Analysis of Algorithms

BLG 335E

Project 1 Report

HİLAL KARTAL
kartalh21@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 26.10.2024

1. Implementation

graphicx

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	25.6042 sec	7.49951 sec	25.7048 sec	9.00466 sec
Insertion Sort	12.3766 sec	0.000616942 sec	24.1704 sec	0.814758 sec
Merge Sort	0.0925266 sec	0.0893565 sec	0.0879231 sec	0.082731 sec

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

	5K	**10K**	**20K**	**30K**	**50K**
Bubble Sort	0.239537 sec	0.909458 sec	3.37447 sec	7.99071 sec	22.0135 sec
Insertion Sort	0.191336 sec	0.491291 sec	1.84814 sec	4.07695 sec	12.2601 sec
Merge Sort	0.00659905 sec	0.0181532 sec	0.0400152 sec	0.0501351 sec	0.0872287 sec

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

Discuss your results

I have sorted the datas by the retweet count as the given files are sorted by retweet counts.

For BubbleSort:

- For Table 1.1: Worst case scenerio would be descending order which is the dataset "tweetsSD" and best case scenerio would be ascending order which is the dataset "tweetsSA".
- For Table 1.2: At average case time complexity is N^2. So:
  - 5K $\rightarrow$ 10K: As N increases by 2 times, time should increase by 4 times.
  - 10K $\rightarrow$ 20K: As N increases by 2 times, time should increase by 4 times.
  - 10K $\rightarrow$ 30K: As N increases by 3 times, time should increase by 9 times.
  - 10K $\rightarrow$ 50K: As N increases by 5 times, time should increase by 25 times.
- All of this holds true.

For Insertion Sort:

- For Table 1.1: Worst case scenerio would be descending order which is the dataset "tweetsSD" and best case scenerio would be ascending order which is the dataset "tweetsSA".
- For Table 1.2: At average case time complexity is N^2. So:
  - 5K $\rightarrow$ 10K: As N increases by 2 times, time should increase by 4 times.
  - 10K $\rightarrow$ 20K: As N increases by 2 times, time should increase by 4 times.
  - 10K $\rightarrow$ 30K: As N increases by 3 times, time should increase by 9 times.
  - 10K $\rightarrow$ 50K: As N increases by 5 times, time should increase by 25 times.
- All of this holds true.

For Merge Sort:

- For Table 1.1: Worst case scenerio would be descending order which is the dataset "tweetsSD" and best case scenerio would be ascending order which is the dataset "tweetsSA". However with Merge Sort all time complexities are same so all times should be close.
- For Table 1.2: At average case time complexity is $N * log(N)$. So:
  - 5K $\rightarrow$ 10K: As N increases by 2 times, time should increase by little more than 2 times.
  - 10K $\rightarrow$ 20K: As N increases by 2 times, time should increase by little more than 2 times.
  - 10K $\rightarrow$ 30K: As N increases by 3 times, time should increase by nearly 3 times.
  - 10K $\rightarrow$ 50K: As N increases by 5 times, time should increase by nearly 5 times.
- All of this holds true.

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	1.798e-06 sec	1.498e-06 sec	1.399e-06 sec	1.199e-06 sec	7.99e-07 sec
Threshold	9.74e-05 sec	0.000164032 sec	0.000315378 sec	0.000478311 sec	0.000627659 sec

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

Discuss your results

For the binary search, I have sorted the datas first with merge sort.
For Binary Search:

- For Table 1.3: Worst case scenerio would be the element being at the end or the beginning and best case scenerio would be it being at the right middle.
- At average case time complexity is $log(N)$. So:
  - 5K $\rightarrow$ 10K: As N increases by 2 times, time should increase by a constant amount.
  - 10K $\rightarrow$ 20K: As N increases by 2 times, time should increase by a constant amount.
  - 10K $\rightarrow$ 30K: As N increases by 3 times, time should increase by a constant amount.
  - 10K $\rightarrow$ 50K: As N increases by 5 times, time should increase by a constant amount.
- All of this holds true.

For Threshold:

- For Table 1.3: At average case time complexity is N. So:
  - 5K $\rightarrow$ 10K: As N increases by 2 times, time should increase by 2 times.
  - 10K $\rightarrow$ 20K: As N increases by 2 times, time should increase by 2 times.
  - 10K $\rightarrow$ 30K: As N increases by 3 times, time should increase by 3 times.
  - 10K $\rightarrow$ 50K: As N increases by 5 times, time should increase by 5 times.
- All of this holds true.

