**Information Search and Analysis Skill**

**(ISAS)**

**Analyzing Quick Sort**



Arranged by:

GROUP 4:

1. Abdur Rashid Firdaus
2. Achmad Maulana Ibrahim
3. Ayunda Pramita Kurnia Hapsari

Class: 2CS1

**CEP CCIT**

**FAKULTAS TEKNIK UNIVERSITAS INDONESIA**

**2023**

# PREFACE

Praise be to Allah the Almighty for His blessings and grace, so we can complete this ISAS (Information Search and Analysis Skills) task both in the form of presentation and paper in a timely manner.

The motivation of this analysis is to provide information regarding topics that we learn throughout the first phase of the second semester. This research’s aim is not only to present findings but also to encourage more discussion among the class. We thank our faculty Mr. Tri Agus Riyadi, M,T for the insights and suggestions to enhance this paper quality so that readers can easily digest and understand the topic that is covered in the paper.

Finally, the authors hope that this paper can be useful for all to gain a better understanding into the algorithm and data structure. We also realize that this paper may come with its imperfections, we accept all suggestions and criticisms from readers who are constructive in order for the perfection of this paper. We truly hope that this paper will provide only cover the best analysis.

Depok, March 2024

Author

# TABLE OF CONTENT

# TABLE OF FIGURES

# TABLE OF TABLES

# CHAPTER I INTRODUCTION

# I.1. Background

In one of the many things that we learn as a computer science student is the sorting algorithm. Sorting is a fundamental operation in computer science for optimizing various algorithms and applications. The need for efficient way of sorting has been constant with the ever-growing volumes of data that modern computing must handle. In this case, the Quick Sort algorithm emerges as a standout solution.

Developed and introduced by Tony Hoare in 1960, Quick Sort marked a shift in sorting methodologies. Using the divide and conquer strategy, Quick Sort showed an amazing efficiency in practice, earning its place as one of the most widely used sorting algorithms in the field.

As the computer science subject evolved thanks to the advancements of technology, the analysis of sorting algorithms remains to be looked upon for a significantly long time. This paper provides a comprehensive analysis of the Quick Sort algorithm, explaining its mechanics, efficiency, and comparisons. Help readers understand more to make informed decisions in selecting sorting methods for computational task

# I.2. Writing Objective

The purpose of this paper titled "Analysis of Quick Sort" is to fulfilling the task of operating the algorithm and data structure task and to help authors and readers understand more about sorting algorithm.

# I.3. Problem Domain

This paper will only discuss Quick Sort. The main issues Includes:

* + 1. Quick Sort algorithm
    2. Characteristic of Quick Sort
    3. Implementation
    4. Advantages and Disadvantages
    5. Comparison with other sorting algorithm

# I.4. Writing Methodology

This paper will use a descriptive research methodology to analyze the Quick Sort algorithm. It involves conducting in-depth research, discussions, and ensure all the available facts and data are gathered from trusted and reputable sources.

# I.5. Writing Framework

The writing systematics is written as follows:

# I.5.1. Chapter I Introduction

This chapter will discuss background, writing objectives, problem domains, writing methodology, and writing framework.

# Chapter II Basic Theory

This chapter will provide basic explanation of the things that are related to algorithm and data structures, and the topic for this paper.

# Chapter III Problem Analysis

This chapter will provide a deep discussion of Quick Sort.

# I.5.4. Chapter IV Conclusions and Suggestions

# The last chapter will provide a summary and recommendations that may prove useful to the readers.

# CHAPTER II BASIC THEORY

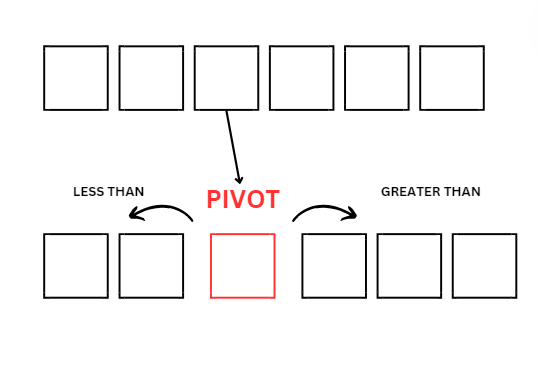
# CHAPTER III PROBLEM ANALYSIS

# Quick Sort

Quick sort is a sorting algorithm based on the divide and conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

The way it works is to place the pivot at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.

# Quick Sort algorithm

Every sorting begins with Quick Sort selects an element called pivot. The pivot is fixed in place by moving all the elements less than that to its left and all the elements greater than that to its right. The partitions of the element sequence into left, pivot, and right are referred as a sorting by partitioning.

**Figure III. 1 Quick Sort Algorithm**

**(https://www.canva.com/design/DAF5j9p4zP8/hIB9J\_Qif\_F89GFNHvykRA/edit)**

# Characteristics of Quick Sort

Quick Sort is a highly effective algorithm used for sorting arrays. It uses a divide and conquer approach. Quick Sort partitions the array into smaller segments, sorting them recursively, and combining the result at the last. This include selecting an element as a pivot, a pivot can be the first, middle or any element of the array. Pivot used as a way to partition the array by placing the element greater than to its right and the element less than to its left until the array is sorted.

Despite its efficiency, Quick Sort is not a stable sort, it means that the relative order of equal elements is not guaranteed to be preserved during the sorting process. Quick sort prioritizes speed and simplicity over stability. This could be a major consideration depending on the specific requirements of a given task.

# Implementing Quick Sort Algorithm

# Quick sort is more efficient when implemented on smaller data sets. Therefore, the examples written below only use small data sets. The data sets implemented here are sequences of integers.

# III.4.1. Implementing Quick Sort in Even Number of Elements

# The array below contains an unsorted sequence of the first 8 prime numbers, excluding the number 2 that is mistyped as 1.

