Journal of Natural Science and Textile Technology
Volume 2, Issue 1 (2023)
ISSN: 2789-9411

Journal homepage: textile.nub.ac.bd/page/53/Journal

A Study of Pivot Positioning Methods for Quicksort Algorithm

Towhidul Islam[1,*], Zahin Mostakim[1] and Md. Sanaullah Murad[1]

[1] *Department of Textile Engineering, Northern University Bangladesh, Dhaka 1230, Bangladesh*
**Corresponding Author's E-mail: towhidulislam133086@gmail.com*
DOI: 10.5281/zenodo.8255601

KEYWORDS

Quicksort
Pivot
Algorithm
Random
Dual pivot

ABSTRACT

Traditionally deterministic algorithm for quicksort is used which gives O (n log n) running time for an average case and O (n2) for the worst case, where n is the number of elements to be sorted. If a randomized approach is used the worst-case running time is reduced to O (n log n). The median of three approaches for choosing pivot and multi-pivot approach like dual pivot also improves the running time. In this paper, we analyze the effectiveness of various pivot positioning methods. To achieve this, we take randomized generated and shorted array of different sizes which provide us with enough data to conclude.

1. INTRODUCTION

Any technique of placing objects deliberately is known as sorting. It is usually done in numerical order or lexicographical order. Among different algorithms, quicksort is very popular in computer science. It is very fast and requires very less additional space which is based on the principle of the Divide and conquer algorithm [1]. Divide and conquer is an algorithmic strategy based on multi-brunched recursion. It functions by repeatedly splitting a problem into two or more sub-problems of the same or similar kind until they are straightforward enough to be solved directly. After then, the sub-problems' solutions are integrated to formulate a solution to the main problem [1].

The classical quicksort algorithm in [2] was developed by Tony Hoare, which gives on average $O\ (n \log n)$ comparisons to sort n data by partitioning. Even though, it is rare, in the worst-case scenario it gives $O\ (n^2)$ comparison. Here, we take a pivot and place all items smaller than the pivot in a position on its left while placing all other items in a position on its right. Quicksort is frequently the most realistic choice for sorting since it is highly effective on average, despite its sluggish worst-case running time [1].

We can counter the worst-case situation by random sampling and get a randomized version of quicksort. Despite choosing a pivot from one side, here, we randomly choose a pivot and do the partitioning process. For this approach, the impact of the worst-case lessens and the average runtime remains $O\ (n \log n)$ [3]. Another way to

counter the worst-case scenario is to use the median of three approaches where the pivot is chosen more carefully than picking a random element from the subarray. It is recommended by Robert Sedgewick [4]. We determine the pivot as the median of a set of three elements randomly selected from the subarray.

Multi-Pivot Quicksort outlines quicksort variations in which the input is divided into k + 1 segments during the splitting stage using k pivots. With a small constant number of pivots, we can increase the performance of classical quicksort. In this paper, we have only discussed the dual pivot approach [5] of the multi-pivot quicksort. The concept behind the dual pivot is that we may pick two items at random and utilize them as pivots for dividing the array. Dual pivot quicksort does not seem to be quicker than a conventional approach in terms of mean values. But in practical cases, it is way more efficient [6].

2. MATERIALS AND METHODS

The rest of this paper is organized as follows: in section II, we describe classical quicksort, randomized quicksort, a median of three approaches, and dual-pivot quicksort. Experimental results are reported in the following section. Finally, in section IV, we present our conclusions from these results.

2.1 Quicksort

Quicksort follows the following rules:
• Choose a pivot element. The final element in the sorting box is what we utilize.
• Place any numbers smaller than the pivot to a place on its left, and all other numbers to a position on its right, as you go through the sorting area. This is done by swapping elements.
• Assuming that the pivot is now in its sorted location, we proceed with the divide-and-conquer method by using the same algorithm on both the part to the left and the part to the right of the pivot.

Fig. 1 shows a visualization of the result of the partitioning method using a single pivot. The elements whose values are less than the pivot element are placed on the left side of the pivot and the remaining are on the right side. Now the pivot is in its optimal position in the sorted array. Further, we must apply this partitioning method both at the left and the right side of the pivot recursively.

Fig. 1: The position of elements regarding pivot after applying the partition scheme.

