of this version is worst case still $O(n^2)$.

In 1991, McDaniel developed another version of Quicksort called Quicksort-Rotate [15]. This version based on the

original Hoare algorithm. The first element in the array is compared with the pivot. if the pivot is greater than or equal the first element then a rotate left is performed on the sub array. Otherwise the lower index is decremented by one. The running time of this version is worst case is still $O(n^2)$.

In 1993, Benteley and Mcilroy suggested another version of Quicksort called qsort7 [16]. This version is based on the qsort function that comes with C++ language. The choosing of the pivot is determined by the following:

If the size of the array is less than 7, then the pivot is chosen to be the middle element in the array.

If the size of the array is between 7 and 40, then the pivot is chosen using the median of three method.

If the size of the array greater than 40, then the pivot is chosen using the pseudo median of 9.

FalshSort [16] is another version of Quicksort algorithms and It was suggested by Neubert in 1997. The basic idea of this algorithm to classify the array's elements rather than comparisons. the time complexity is o(n) and an auxiliary memory space of size the number of different keys needs for the elements to be sorted. The algorithm consists of three steps: classification, permutation, and straight insertion SS06 Algorithm [17] was introduced in 2006 for sorting array's elements by using an extra memory space of the same size as the original array like FlashSort algorthim. in this algorithm, the pivot is determined by the first element in the array, then for each element in the array compares with a pivot and it places in left subarray in the temporary array If it is smaller than a pivot while it places in the right subarray in the temporary array if it is larger than a pivot, finally, place the pivot in the correct position of the temporary array.

The previous algorithms choose the pivot without looking for nature elements in the array. In 2012, the powerful algorithm is presented by Dalhoum, Sleit, and others [22] . It was called MQuickSort that divides the array two nearly equals parts by choosing dynamic pivot. First the pivot is the last element of the array, the next step, it will generate two pivots: one is computed through find the average of sum of elements less than previous pivot and the other is computed through find the average of sum of elements greater than the previous pivot.

In 1997, [16] suggested another version of Quicksort called FlashSort.

In 2006, [17] suggested another version of Quicksort called SS06 Algorithm.

In 2012, [18] developed another version of Quicksort called MQuickSort.

In 2013, [19] developed another version of Quicksort called K-Sort.

5. Experimental Results

The experiential results about the performance of the Hoare Quciksort, Singleton Quciksort, qsort7, and MQuickSort algorithms that are described in this paper was studied for sorting arrays of integers that are already sorted, sorted in reverse order, and randomly generated. The experiment was conducted on a laptop with an Intel(R) Core (TM) i7-4500U a speed of 2.4 GHz, and 8GB of RAM. The sizes of the arrays ranged from 200,000 to 1000,000 elements. To study the behavior of the algorithms on arrays of random elements, each algorithm was used to sort three sequences of random numbers of a specific size, and the average running time are calculated.

Algorithm 4 SingletonQuickSort

```
procedure Singleton( A, l, r )
  if (l + 20 <= r)
      q = median3( A, l, r)
      i = l
      j = r-1
   while  i < j
          while (A[++i] < q)
          while (A[--j] > q)
          if (i < j)
              exchange i with  j
     exchange i with  r-1
     QuickSort ( A, l, i-1 )
     QuickSort ( A, i+1, r)
```

Table 1. Summarize some aspects of Quicksort and it's variants**.**

Variant	Difference from others	Effect	performanc e	Year
Hoare QuickSort	Pivot is selected randomly	Worst case rarely to happen	$O(n^2)$ occurred rarely	1961
Scowen QuickSort	Pivot is the middle of the array	When array is sorted or nearly sorted	$O(n \log_2 n)$	1965
Singleton QuickSort	Pivot is chosen using the median of	worst case is rarely to happen	$O(n^2)$ occurred rarely	1969

	three method			
Sedgewick QuickSort	Pivot is selected randomly	largest element is located in the first element in the array	$O(n^2)$ occurred not often	1978
Bentley QuickSort	Pivot is chosen by scanning array from left and right then swapping elements that are out of order	the number of swapping between array elements is reduced	Running time for reversed array is slower than sorted array	1984
Bsort	Pivot is the middle of the array	the largest element is located in the rightmost of the left subarray, and the smallest element is located in the leftmost of the right subarray	$O(n^2)$ occurred not often	1985
Qsorte	Pivot is the middle of the array and the left and right subarray are checked to be sorted or not.	the number of swapping between array elements is reduced	$O(n^2)$	1987
QuickSort-Rotate			$O(n^2)$	1991
Qsort7	Different ways according to the size of the array			1993
FlashSort	consists of three stages: classification, permutation, and straight insertion	To avoid the compariso ns	O(n) + an auxiliary memory space with length equal to the number of different keys	1997
SS06	Pivot is the first element of the array and it divide the temporary array into two sub arrays	Using extra memory space to reduce swapping	$O(n^2)$ for sorted and reversed array. $O(n \log_2 n)$ for random order.	2006
MQuickSort	Pivot is dynamic selection. It does not depend on the location in the array.	the height of the splitting tree is reduced	$\Omega(n)$ for sorted array. $O(n \log_2 n)$ in worst case	2012
K-sort				2023

