Analysis of Algorithms

BLG 335E

Project 1 Report

Feyza Nur Keleş
kelesf22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 02.11.2024

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	31081.1 ms	22056.9 ms	29488.8 ms	22488.9 ms
Insertion Sort	11513.9 ms	1.0025 ms	23199 ms	608.259 ms
Merge Sort	44.061 ms	34.7617 ms	32.4962 ms	37.4928 ms

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

	5K	10K	20K	30K	50K
Bubble Sort	314.042 ms	1237.55 ms	5142.72 ms	11489.2 ms	31192.2 ms
Insertion Sort	116.539 ms	463.905 ms	1936.67 ms	4271.2 ms	11520.4 ms
Merge Sort	3.81492 ms	8.07642 ms	17.1303 ms	25.0014 ms	43.8105 ms

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

Discuss your results

Overall, these results highlight the strengths and weaknesses of each sorting algorithm. Merge Sort is the most efficient choice for large datasets, while Insertion Sort is faster for nearly sorted datasets. However, Bubble Sort shows up to be the most inefficient especially as data size increases.

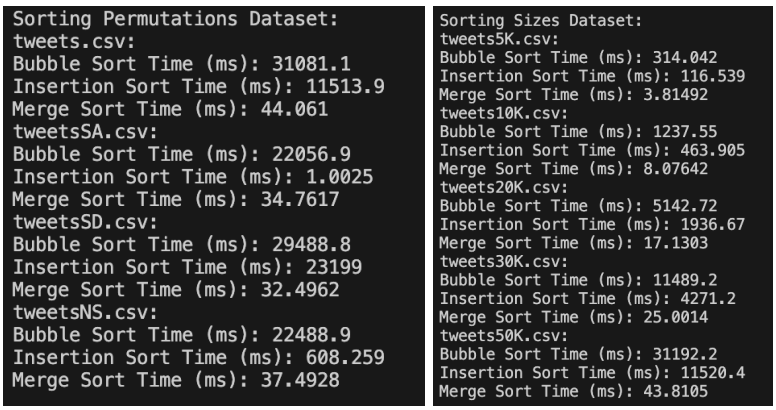

Figure 1.1: Sorting Permutations Dataset and Sorting Sizes Dataset

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	1.54342 ms	2.3535 ms	4.59008 ms	6.33221 ms	10.1295 ms
Threshold Counting	3.93875 ms	4.19542 ms	6.45625 ms	8.04537 ms	12.8668 ms

Table 1.3: Comparison of Binary Search and Threshold Counting times on datasets of different sizes.

Discuss your results

Figure 1.2: Binary Search and Treshold Counting

The binary search times for each dataset show a linear increase with the size of the dataset, which is consistent with the logarithmic time complexity $O(\log n)$ of binary search. The counts of tweets with more than 250 favorites also scale proportionally with the dataset sizes. The rapid increase in counts demonstrates the utility of binary search in efficiently locating data in sorted datasets, confirming its effectiveness for such tasks.

1.3. Discussion Questions

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

- **Counting Tweets Above Threshold**: This function iterates over each tweet to count how many element is above the given treshold of a given metric. *Complexity*:
  - Time Complexity: $O(n)$, as each tweet is examined once.
  - Space Complexity: $O(1)$, using a single counter for counting.
- **Binary Search**: Searches on a sorted list of tweets, finding a target by repeatedly halving the space to be searched. *Complexity*:
  - Time Complexity: $O(\log n)$, as each search step reduces the search space by half.

  - Space Complexity: $O(1)$, with no additional storage required.

- **Bubble Sort**: Bubble Sort repeatedly swaps adjacent elements until the list is sorted. *Complexity*:
  - Time Complexity: $O(n^2)$, For a reverse-sorted array, each element will need to be carried from the beginning to its correct position, requiring n-1 passes. On each pass, it makes up to n-1 comparisons and swaps. But if there was a flag to see if there was any swaps made during an iteration, when theres no swaps needed that means it can exit early. With this flag best case scenario would be $O(n)$. I didnt implement this.
  - Space Complexity: $O(1)$, sorting in place without extra memory.
- **Insertion Sort**: Insertion Sort builds a sorted section of the list by placing each element in its proper position within that section. Time it takes depends on how sorted the space is which makes it unpredictable. *Complexity*:
  - Time Complexity: $O(n^2)$ on worst case, if the space is sorted in reverse order. But $O(n)$ best case, if almost sorted already.
  - Space Complexity: $O(1)$, sorting is done in place.
- **Merge Sort**: This is a divide-and-conquer algorithm that recursively splits and merges the list. *Complexity*:
  - Time Complexity: $O(n \log n)$, as the list is repeatedly divided and merged. Complexity is consistent whether it is sorter or reversed.
  - Space Complexity: $O(n)$, requiring additional space for merging, in total exactly n space would be required.
- **File Read/Write**: These functions handle the reading and writing of tweet data files. *Complexity*:
  - Time Complexity: $O(n)$, as each tweet is read or written once.
  - Space Complexity: $O(n)$, each tweet is stored in memory.

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

Since it depends on whether the middle is greater or smaller than the searched key, binary search requires the space to be sorted already. On an unsorted list we cannot determine whether an element is at the first or second half so we cannot do binary search.

Method	Time Complexity	Space Complexity
Counting Above Threshold	$O(n)$	$O(1)$
Binary Search	$O(\log n)$	$O(1)$
Bubble Sort	$O(n^2)$	$O(1)$
Insertion Sort	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n)$
File Read/Write	$O(n)$	$O(n)$

Table 1.4: Complexity Summary of Methods Implemented

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 maintains its time complexity of $O(n \log n)$ regardless of the dataset's order. As i said before in 1.3 earlier the complexity is consistent whether it is sorted or not because we do not perform any comparisons until each subarray is left with 2 elements.

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

When sorting with the 3 algorithms, since i didnt set up any flags inside the bubble algorithm there wouldnt be any noticable differences. I didnt realise i didnt use "retweet_count" to see the effects of permutation files so i had to change all my images and tables now for the permutation part. When i re-did them i saw the difference with insertion sorting as i expected since when it was sorted the complextity would be $O(n)$ and unsorted would be $O(n^2)$ as i mentioned in 1.3. However bubble and merge sorting remained roughly the same as i expected because both didnt depend on the initial positions. Check 1.3 Bubble Sort for more explanation.