Analysis of Algorithms

BLG 335E

Project 1 Report

Berra MUTLU
mutlub22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: DATE HERE

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 1.3 and 1.4. Provide the execution time in the related cells. Make sure to provide the unit.

	tweets	**tweetsSA**	**tweetsSD**	**tweetsNS**
Bubble Sort	28.965 s	29.118 s	29.079 s	28.767 s
Insertion Sort	16.487 s	0.001 s	34.212 s	0.948 s
Merge Sort	0.047 s	0.035 s	0.035 s	0.040 s

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

	tweets	**tweetsSA**	**tweetsSD**	**tweetsNS**
Bubble Sort	33.515 s	27.840 s	34.888 s	28.087 s
Insertion Sort	12.655 s	3.623 s	20.755 s	3.249 s
Merge Sort	0.043 s	0.039 s	0.039 s	0.040 s

Table 1.2: Comparison of different sorting algorithms on input data for favorite count (Same Size, Different Permutations).

	tweets	**tweetsSA**	**tweetsSD**	**tweetsNS**
Bubble Sort	50.884 s	51.791 s	50.579 s	50.820 s
Insertion Sort	21.374 s	21.389 s	21.550 s	21.425 s
Merge Sort	0.051 s	0.051 s	0.039 s	0.051 s

Table 1.3: Comparison of different sorting algorithms on input data for ID (Same Size, Different Permutations).

	5K	**10K**	**20K**	**30K**	**50K**
Bubble Sort	0.446 s	1.801 s	7.390 s	15.376 s	42.844 s
Insertion Sort	0.193 s	0.747 s	2.886 s	6.472 s	16.471 s
Merge Sort	0.004 s	0.008 s	0.017 s	0.028 s	0.049 s

Table 1.4: Comparison of different sorting algorithms on input data for retweet count (Different Size).

Discuss your results

Bubble sort is not efficient for large datasets. While the initial change of the dataset does not make a big change in the time the size change effects it by showing exponential growth in execution time. It is the slowest one comparing to the other algorithms that were expected from us to implement. This aligns with the O(n^2) time complexity of the algorithm. Overall, Bubble Sort is not an efficient way to sort a large dataset.

Insertion sort performs well with already sorted datasets taking less than a second, but when the dataset is not in a particular order or in reversed order like tweetsSD which is descending order, it is inefficient, reaching more than 30 seconds due to its O(n^2) time complexity. While it still is faster than Bubble Sort for different sized datasets, it shows the same kind of manner by showing exponential growth in execution time. As a result, Insertion Sort is not an efficient way to sort unordered large datasets.

Merge sort is the most efficient and fast one compared to the other algorithms in this project. It completed the task for different sorted and sized datasets in less than a second, which shows the consistency of the algorithm with it's O(n logn) time complexity. In conclusion, merge sort is an efficient way to sort both big and unordered datasets.

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 1.5. Provide the execution time in the related cells. Make sure to provide the unit.

	5K	**10K**	**20K**	**30K**	**50K**
Binary Search	0.019 ms	0.021 ms	0.010 ms	0.008 ms	0.007 ms
Threshold	0.116 ms	0.227 ms	0.458 ms	0.692 ms	1.124 ms

Table 1.5: Comparison of different metric algorithms on input data (Different Size).

Discuss your results

Binary Search takes minimal time across all dataset sizes. This is expected due to its O(log n) time complexity, making it efficient even for larger datasets like 50K tweets.

Threshold Search shows a slight increase in time as the dataset size grows, but it remains very efficient. For 50K tweets, it completes in just 0.124 miliseconds, which is acceptable for real-time applications.

In conclusion both Binary Search and Threshold Search method perform very well and scale efficiently with increasing dataset sizes. This efficiency makes them suitable for large datasets and real-time querying tasks.

1.3. Discussion Questions

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

- Bubble Sort: $O(n^2)$ Bubble Sort uses nested loops to repeatedly swap adjacent elements if they are in the wrong order, making it inefficient for large datasets.
- Insertion Sort: $O(n^2)$ Insertion Sort performs well for small or nearly sorted datasets but suffers with unsorted data due to shifting operations.
- Merge Sort: O(n log n) Merge Sort divides the dataset recursively and merges them in sorted order, making it efficient for both large and unsorted datasets.
- Binary Search: O(log n) Efficient for searching sorted datasets by dividing the search range in half on each step.
- Threshold Search: O(n) Iterates through the dataset and counts the elements that meet a certain condition. Efficient for moderate-sized datasets.

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

Binary search can only be applied to the sorted datasets. Since the method expects that splitting the array in half will isolate the target element, it cannot reliably find the correct element if the dataset is not sorted. Sorting the dataset first is necessary for unsorted data, which adds to the computational expense.

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 still runs in O(n logn) time complexity when the data is already sorted or all the elements are the same, as it continues to divide and merge regardless of the initial order. Therefore, there is no improvement in performance, but there is also no degradation.

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

Sorting in ascending vs. descending order generally doesn't result in significant performance differences for algorithms like Merge Sort, as they follow the same recursive division and merging process. The initial structure of the data, however, might affect how well algorithms like Bubble Sort and Insertion Sort perform. In Insertion Sort, nearly sorted data in ascending order is much faster than reverse-sorted data, as fewer shifts are required. Although there is less of a performance difference with Bubble Sort, reverse-sorted data often takes longer since more swaps are needed.