
# Analysis of Algorithms

## BLG 335E

## Project 1 Report

ATAHAN ÇOLAK
colakat21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 03.11.2024

# 1. Implementation

## 1.1. Sorting Strategies for Large Datasets

Apply ascending search with the algorithms you implemented on the data expressed in the header rows of Tables 1.1 and 1.2. Provide the execution time in the related cells. Make sure to provide the unit.

| | tweets | tweetsSA | tweetsSD | tweetsNS |
|---|---|---|---|---|
| **Bubble Sort** | 21.858 sec | 0.0003 sec | 22.378 sec | 7.934 sec |
| **Insertion Sort** | 5.4808 sec | 0.0056 sec | 11.0486 sec | 0.295 sec |
| **Merge Sort** | 0.4633 sec | 0.3436 sec | 0.3423 | 0.3407 sec |

**Table 1.1:** Comparison of different sorting algorithms on input data (Same Size, Different Permutations).

| | 5K | 10K | 20K | 30K | 40K | 50K |
|---|---|---|---|---|---|---|
| **Bubble Sort** | 0.204 sec | 0.840 sec | 3.453 sec | 7.791 sec | | 22.058 sec |
| **Insertion Sort** | 0.055 sec | 0.228 sec | | 2.062 sec | 0.873 sec | |
| **Merge Sort** | 0.00355 sec | 0.00722 sec | 0.0156 sec | 0.0239 sec | | 0.0414 sec |

**Table 1.2:** Comparison of different sorting algorithms on input data (Different Size).

### Discuss your results

Answer here.

## 1.2. Targeted Searches and Key Metrics

Run a binary search for the index 1773335. Search for the number of tweets with more than 250 favorites on the datasets given in the header row of Table 1.3. Provide the execution time in the related cells. Make sure to provide the unit.

| | 5K | 10K | 20K | 30K | 40K | 50K |
|---|---|---|---|---|---|---|
| **Binary Search** | 3e-06 sec | 5e-06 sec | 5e-06 sec | 5e-06 sec | | 5e-06 sec |
| **Threshold** | 2.4e-05 sec | 8e-05 sec | 9.2e-05 sec | 0.000135 sec | | 0.000233 sec |

**Table 1.3:** Comparison of different metric algorithms on input data (Different Size).

### Discuss your results

I did get the expected results on the sorting algorithms except the bubble sort algorithm. My algorithm tries to initialize a Tweet element in every iteration that is the cause of the unexpected time result. My metric functions works on the expected time gaps.

## 1.3. Discussion Questions

### Discuss the methods you've implemented and the complexity of those methods.

Bubble Sort: The algorithm repeatedly steps through the list, comparing adjacent elements and swapping them if they are out of order. This process continues until no more swaps are necessary, meaning the list is sorted. The complexity is O(n$^2$).

Insertion Sort: Insertion sort builds the final sorted array one element at a time by comparing each new element to the already sorted portion and placing it in the correct position. The complexity is O(n$^2$).

Merge Sort: Merge sort splits the array into two halves, recursively sorts each half, and then merges the sorted halves back together. The complexity is O(n$\log n$).

Binary Search: Binary search algorithm repeatedly divides the array in half, comparing the middle element to the target key and eliminating the half where the key cannot be. The complexity is O($\log n$).

Count Above Threshold: This function scans the array of tweets, counting how many tweets have a specific metric value (retweets or favorites) above a given threshold. The complexity is O(n).

### What are the limitations of binary search? Under what conditions can it not be applied, and why?

Binary search requires that the array be sorted. If the array is unordered, the search will not work correctly because binary search relies on eliminating half of the possible search space based on the middle element. Additionally, binary search needs to be used with data structures that allow for constant-time access to elements.It cannot be applied effectively to structures like linked lists, where accessing the middle element takes linear time.

### How does merge sort perform on edge cases, such as an already sorted dataset or a dataset where all tweet counts are the same? Is there any performance improvement or degradation in these cases?

Merge sort consistently performs at O(nlogn) regardless of the input order. This means that even for an already sorted dataset, it still goes through the full process of dividing and merging. Unlike simpler algorithms like insertion sort, which can take advantage of sorted input, merge sort doesn't benefit from pre-sorted or identical data. Because of that, its performance remains the same whether the list is sorted, reversed, or filled with identical elements.

## Were there any notable performance differences when sorting in ascending versus descending order? Why do you think this occurred or didn’t occur?

In general, the performance of sorting algorithms should be similar whether sorting in ascending or descending order, because the algorithms execute the same number of comparisons and operations. However, small differences can occur based on how the data is structured. In most cases, efficient algorithms like merge sort do not show notable differences between sorting in ascending versus descending order.