1.3. Discussion Questions

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

1.3.0.1. Bubble Sort

For Bubble Sort, code starts by checking the ascending or descending with if-else. Then according to the `sortBy` value, it goes into the if-else condition. In these conditions, there are for loops that does the actual bubblesort.
First, we define two integers i and j; i makes it so that we can make the list smaller as we sort the elements and j is to compare elements. Then, we define `Tweet temp` to store

the one of the changing values to not lose it while the change happens.
Complexity:

- `for(int i = (tweets.size()-1); i >= 0; i-)` is so inner loop will run n times.;
- `for(int j = 1; j <= i; j++)` this loop will run decreasingly starting from $N-1$ to 1.
- when we add up these values $(N-1)+(N-2)+...+1 = ((N-1)+1)*N/2 = N^2/2$
- So time complexity of Bubble Sort is $O(N^2)$

1.3.0.2. Insertion Sort

For Insertion Sort, code starts by checking the ascending or descending with if-else. Then according to the `sortBy` value, it goes into the if-else condition. In these conditions, there are for loops that does the actual insertion sort.
First, we define two integers i and j; i makes it so that we can make the list smaller from the beginning as we sort the elements and j is to compare elements. Then, we define `Tweet temp` to store the one of the changing values to not lose it while the change happens.
Complexity:

- `for(int i = 1; i <= (tweets.size() - 1); i++)` to keep up with j.
- `while(j > 0 && tweets[j-1].tweetID > tweets[j].tweetID)` this loop will run increasingly starting from 1 to $N-1$.
- when we add up these values $(N-1)+(N-2)+...+1 = ((N-1)+1)*N/2 = N^2/2$
- So time complexity of Insertion Sort is $O(N^2)$

1.3.0.3. Merge Sort

For Merge Sort, there are two functions: merge() and mergeSort().

Firstly, in merge function: We make two new vectors for dividing and we pushback the datas to their corresponding vectors. Then, to merge we start by checking the ascending or descending with if-else. Then according to the `sortBy` value, it goes into the if-else condition. In these conditions, there are while loops that does the actual merge.

First, we define three integers i, j and k; with i, we follow the first vector and with j, we follow the second vector. Then, we compare the i^{th} and j^{th} values from their vectors and do the necessary changes to tweets vector.

Secondly, in the mergeSort function: We divide the vectors in half and then merge them together, recursively.
Complexity:

- While Dividing, as we divide to the half recursively it takes $O(log(N))$ time.
- And while merging, as we merge all the elements back together, it takes $O(N)$ time.
- So time complexity of Threshold is $O(N * log(N))$.

1.3.0.4. Binary Search

For Binary Search, code starts by defining two integers to keep lower and upper bounds. Then according to the `sortBy` value, it goes into the if-else condition. In these conditions, there are while loops that goes on until upper bound is smaller than lower bound. Inside the loops, we define a new integer mid to keep the middle of the lower and upper bounds.

In binary search, code checks the middle element and if it is smaller than the key, we change the lower bound. If it is bigger than the key, we change the upper bound accordingly. When we find the element we return it's index.

Complexity:

- As the search divides the list by 2 at each iteration, time complexity becomes $O(log(N))$.
- So time complexity of Threshold is $O(log(N))$.

1.3.0.5. Threshold

For Threshold, code starts by defining an integer named count as 0 to count the number of tweets that are bigger than the threshold. Then according to the `metric` value, it goes into the if-else condition. In these conditions, there are for loops that count up if the value is bigger than threshold.

Complexity:

- `int count = 0;` is $O(1)$;
- `for(Tweet tweet : tweets)` these loops are $O(N)$
- $O(1) + O(N) = O(N)$
- So time complexity of Threshold is $O(N)$

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

To be able to do binary search, the given list must be a sorted list. When the list is not sorted, binary search can not be applied because we lose the control of lower and upper bounds.

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?

Ordering of datasets does not change the performance of merge sort. When the list is sorted it becomes slightly faster but as it is by such a small amount, it can be negligible.

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

As we were trying to sort everything in ascending order, the datasets with ascending order were much faster than the descending ones. Because while sorting, the algorithms had to change the all elements of the descending datasets.

References

https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm -> pseudocode
https://www.youtube.com/watch?v=JU767SDMDvA -> pseudocode
https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm -> pseudocode