Analysis of Algorithms

BLG 335E

Project 1 Report

Melike Beşparmak
besparmak22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 3/11/2024

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	75433 ms	63619 ms	93764 ms	69997 ms
Insertion Sort	13788 ms	1 ms	717 ms	27803 ms
Merge Sort	46 ms	41 ms	45 ms	49 ms

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

	5K	**10K**	**20K**	**30K**	**40K**	**50K**
Bubble Sort	881 ms	5434 ms	13848 ms	31704 ms	55673 ms	96362 ms
Insertion Sort	155 ms	628 ms	3165 ms	6455 ms	10812 ms	20869 ms
Merge Sort	6 ms	11 ms	26 ms	31 ms	46 ms	67 ms

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

1.1.1. Discussion

The unit for all measurements is milliseconds, and the algorithms were tested with the ascending flag enabled.

Bubble Sort

Same Size, Different Permutations

Different permutations do not significantly affect the execution time of Bubble Sort. Since the algorithm works with a time complexity of $O(n^2)$, it is independent of the input permutation. This means the algorithm will iterate over the array with two nested loops in each case, whether it needs to swap or not. A possible optimization for this algorithm is to use a swapped flag, but it was not implemented in this code to observe the algorithm's behavior accurately.

Different Sizes

The results are expected to grow as the input size grows.

Insertion Sort

Same Size, Different Permutations

The permutation of the input highly affects the Insertion Sort algorithm. It iterates with

an outer for loop and an inner while loop only if the current element is in the wrong place. Therefore, this algorithm will only make the necessary swaps, and the input permutation will affect the execution time. The worst case has a time complexity of $O(n^2)$, while the best case is $O(n)$ if the input is sorted (or nearly sorted).

Different Sizes

The results are expected to grow as the input size grows.

Merge Sort

Same Size, Different Permutations

Merge Sort works in $O(n \log n)$ time, independent of the input permutation, since it will divide, conquer, and merge even if the elements are already in order.

Different Sizes

The results are expected to grow as the input size grows, but not as significantly as $O(n^2)$.

Comparison Among Sorting Algorithms

It is expected that Merge Sort will be the fastest among these three algorithms, based on the theoretical time complexities. However, there are cases where the input is sorted, and Insertion Sort works much faster. The difference between Bubble Sort and Insertion Sort is also notable: although both have a time complexity of $O(n^2)$, this is the general case for Bubble Sort and the worst case for Insertion Sort.

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**	**50K**
Binary Search	725 ns	1854 ns	3846 ns	2268 ns	1730 ns
Threshold	157579 ns	272310 ns	650434 ns	2968658 ns	1980987 ns

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

1.2.1. Discussion

I first measured the execution times with milliseconds and I observed that every execution took 0 or 1 ms for both binary search and linear counting. Therefore I changed the unit to nanoseconds for more accuracy. The linear counter task is expected to take longer than binary search since binary search is more efficient, operating at $O(\log n)$, which divides the data in half each time, compared to $O(n)$, which iterates through the array linearly. The execution time of the linear search algorithm is guaranteed to increase

as the input size grows since it will always iterate through the array, even if just once. Binary search can also be affected, but not as much as linear searching, depending on where the value is located.

1.3. Discussion Questions

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

I implemented bubble sort, insertion sort, and merge sort. The time complexity of bubble sort and insertion sort is $O(n^2)$, making them slower for larger datasets. Merge sort, with a time complexity of $O(n \log n)$, is significantly faster, especially for large datasets, due to its divide-and-conquer approach.

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

Binary search requires the dataset to be sorted; it cannot be applied to unsorted data because it works with comparisons within a structured order to divide and search the dataset.

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 performs similarly even on edge cases like an already sorted dataset or one with all identical values, as it always divides and merges independently of the initial order. Therefore, performance is not affected, since the process and time complexity $O(n \log n)$ remain the same.

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

There were no notable performance differences when sorting in ascending versus descending order with any sorting algorithms, as they compare independently of the order direction. The only thing to point out is Insertion Sort gets affected by the input permutation. As an example, if the flag is descending and the input is in descending order, it will work much faster than an ascending input. For other cases, the algorithms' complexities and execution steps do not change for either order.