2024 4th Int. Conf. on Innovations in Science, Engineering and Technology (ICISET)
26-27 October 2024, Chittagong, Bangladesh

An Experimental Analysis on Different Pivot Selection Approaches for the Quicksort Algorithm

Jeba Tahsin
Department of Computer Science
American International University -Bangladesh
Dhaka, Bangladesh
ai.lily775@gmail.com

Dipta Gomes
Department of Computer Science
American International University -Bangladesh
Dhaka, Bangladesh
diptagomes@aiub.edu

Md. Manzurul Hasan
Department of Computer Science
American International University -Bangladesh
Dhaka, Bangladesh
manzurul@aiub.edu

2024 International Conference on Innovations in Science, Engineering and Technology (ICISET) | 979-8-3503-5549-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICISET62123.2024.10939753

***Abstract*—The research primarily examines the significance of pivot selection of the widely used QuickSort algorithm in order to increase the overall performance and efficiency. Quicksort has an average time complexity of $O(nlogn)$, but its performance can degrade to $O(n^2)$ in the worst-case scenario, which occurs when a pivot element is chosen badly. This study focuses on the influence of different pivot selection techniques on the efficiency of the Quicksort algorithm through empirical evaluation. To determine which strategy works best for an individual data set and array size, different methods have been evaluated, aiming to choose a pivot that is in close proximity to the median of the sub-array, evaluating their efficiency and any drawbacks. In terms of efficiency, the Median of Seven (MO7) and Median of Three (MO3) exhibits the best results, where MO7 gives an execution time of 0.0112s and MOT of 0.0124s. A comparative decision criteria has also been proposed in this research in choosing the optimum approach among the best performing MOT and MO7, where MOT being simpler and MO7 being more efficient. These insights offer practical guidance for optimizing Quick Sort implementations in real-world scenarios, where its performance is paramount.**

***Index Terms*—Quicksort, Pivot selection, Sorting algorithms, Experimental analysis, Median of Three (MOT), Median of Five (MO5), Median of Seven (MO7), Median of Nine (MO9), Random selection, Time complexity; efficiency; worst-case scenario, Quicksort variations.**

I. Introduction

Sorting algorithms are fundamental to various data processing tasks which plays an important role in organizing vast datasets efficiently. Quicksort, renowned for its high efficiency and speed, employs a divide-and-conquer approach . The algorithm recursively divides an array based on a selected pivot element, arranging elements smaller than the pivot to its left and bigger elements to its right. Optimally, the pivot should partition the array into about equal sub-arrays, minimizing comparisons and achieving efficient sorting. Nevertheless, the efficiency of Quicksort depends on the quality of the pivot selection [1]. Choosing a pivot positioned at one of the outermost ends of the sub-array, such as the first or final element, might result in imbalanced partitions [2]. This can greatly escalate the number of comparisons needed and lead to a complexity of $O(n^2)$.

In this research, we investigate the efficacy of different pivot selection strategies in enhancing Quicksort algorithm performance. Through experimental analysis and statistical evaluation, we aim to identify the most efficient pivot selection approach for optimizing the runtime. By shedding light on the strengths and limitations of each technique, this research seeks to contribute valuable insights to the field of sorting algorithms and algorithmic optimization.

In the following sections, we discuss the background of the Quicksort algorithm, explore various pivot selection strategies, outline the experimental design and methodology employed, present the obtained data, analyze the results, and discuss avenues for further research and exploration.

II. QUICKSORT ALGORITHM: LITERATURE AND BACKGROUND

A. Previous Studies

The book titled *Quicksort and Sorting Lower Bounds* by Blelloch [1] offers an in-depth exploration of Quicksort and theoretical lower bounds for sorting algorithms, providing valuable insights into algorithm design and optimization. Similarly, Sedgewick et al. [3] present a comprehensive analysis of sorting algorithms, including Quicksort and Mergesort, with detailed explanations, pseudo-code, and performance analysis, making it a key resource for learners at all levels. Cormen et al. [4], in their seminal work, provide rigorous theoretical and practical insights into sorting algorithms, including Quicksort, establishing it as a foundational reference in the field. Blelloch et al. [2] further discuss lower bounds for sorting algorithms, highlighting theoretical limitations and challenges in achieving optimal worst-case complexity.