2.2 Basic Quicksort

The quicksort's execution time is determined by whether the partitioning is balanced or unbalanced. Here balanced partitioning means the sub-problems derived by the partitioning scheme are relatively the same size. If the difference between the sizes of the sub-array is very high, then it is called unbalanced partitioning. If the partitioning is balanced, the algorithm will be more efficient. But this phenomenon depends on which elements are used as pivots for partitioning. It will exhibit the best case. If the elements to be sorted are already sorted or reversely sorted, then the worst case will happen. When the algorithm provides the most unbalanced partitioning possible at every recursive level, then there is one subproblem with $n - 1$ element and the other with 0 elements. Thus, the time complexity becomes $O(n^2)$ [1].

In the case of the best case, the problem is split in the most evenly possible way. A problem becomes two subproblems of the almost same size. If the size of the array to be sorted is n, then for the best case, the size of the subproblem is no more than $n/2$. Intuitively, we can see that the time complexity for the best case is $O(n \log n)$.

2.3 Randomized Quicksort

In the randomized version of quicksort, we modify the partition procedure to avoid the worst case. At each step of the algorithm, we choose the pivot randomly and then we exchange an element with the pivot chosen from

the given array. Then we conventionally do the partitioning and recursively sort the given array [3].

In a randomized approach, no input can elicit worst-case behavior. It is only possible if we get "unlucky" numbers from the random number generator. Here by the term "unlucky", we are stating the fact that there is a possibility of getting the pivot elements in sorted order. But the possibility is very low if using a good randomization process. Randomization cannot eliminate the worst case, but it can make it less likely.

The performance of randomized quicksort can be improved in various ways. If we pick a small number of elements randomly and use the median of these elements as the partition element, then we can further improve the randomization. These randomly chosen elements are from a random sample of the elements and it is expected to be an approximate median of the whole problem set. Thus, the problem set will be partitioned evenly [3].

2.4 Median of Three Approaches

The median of the three approaches is a brilliant way to avoid the worst case. It can also be used to improve the randomized algorithm. In this method, instead of choosing a pivot from a side of the given array, we take the median of the first, middle, and last elements of the array as a pivot. Then we regularly do the partitioning using that pivot. [4] We can further improve the process by sorting those three items, not just using the median as the pivot. This scheme ensures that in the case of sorted data, the process will remain optimal. In that case, the median will be the middle element of the array and the problem will be partitioned into two subproblems of the same size.

The most balanced split will result from using the overall median as a pivot, and the best run time will follow. However, selecting the real median takes time. To get a better approximate median, we take a sample of three elements. It is still possible to get the worst case in this process, but it is more difficult to manipulate the data into giving the worst case.

2.5 Dual Pivot Quicksort

In 2009, Yaroslavskiy [5] has given the following more effective divide-and-conquer procedure for quicksort using two pivots instead of one:

- Choose two pivot elements p1 and p2 where p1 < p2.
- Then use partition algorithms and distribute the elements into three parts, where the values less than p1 will be in the first part, values between p1 and p2 will be in the second part, and values greater than p2 will be in the third part.
- These steps are repeated recursively for each of the three parts.

Fig. 2 shows us the state of the array after applying a dual pivot scheme where the smaller pivot P1 and the larger pivot P2 are respectively in their optimal position. The elements whose values are less than P1 are in the first part which is at the left of P1. Values between P1 and P2 are in the second part and values greater than P2 are in the third part which is at the right of P2.

<P1	P1	P1<=&<=P2	P2	P2<

Fig. 2: The position of elements regarding pivot P1 and P2 after applying a dual pivot scheme where P1 < P2.

Although the dual pivot scheme does more comparisons than basic quicksort, this procedure is more efficient than the classical approach for sorting larger-sized primitive unsorted arrays and can be easily adjusted for other numeric strings and comparable types. This is also effective for sorted arrays or arrays with repeated elements [5]. We can further apply special choice procedures for pivot elements to improve this scheme.

3. RESULTS AND DISCUSSIONS

We implemented the discussed algorithms in Java programming language using Netbeans 8.1. We used HP Pavilion 14-e036tx notebook pc with Intel(R) Core (TM) i7-4702MQ @ 2.2GHz and 4 GB 1600 MHz DDR3 RAM.

3.1 Average Case Analysis

Table 1 shows us the runtime of different variants of quicksort for random input

which resembles the average case runtime. By taking random input, we lessened the possibility of the worst case. Here the column size means the number of elements in the given array. Here the columns Basic, Randomized, Median-of-3, and Dual refer to the running time of classical, randomized, median of three, and dual-pivot approaches of quicksort which are measured in nanoseconds.