Figure 1 below show the running times for the sorting algorithms when used to sort arrays of random numbers of different sizes. It is evident that (MQuickSort) is the fastest, giving the best performance for sorting arrays of random integers then qsort7, singleton and finally Hoare.

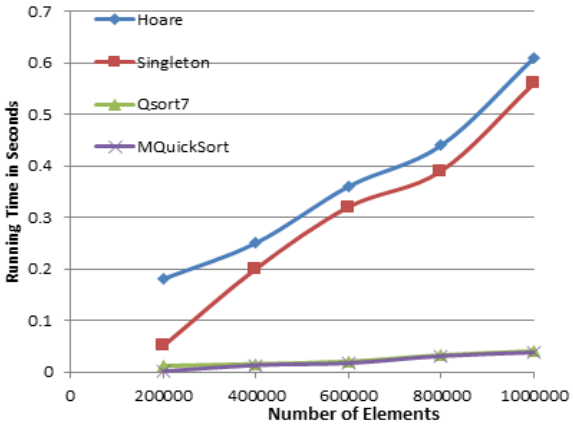

Fig1. Average Running Times for Random Data

Four algorithms are comparable for values of N ranging from 200,000 up to 1000,000. For values of N greater than 200,000, Hoare and singleton becomes slightly slower than Qsort7 and MquickSort. the running time of Hoare and Singleton increases the gap from Qsort7 and MquickSort when increasing n.

For data that is already sorted, Figure 2 gives the result such that MquickSort proved to be the fastest. qsort7 is faster than Singleton and Hoar's.

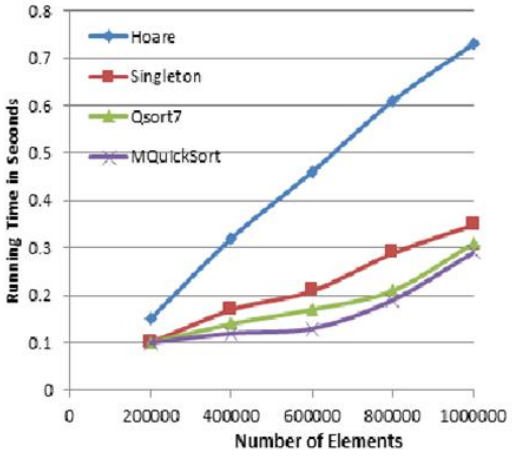

Fig2. Average Running Times for Sorted Data

For data sorted in reverse order, the fastest running times is achieved by MQuickSort.

The gap between running time of MQuickSort and Singleton is small, and the gap is widely increased with the Hoare for N >= 200,000 as seen in figure 3.

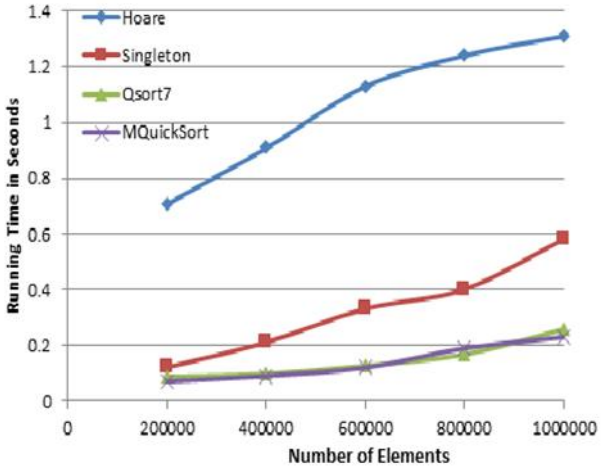

Fig3. Average Running Times for Data Sorted in Reverse.

6. Conclusion

This paper presented a comprehensive survey and comparison of the Quicksort algorithm and its variants from 1961 to 2023, incorporating the results of recent experimental studies. The experimental results demonstrate that the theoretical performance predictions align with the practical outcomes for several Quicksort variants. Additionally, the experiments confirmed that the algorithm's behavior varies based on the size of the elements being sorted and the method of pivot selection. The choice of pivot is critical to the efficiency of the Quicksort algorithm, underscoring that each variant has its own advantages and disadvantages. Each variant outperforms others under specific conditions, making the selection of the most efficient variant problem-specific. Thus, it is concluded that the Quicksort algorithm is tailored to the particularities of the sorting problem at hand.