Recent research has focused on improving Quicksort's efficiency. Sabir et al. [5] proposes a variant that reduces worst-case time complexity to $O(nlogn)$ by minimizing comparisons through a manual sort for small inputs and using the mean as the pivot. Aoun Aftab et al. [6] introduce a dynamic pivot selection technique, the SMS-Algorithm, which organizes input into provisional arrays $(PosArray, NegArray, andFreqArray)$ and achieves a complexity of $O(n + f(max + min))$ for distinct elements. Aumüller et al. [7] propose a Dual-pivot Quicksort algorithm, subdividing input into three parts and achieving a complexity of $1.8nlnn + O(n)$ by reducing key comparisons.

Smith et al. [8] explore machine learning-based pivot selection strategies, demonstrating potential for enhancing sorting efficiency. Kumar and Chakraborty [9] experimentally analyze pivot selection's impact on worst-case complexity, offering practical insights into optimizing Quicksort.

B. Psudeocode

A simplified representation of the Quicksort algorithm has been presented in Algorithm 1.

C. The Importance of Pivot Selection

The efficiency of Quicksort is significantly impacted by the selected pivot element. Optimally, the pivot should be positioned in close proximity to the median of the sub-array, leading to partitions that are about the same size. Choosing a pivot that is biased towards one end of the sub-array results in uneven partitions, which increases the number of comparisons and might potentially reduce the effectiveness of the method.

Algorithm 1 Quicksort

```
0: procedure QUICKSORT($arr, low, high$)
0:   if $low < high$ then
0:     $pivot\_index \leftarrow$ PARTITION($arr, low, high$)
0:     QUICKSORT(arr, low, pivot_index - 1)
0:     QUICKSORT(arr, pivot_index + 1, high)
0:   end if
0: end procedure

0: procedure PARTITION($arr, low, high$)
0:   $pivot \leftarrow$ SELECT_PIVOT($arr, low, high$)
0:   $i \leftarrow low - 1$
0:   for $j \leftarrow low$ to $high - 1$ do
0:     if $arr[j] \leq pivot$ then
0:       $i \leftarrow i + 1$
0:       swap($arr[i]$, $arr[j]$)
0:     end if
0:   end for
0:   swap($arr[i+1]$, $pivot$)
0:   return $i + 1$
0: end procedure=0
```

III. PIVOT SELECTION STRATEGIES

The following pivot selection strategies will be discussed in this section, all aiming to choose a value near the median of the sub-array:

A. Median of Three (MOT):

The median value from the first, middle, and last element of the sub-array is selected in Median of Three (MOT). So, the elements are,

L = Low
H = High
$\frac{L+H}{2}$ = Average of Low and High

B. Median of Five (MO5):

This selects five elements randomly from the sub-array and chooses the median of those five values. So, the elements are,

L = Low
H = High
$\frac{L+H}{2}$ = Average of Low and High
+ 2 Random Elements
Random Element 1 = *Value 1*
Random Element 2 = *Value 2*

MO5 without Random Selection: This divides the sub array into four equal parts and selects the median from those five elements. So, the elements are,

$$L, H, \frac{(L+H)}{2}, \frac{(L+H)}{4}, 3\left(\frac{(L+H)}{4}\right)$$
where, $L = Low, H = High$

C. Median of Seven (MO7):

MO7 with Random Selection: Similar to MO5 with random selection but uses seven elements. So, the elements are,

$$L, H, \frac{(L+H)}{2}, \frac{(L+H)}{4}, 3\left(\frac{(L+H)}{4}\right) + 2$$
where, $L = \text{Low}, H = \text{High}$,
and Random Element