# Choose a pivot in the array.

# 

# Move it to the end.

# 

# Partition the subarray, there will be left and right bounds.

# 

# Move the left bound, 1 index at a time.

# 

# Repeat until the element >= pivot or crossing the right bound.

# 

# If the element in the left bound is greater than the pivot, go to step 8.

# Else, go to step 10.

# Move the right bound, -1 index at a time.

# 

# Repeat until crossing the left bound or index < pivot.

# 

# When it crosses the left bound:

# 

# a. To the left of the left bound, each element < pivot.

# b. To the right of the right bound, each element >= pivot.

# The pivot will be placed to its final location and cannot be picked again as pivot.

# 

# Repeat from step 1 until all the elements are placed to their correct position.

# III.4.2. Implementing Quick Sort in Odd Number of Elements

Below is the quick sort algorithm for sorting a sequence with odd numbers of elements. The chosen example is a sequence of the first until fifth power of 2.

# Choose a pivot in the array.

# 

# Move it to the end.

# 

# Partition the subarray, there will be left and right bounds.

# 

# Move the left bound, 1 index at a time.

# 

# Repeat until the element >= pivot or the left bound crossing the right bound.

# 

# If the element in the left bound is greater than the pivot, go to step 8.

# Else, go to step 10.

# Move the right bound, -1 index at a time.

# (In this example, this step is skipped).

# Repeat until crossing the left bound or element < pivot.

# (This step is skipped as well).

# When it crosses the left bound:

# a. To the left of the left bound, each element < pivot.

# b. To the right of the right bound, each element >= pivot.

# The pivot will be placed to its final location and cannot be picked again as pivot.

# 

# Repeat from step 1 until all the elements are placed to their correct position.

# Advantages and Disadvantages of Quick Sort

# Quick Sort, a highly efficient sorting algorithm, has an excellent average case time complexity of O(n log n) and excels at handling large data sets due to its in-place sorting capabilities. However, there are advantages and disadvantages to using this quick sort. Here are some of the advantages and disadvantages of quick sort:

|  |  |  |
| --- | --- | --- |
| No. | Advantages of Quick Sort | Disadvantages of Quick Sort |
| 1. | Fast and Efficient | Unstable |
| 2. | Cache-friendly | Worst-case time complexity (O(n^2)) |
| 3. | Space complexity (in-place sorting) | Dependent on pivot selection |
| 4. | Predictable average-case performance | Limited suitability for small datasets |
| 5. | Adapts well to many data types | Less suitable for linked lists |
| 6. | Reduced auxiliary space requirements | Recursive implementation – stack overflow |
| 7. | Widely used in practice | Doesn’t guarantee stable sorting |
| 8. | Efficient for large datasets | Requires careful pivot selection |
| 9. | No additional data structure required | Perfomance degradation with sorted data |
| 10. | No additional memory overhead | Suboptimal performance for equal elements |
| 11. | Minimizes comparisons | Difficult to implement for non-comparable data |
| 12. | Supports parallel processing | Complexity in the implementation compared simpler algorithms |
| 13. | Good general-purpose sorting algorithm | Potential security vulnerabilities |
| 14. | Time complexity O(n log n) on average | Poor performance with repetitive data patterns |
| 15. | Can be optimized with different picot selection strategies | Not as intuitive as other sorting algotihms |
| 16. | Degrades gracefully with nearly sorted data | Not suitable for external sorting of large files |
| 17. | Efficient with randomized input data | Requires additional memory for the call stack in recursive implementations |
| 18. | Good trade-off between average-case and worst-case performance | Cache performance is sensitive to the choice of pivot |
| 19. | Laverages devide-and cconquer startegy |  |

# Comparison to Other Sorts

# It is not feasible to compare quick sort with every other sorting algorithms. Many other algorithms are so different in nature that comparing to them are not like apple to apple. A number of sorting algorithms are chosen to be comparisons. The algorithms in question are bubble sort and merge sort, two sorting methods that are often compared to quick sort.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Quick Sort** | **Bubble Sort** | **Merge Sort** |
| **Algorithm Type** | Internal sorting based on divide and conquer strategy. | Simple comparison-based sorting algorithm. | External sorting based on divide and conquer strategy. |
| **Partitioning** | Divides the array into any ratio. | No explicit partitioning. | Splits the array into exactly 2 halves (n/2). |
| **Worst Case Complexity** | O(n^2) due to many comparisons in worst conditions. | O(n^2) due to repeated passes over the entire array. | O(n log n) for both worst and average cases. |
| **Dataset Size** | Efficient for smaller datasets. | Inefficient for large datasets. | Works well with any dataset size. |
| **Additional Storage** | In-place sorting, no additional memory required. | In-place sorting, no additional memory required. | Requires additional memory for auxiliary arrays. |
| **Efficiency** | Faster for smaller arrays. | Slow for all cases, especially large arrays. | Faster for larger arrays. |
| **Sorting Method** | Internal sorting in main memory. | Internal sorting in main memory. | External sorting (data doesn’t fit in memory; uses auxiliary memory). |
| **Stability** | Unstable (two elements with the same value are not guaranteed to maintain their original order). | Stable (equal elements maintain their order in sorted output). | Stable (equal elements maintain their order in sorted output). |
| **Preferred For** | Arrays. | Educational purposes or small datasets. | Linked lists. |

# III.5. Advantage and Disadvantages

# III.6. Comparison to Other Sorting Algorithms

# CHAPTER IV CONCLUSION AND SUGGESTION

# Conclusion

S

# Suggestion

F

# BIBLIOGRAPHY

1. **Al-Kharabsheh et al**., 2013. *Review on Sorting Algorithm*. Aqoba: Academia.edu.