Table 1. Average case running time of different schemes of quicksort in nanoseconds for different input sizes

Size	Basic	Randomized	Median-of-3	Dual
10^3	4.4^5	2.3^5	2.7^4	7.6^4
10^4	1.7^6	6.6^5	2.2^5	7.4^5
10^5	1.1^7	5.4^6	1.8^6	4.4^6
10^6	3.2^7	2.0^7	4.3^6	1.1^7
10^7	2.7^8	1.1^8	1.2^7	2.1^7
10^8	2.9^9	1.9^9	1.0^8	2.1^8

Table 1 shows us the runtime of different variants of quicksort for random input which resembles the average case runtime. By taking random input, we lessened the possibility of the worst case. From the table, we can deduce that all the other three variants gave better running time than the classical approach. Randomized, median of three, and dual-pivot schemes improved the performance as expected. But here randomized approach did not perform as well as the other two variants except for the basic scheme. The lack of modification of the randomized algorithm was responsible for it. We can overcome this problem by modifying the randomized algorithm and making it more desirable.

3.2 Worst Case Analysis

In table 2 there is experimental data for a worst-case running time in nanoseconds. Here the column size means the number of elements in the given array. The columns Basic, Randomized, Median-of-3, and Dual refer to the running time of classical, randomized, median of three, and dual-pivot approaches of quicksort. The input data was taken in a way such that it was already sorted. In this way, we ensured the occurrence of the worst case.

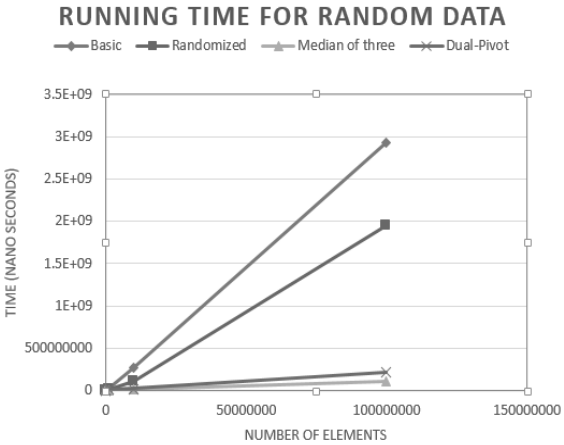

Fig. 3: The position of elements regarding pivot after applying the partition scheme

Table 2. The worst-case running time of different schemes of quicksort in nanoseconds for sorted data

Size	Basic	Randomized	Median-of-3	Dual
10^3	5.4^5	2.2^5	1.9^5	4.8^5
10^4	2.3^6	1.4^5	6.4^6	2.4^6
10^5	1.99^7	8.2^5	4.5^6	2.1^7
10^6	1.3^8	1.05^8	1.4^7	4.8^7
10^7	1.1^9	1.02^9	8.6^7	3.6^8
10^8	1.2^{10}	1.06^{10}	1.3^9	5.3^9

Fig. 4: The position of elements regarding pivot after applying the partition scheme

In **Fig. 4** we can see that all the other three variants work better than the classical scheme regardless of size. From table 2 we can further deduce that for a small-sized array randomized algorithm works better than the median of three and the dual pivot approach. This happens because the other two schemes need more comparisons than the randomized algorithm. Though dual pivot quicksort is not theoretically as sound as the classical approach, this procedure, and the median of three schemes drastically improve the running time for larger-sized arrays. So, these two approaches are preferable for practical use.

4. CONCLUSION

In this paper, we investigated the impact of different pivot positioning methods on quicksort. We found that the median of the three approaches gave the overall best running time. But the dual pivot scheme is the best approach for larger-sized problems in practice. If we modify the randomized and dual pivot approach according to the median of the three procedures, we can hope to achieve a better result in the future.

REFERENCES

[1] [Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L., and Stein, Clifford Introduction to Algorithms, Third Edition. The MIT Press, 2009.
[2] Hoare, C. A. R. "Algorithm 64: Quicksort'' Commun. ACM, vol. 4, no. 7, pp. 321-327, July 1961.
[3] Horowitz, Ellis. and Rajasekaran, Sanguthevar. and Sahni, Sartaj Fundamentals of Computer Algorithms, Second Edition. Universities Press/Orient Blackswan, 2008.
[4] Sedgewick, Robert "Implementing Quicksort Programs'' Commun. ACM, vol. 21, no. 10, pp. 847–857, Oct. 1978.
[5] Yaroslavskiy, Vladimir "Dual-pivot quicksort'', Research Disclosure, 2009.
[6] Wild, Sebastian "Why Is Dual-Pivot Quicksort Fast?'' arXiv preprint arXiv:1511.01138, 2015