MO7 without Random Selection: Similar to MO5 without random selection but uses seven elements. So, the elements are,

$$L, H, \frac{(L+H)}{3}, \frac{(L+H)}{6}, 2\left(\frac{(L+H)}{3}\right), 5\left(\frac{(L+H)}{6}\right)$$
where, $L = \text{Low}, H = \text{High}$

D. Median of Nine (MO9):

MO9 with Random Selection: Similar to MO5 with random selection but uses nine elements. So the elements are,

$$L, H, \frac{(L+H)}{2}, \frac{(L+H)}{3}, \frac{(L+H)}{6}, 2\left(\frac{(L+H)}{3}\right), 5\left(\frac{(L+H)}{6}\right) + 2$$
where, $L = \text{Low}, H = \text{High}$
Random Elements

MO9 without Random Selection: Similar to MO5 without random selection but uses nine elements. So the elements are,

$$L, H, \frac{(L+H)}{2}, \frac{(L+H)}{4}, \frac{(L+H)}{8}, 3\left(\frac{(L+H)}{8}\right), 5\left(\frac{(L+H)}{8}\right)$$
and $5\left(\frac{(L+H)}{8}\right)$
where, $L = \text{Low}, H = \text{High}$

IV. EXPERIMENT DESIGN AND METHODOLOGY

A. Experimental Setup

The experiment is carried out in a Windows laptop of model HP Omen 15-dcixxx with 16 GB of RAM with NVIDIA GeForce GTX 1650 with 4 GB DDR5 memory in C environment.

B. Data Generation:

The experiment utilizes randomly generated data to provide a baseline for comparison. Here's the approach:

- Generate 100 random integer arrays of a specific size (e.g., 100 elements) using Random class.
- Each element in the array will have a random value within a predefined range (e.g., between 0 and 999).

C. Main Function:

- Iterates through each array and applies all seven Quicksort variations with different pivot selection strategies.
- Measures execution time for each strategy using Stopwatch and stores the results in a double array.
- Writes results to a text file using StreamWriter, including:
  - Generated array elements.
  - Execution time for each Quicksort variation applied to the array.
  - Average execution time across all arrays for each strategy.

D. Helper Functions:

Median Calculation Functions:

- **MedianOfThree:** Calculates the median from three elements after ensuring proper ordering.
- **MedianOfFive:** Calculates the median by finding the median of the medians from four equal sub arrays.
- **MedianOfSeven:** Similar to MedianOfFive but uses three sub-arrays for each half.
- **MedianOfNine:** Similar to MedianOfFive but uses six sub-arrays for each half.

Partition Function (Partition): Rearranges the array elements around the chosen pivot, placing smaller elements to the left and larger elements to the right.
Swap Function (Swap): Swaps the values of two integer variables.
Utility Functions:

- **PrintTechniqueName:** Writes the name of the pivot selection strategy to the output file.
- **PrintArray:** Prints the elements of an integer array to the output file.

This code effectively implements the experiment design, allowing for measurement and comparison of execution times across different pivot selection strategies and data sets

TABLE I
EXECUTION TIMES FOR DIFFERENT MEDIAN ALGORITHMS (MOT, M5, M7) (FIRST 50 ROWS)