Conflicts of Interest

Suggestion a good features for quicksort variant:

The MQuickSort is powerful algorithm when the dispersion values of data is small range. The worst case O(n2) may be happen when the dispersion of data values is wider range.

Author Contributions

Conceptualization, quicksort, sorting, and algorithm variants; methodology, Experimental Study; software, C++ ; validation, running time of each variant for input array sizes ranging.

Acknowledgments

Acknowledgments are to show that the article is supported by what organization. For example, “This work was supported by the National Nature Science Foundation under Grant No. 405”.

References

[1] Hoare, C., (1961). "Algorithm 64: Quicksort". *Comm. ACM* 4, 7 ,321.

[2] Thoma, C., Charles, L., Ronald, L., and Clifford, S., (2009). "Introduction to Algorithms. Third Edition". *The MIT Press*. Cambridge, Massachusetts London, England.

[3] Martnez, C., and Roura, S., (2002)."Optimal sampling strategies in Quicksort and Quickselect". *SIAM Journal on Computing*, 31(3).

[4] Abdel, D., Thaer K, Azzam S., Manuel, S., Alfonso, O., (2012). "Enhancing QuickSort Algorithm using a Dynamic Pivot Selection Technique". *Wulfenia Journal Klagenfurt, Austria. Vol 19, No. 10.*

[5] Sedgewick, R, (1978)." Implementing Quicksort programs". *Communications of the ACM*, 847 (2).

[6] Loeser, R., (1976). "Survey on Algorithms 347, 426, and Quicksort". *ACM Transactions on Mathematical Software (TOMS)*, Vol. 2 No. 3.

[7] Mishra, A., and Garg, D., (2008). "Selection of Best Sorting Algorithm". *International Journal of Intelligent Processing*, 2(2), pp.363–368.

[8] Ishwari, R., Bhawnesh, K., and Tinku, S.,(2012). Performance Comparison of Sequential Quick Sort and Parallel Quick Sort Algorithms. *International Journal of Computer Applications*, Vol.57, No.9, pp.14-22.

[9] R.S. Scowen, (1965)."Algorithm 271: Quickersort", Comm. ACM 8,11, pp. 669-670.

[10] R. C. Singleton, (1969). "Algorithm 347: An efficient algorithm for sorting with minimal storage", *Comm. ACM* 12, 3, pp. 186-187.

[11] J. Bentley, (1984). "Programming Pearl: How to sort", *Com.. ACM,*Vol. 27 Issue 4.

[12] B. McDaniel, (1991). "Variations on Put First", *Conference on Applied Mathematics* , University of Central Oklahoma.

[13] R. L. Wainwright, (1985). "A class of sorting algorithms based on Quicksort", *Comm. ACM,* Vol. 28 Number 4.

[14] R. L Wainright, (1987). "Quicksort algorithms with an early exit for sorted subfiles" ,*Comm. ACM*.

[15] B. McDaniel, (1991). "Variations on Put First", *Conference on Applied Mathematics* , University of Central Oklahoma.

[16] J. L. BENTLEY, M. D. McILROY, (1993). "Engineering a Sort Function". *Software—Practice and Experience,*Vol. 23(11), pp. 249 – 1265.

[17] K.D. Neubert, (1997)."The FlashSort algorithm," *In Proc. of the euroFORTH'97 –Conf.*, Oxford, England, pp. 26 – 28,.

[18] K. K. Sundararajan, and S. Chakraborty, (2006)." A new sorting algorithm", *InterStat, Statistics on the Internet*.

[19] Sundararajan, KK, Pal, M, Chakarborty ,S., & Mahanti, NC, (2013) "K-Sort: A New Sorting Algorithm that beats Heap Sort for n ≤ 70 Lakhs!. *Int. J. on Recent Trends in Engineering and Technology-IJRTET*.

[20] Aumüller, M., Dietzfelbinger, M., & Klaue, B. (2016). "Practical Quicksort variants with Yaroslavskiy's dual-pivot algorithm". *ACM Journal of Experimental Algorithmics*, 21.

[21] Edelkamp, S., & Weiss, A. (2022). "Recent Advances in Quicksort". *ACM Computing Surveys*, 54(7), Article 157.

[22] Rahman, S., & Munro, J.I. (2023). "Enhanced Quicksort Techniques for Large Data Sets". *Journal of Computer Science and Technology*, 38(2), pp. 349-361.