Array	**Median of Three**	**Median of Five**	**Median of Seven**
1	0.4879	0.455	0.3301
2	0.0132	0.0128	0.0126
3	0.0116	0.012	0.0114
4	0.0173	0.0183	0.0118
5	0.0172	0.0152	0.0444
6	0.0145	0.0172	0.0157
7	0.0154	0.0169	0.0174
8	0.0197	0.0188	0.0197
9	0.0172	0.0191	0.0117
10	0.0117	0.0183	0.0114
11	0.0156	0.0203	0.012
12	0.0117	0.0183	0.0114
13	0.0116	0.0186	0.012
14	0.013	0.0186	0.0116
15	0.0117	0.0186	0.0118
16	0.0118	0.0187	0.0116
17	0.0117	0.0187	0.0115
18	0.0116	0.0186	0.0115
19	0.0116	0.0183	0.0115
20	0.0116	0.0187	0.0116
21	0.0225	0.0233	0.0191
22	0.0221	0.0249	0.0159
23	0.0154	0.0153	0.0161
24	0.0152	0.0148	0.0147
25	0.0146	0.0151	0.0157
26	0.02853	0.0164	0.0158
27	0.0159	0.0161	0.0176
28	0.0159	0.0161	0.0179
29	0.0143	0.0154	0.0154
30	0.0154	0.0153	0.0163
31	0.0152	0.0157	0.0153
32	0.0151	0.0156	0.0159
33	0.0155	0.0159	0.0158
34	0.0153	0.0157	0.0159
35	0.0147	0.0156	0.0156
36	0.0156	0.0157	0.0157
37	0.0149	0.0159	0.0157
38	0.0152	0.0156	0.0158
39	0.0151	0.0159	0.0156
40	0.015	0.0156	0.0158
41	0.000015	0.0000324	0.0000342
42	0.0000189	0.0000192	0.0000181
43	0.0000146	0.0000161	0.0000164
44	0.0000152	0.0000182	0.0000174
45	0.0000146	0.0000156	0.0000172
46	0.0000157	0.000018	0.0000448
47	0.0000158	0.0000187	0.0000155
48	0.000015	0.0000188	0.0000155
49	0.0000154	0.0000162	0.0000205
50	0.0000158	0.0000165	0.0000206
Average Execution Time	0.012447594	0.013926294	0.011229894

E. Execution Time Measurement

For each array, the experiment measures and records the execution time of Quicksort using each strategy. The execution times for the methods has been represented in table 1 and table 2.

- Utilize a Stopwatch class or similar functionality to measure the elapsed time during Quicksort execution with

TABLE II
EXECUTION TIMES FOR DIFFERENT MEDIAN ALGORITHMS (MOT, M5, M7) (LAST 50 ROWS)

Array	Median of Three	Median of Five	Median of Seven
51	0.0000146	0.0000196	0.0000155
52	0.0000143	0.000017	0.000017
53	0.0000149	0.000016	0.000016
54	0.0000154	0.0000173	0.000016
55	0.0000149	0.000018	0.0000159
56	0.0000147	0.0000164	0.000016
57	0.0000145	0.0000173	0.0000171
58	0.0000151	0.0000165	0.0000164
59	0.0000148	0.0000165	0.0000174
60	0.0000147	0.000018	0.0000162
61	0.0000164	0.0001034	0.0000173
62	0.0000198	0.0000243	0.0000161
63	0.0000174	0.0000202	0.0000214
64	0.0000209	0.0000181	0.0000142
65	0.0000191	0.0000173	0.0000166
66	0.0000359	0.0000172	0.0000162
67	0.0000146	0.0000182	0.0000372
68	0.000015	0.0000152	0.0000118
69	0.0000173	0.0000193	0.0000143
70	0.0000196	0.0000148	0.0000118
71	0.0000109	0.0000122	0.0000116
72	0.0000122	0.0000123	0.0000147
73	0.0000142	0.0000156	0.0000147
74	0.0000169	0.0000176	0.000024
75	0.0000168	0.000015	0.000017
76	0.0000149	0.0000152	0.0000204
77	0.0000129	0.0000153	0.0000153
78	0.0000142	0.0000154	0.0000144
79	0.0000145	0.0000226	0.0000162
80	0.000015	0.0000156	0.000016
81	0.0000163	0.0000223	0.0000139
82	0.0000141	0.0000177	0.0000152
83	0.0000152	0.0000162	0.0000163
84	0.0000139	0.0001641	0.0000217
85	0.0000154	0.0000161	0.0000156
86	0.0000132	0.0000377	0.0000242
87	0.000023	0.0000235	0.0000156
88	0.0000527	0.0000194	0.0000168
89	0.0000122	0.0000123	0.0000126
90	0.000011	0.000013	0.0000126
91	0.0138	0.015	0.0441
92	0.015	0.0214	0.025
93	0.0153	0.0168	0.0182
94	0.0159	0.0189	0.0163
95	0.022	0.0158	0.016
96	0.0152	0.0159	0.0152
97	0.0186	0.0151	0.0155
98	0.0156	0.0161	0.0191
99	0.016	0.0169	0.0163
100	0.0175	0.1187	0.0121
Average Execution Time	0.012447594	0.013926294	0.011229894

each pivot selection strategy.

- Store the measured execution times for each array and strategy in a data structure (e.g., an array) for further analysis.

F. Statistical Analysis

The raw execution time data offers valuable insights; however, statistical analysis enables more robust conclusions. First, the average execution time for each pivot selection strategy is calculated across all arrays, providing a high-level performance overview. Next, the distribution of execution times is analyzed, often visualized through histograms or similar graphical representations. A tightly clustered distribution around the mean indicates consistent performance, while a wider spread suggests variability influenced by array characteristics. Finally, statistical significance testing, such as ANOVA, is employed to determine whether observed differences in average execution times between strategies are statistically significant, ensuring that variations are not attributable to random chance.

G. Data Table

This research investigates the following sets of data output in Table 1 and Table 2 from the experimental implementation.

V. EXPERIMENTAL RESULTS AND DISCUSSION

To elaborate on the complexity analysis of the pivot selection strategies in Quicksort, we can examine both the theoretical time complexity and the practical implications of different pivot selection approaches (Median of 3, 5, 7, and 9).

A. Complexity analysis of Quick sort

Quicksort is a comparison-based sorting algorithm with a recursive structure. Its time complexity is influenced by how well the pivot divides the array.

- **Best Case:** The best-case scenario occurs when the pivot always divides the array into two nearly equal halves. This results in the recurrence relation:
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$
This recurrence relation yields a time complexity of $O(nlogn)$.
- **Worst Case:** The worst-case scenario occurs when the pivot consistently divides the array into one element and the rest, resulting in a recurrence relation:
$$T(n) = T(n-1) + O(n)$$
This leads to a time complexity of $O(n^2)$, which is highly inefficient for large datasets.
- **Average Case:** On average, Quicksort performs well with a time complexity of $O(nlogn)$due to the random nature of input and a good choice of pivot that usually results in balanced partitions.

B. Complexity Analysis of Pivot Selection Strategies

The choice of pivot selection strategy in Quicksort significantly impacts its performance. This analysis evaluates the time complexity of four strategies: Median of Three (MO3), Median of Five (MO5), Median of Seven (MO7), and Median of Nine (MO9).

- **Median of Three (MO3):** MO3 selects the pivot as the median of the first, middle, and last elements of the array. Finding the median requires $O(1)$ time due to a constant number of comparisons. MO3 avoids worst-case behavior in sorted or nearly sorted arrays, yielding an expected time complexity of $O(nlogn)$.

- **Median of Five (MO5):** MO5 selects the median of five elements, either randomly or from fixed positions. Sorting five elements takes $O(1)$ time, and MO5 improves partition balance, maintaining an expected time complexity of $O(nlogn)$. However, it incurs slightly higher overhead than MO3.
- **Median of Seven (MO7):** MO7 extends MO5 by selecting seven elements. Sorting seven elements also takes $O(1)$ time, with a higher likelihood of selecting a pivot close to the true median. The expected time complexity remains $O(nlogn)$, but with increased constant overhead.
- **Median of Nine (MO9):** MO9 selects nine elements, further improving pivot selection quality. Sorting nine elements remains $O(1)$, but with additional comparisons. MO9 offers the most robustness against worst-case scenarios, maintaining $O(nlogn)$ time complexity but with higher constant overhead.
- **Randomized Selection:** Randomized versions of MO5, MO7, and MO9 introduce negligible overhead $O(1)$ and reduce the likelihood of poor pivot choices, especially in sorted arrays. The expected time complexity remains $O(nlogn)$, with improved practical performance due to reduced worst-case probability.

While Quicksort can degrade to $O(n^2)$ in the worst case, advanced pivot selection strategies like MO3, MO5, MO7, and MO9, particularly when combined with randomization, significantly mitigate this risk, ensuring near-optimal performance in most scenarios.

TABLE III
COMPARISON OF DIFFERENT MEDIAN CALCULATION TECHNIQUES

Technique	Avg. Time	Performance	Explanation
MOT	Lowest	Fastest	Simplest approach, avoids overhead of random selection.
MO5 (no rand)	Faster	Very good	Good balance between median calculation cost and pivot selection effectiveness.
MO7 (no rand)	Close Second	Very good	Similar to MO5 (no random), might be slightly better pivot selection in some cases.
MO9 (no rand)	Slower	Slower	More overhead compared to MOT, MO5, and MO7 (uses 3, 5, or 7 elements), avoids overhead of random selection in MO5/MO7 with random.
MO5 (rand)	Slower	Average	Random selection adds overhead, benefit might be outweighed by extra comparisons.
MO7 (rand)	Slower	Average	Similar to MO5 with random selection.
MO9 (rand)	Highest	Slowest	Most overhead due to random selection and potentially more comparisons for median calculation.

C. Practical Implications

Based on the provided output, we can observe the following regarding the performance of different pivot selection strategies in Quicksort in Table 5. The terms ”Fastest” and ”Slowest” were determined based on the statistical analysis of execution times. Specifically, a strategy was deemed ”Fastest” if it consistently had the lowest mean execution time across multiple runs, while ”Slowest” was used for strategies with significantly higher mean times.

1) Fastest Execution Time Techniques: **Median of Three (MOT):** Across all arrays, MOT has the lowest average execution time.

Median of Five without Random Selection (MO5): This strategy also performs well with an average execution time.

Median of Seven without Random Selection (MO7): This technique comes in a close second with an average execution time.

2) Slowest Execution Time Techniques: **Median of Five with Random Selection (MO5 Random):** This technique has the highest average execution time.

Median of Nine with Random Selection (MO9 Random): This strategy is also relatively slow with an average execution time.

3) Observation and Discussion: : **Median of Three (MOT) and Median of Five/Seven without Random Selection:** These techniques involve calculating the median from a small number of elements (3 or 5/7) and avoid the overhead of random selection. This reduces the number of comparisons needed to find a reasonable pivot, leading to faster execution times.

Median of Five/Seven with Random Selection: Introducing random selection adds some overhead compared to deterministic selection (MOT, MO5, MO7). While it might improve pivot selection in certain cases, the additional comparisons can outweigh the benefit.

Median of Nine with Random Selection (MO9 Random): Similar to MO5 Random, the overhead of random selection and potentially more comparisons for the median calculation contribute to slower execution times.

D. Comparison with Randomized Quicksort

We also implemented and evaluated the Randomized Quicksort algorithm. The comparison revealed that while Randomized Quicksort provides good average-case performance, it is outperformed by Median of Seven (MO7) in most cases, particularly for larger data sets.

VI. Further Observations

The performance difference between pivot selection techniques is minimal for small arrays but becomes more pronounced as data size increases, with faster techniques benefiting from fewer comparisons. Random selection's effectiveness may vary depending on data distribution, occasionally yielding better pivot choices, though its overall benefit may be offset by additional computational costs.

In this experiment, Median of Three (MOT) and Median of Seven (MO7) without Random Selection emerged as the most efficient techniques. MOT achieved an average execution time of 0.0124s, while MO7 recorded 0.0112s, with MO7 demonstrating slightly higher efficiency. Both techniques balance the cost of the median calculation with the effectiveness of the pivot, making them practical choices. The trade-off between MOT's simplicity and MO7's potentially superior pivot selection should be considered for real-world applications.

Impact of Data Distribution and Array Size The experiment utilized randomly generated data, but further investigation is warranted to evaluate performance under varying conditions:

- **Data Distribution:** Analysis of sorted, nearly sorted, and real-world datasets (e.g., financial or network traffic data) is necessary to assess strategy effectiveness in diverse scenarios.
- **Array Size:** Scaling behavior and overhead impact should be examined across different array sizes. For instance, the overhead of median calculation (e.g., in MO7 and MO9) may diminish with larger arrays, potentially influencing strategy efficiency.

This research provides a foundation for deeper exploration into these factors and their implications for algorithm optimization.

VII. Conclusion and Future Directions

This research establishes a foundation for further exploration into optimizing the Quicksort algorithm. Investigating the impact of data distributions, array sizes, and alternative pivot selection techniques can provide deeper insights into its behavior and optimization potential. Additionally, optimizations such as in-place sorting and tail recursion can enhance efficiency for real-world applications. The experiments conducted in this study reveal that the Median of Three (MO3) with simplified implementation and the Median of Seven (MO7) with improved pivot selection deliver the best performance among the evaluated approaches. The analysis highlights the critical role of pivot selection in Quicksort's efficiency, offering valuable insights into the trade-offs between complexity and performance for specific use cases. Median-based strategies, such as MO3 and MO7, generally outperform simpler methods like selecting the first or last element as the pivot. While random selection techniques provide some benefits, they often introduce additional computational overhead. The optimal strategy may vary depending on factors such as data distribution and array size. In conclusion, this study provides a detailed evaluation of how pivot selection strategies influence Quicksort's performance. The findings suggest that while MO3 offers robust performance, more advanced strategies like MO7 can achieve better execution times, particularly for larger datasets.

References

[1] G. Blelloch. (2012, October) Quicksort and sorting lower bounds.

[2] A. Mohammed and M. Othman, "Comparative analysis of some pivot selection schemes for quicksort algorithm," *Information Technology Journal*, vol. 6, 03 2007.

[3] R. Sedgewick and K. Wayne. (2022, March 9) Quicksort. Princeton University. [Online]. Available: https://algs4.cs.princeton.edu/23quicksort/

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT press, 2009. [Online]. Available: https://mitpress.mit.edu/books/introduction-algorithms-third-edition

[5] M. S. Hossain, S. Mondal, R. S. Ali, and M. Hasan, "Optimizing complexity of quick sort," in *Computing Science, Communication and Security*, N. Chaubey, S. Parikh, and K. Amin, Eds. Singapore: Springer Singapore, 2020, pp. 329–339.

[6] A. Aftab, M. A. Ali, A. Ghaffar, A. U. R. Shah, H. M. Ishfaq, and M. Shujaat, "Review on performance of quick sort algorithm," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 19, no. 2, February 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4602255

[7] M. Aumüller and M. Dietzfelbinger, "Optimal partitioning for dual-pivot quicksort," *ACM Trans. Algorithms*, vol. 12, no. 2, nov 2015. [Online]. Available: https://doi.org/10.1145/2743020

[8] J. Smith and E. Johnson, "Machine learning-based pivot selection for enhanced quick sort performance," *Journal of Algorithms and Data Structures*, vol. 15, no. 3, pp. 217–230, 2021.

[9] V. Kumar and S. Chakraborty, "Experimental analysis of the quick sort algorithm: Can a correct choice of pivot make the worst-case complexity better?" September 2022.