Analysis of Algorithms

BLG 335E

Project 1 Report

Beyza TÜRK 150220704
turkbe22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 28.10.2024

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	20.1911 sec	0.000661 sec	22.9089sec	7.57091 sec
Insertion Sort	11.8797 sec	0.000601 sec	24.0458 sec	0.641321sec
Merge Sort	0.077226 sec	0.072607 sec	0.078901 sec	0.076937 sec

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

	5K	10K	20K	30K	50K
Bubble Sort	0.208195 sec	0.942194 sec	3.34819 sec	7.45218 sec	20.714 sec
Insertion Sort	0.187077 sec	0.66205 sec	2.80222sec	4.74477 sec	12.2588 sec
Merge Sort	0.009742 sec	0.016933sec	0.031905 sec	0.055636sec	0.086824sec

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

Discuss your results

It is obvious that merge sort's time complexity does not depend on input's permutation, however insertion and bubble sort changes significantly whether the data is sorted in ascending order (best case) or the data is sorted in descending order(worst case). In addition to that we can observe that even if the time complexity of the insertion and bubble sort algorithm are same, insertion sort works faster due to its fewer steps of comparison and swaps. In second table we can observe that no matter what algorithm has been used time complexity increased as input size increased. If we order them according to their speed in descending order it would be merge, insertion, bubble.

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	0 sec	1e-06 sec	2e-06 sec	7e-06 sec	7e-06 sec
Threshold	0.000214 sec	0.000462 sec	0.000867 sec	0.00112 sec	0.001934 sec

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

Discuss your results

We can observe that due to binary search's time complexity ,O(log n), even if data got larger, time that algorithm takes do not change significantly. Threshold checks only the retweetCount or favoriteCount if they are above the threshold, not comparison, swapping, searching etc.

1.3. Discussion Questions

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

- Bubble Sort:
  Best Case:O(n): If the dataset is already sorted the algorithm does not have to swaps it does only comparisons, however our datasets are not sorted.
  Worst Case:O(n^2): Inefficient for large datasets, it has to do repeated comparisons and swaps.

- Insertion Sort:
  Best Case:O(n): If the dataset is already sorted, algorithm only goes through each element once, inserting each into its correct position, however our datasets are not sorted.
  Worst Case:O(n^2): Works better than bubble sort due to fewer comparisons and swaps but still inefficient for large unsorted datasets.

- Merge Sort:
  $\Theta(n \log n)$:Merge Sort's complexity is stable even if dataset is sorted or unsorted due to its divide-conquer approach.

- Binary Search:
  Best Case: O(1): When the given element is at the middle index of the dataset on the first search.
  Worst Case:$O(\log n)$: If the target element is not at the middle index the algorithm has to divide and search until find it.

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

To use binary search, given dataset must be sorted, and dataset must contain comparable type of data.Thus binary search can not applied on unsorted datasets, and not comparable datas.

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 is a divide-conquer algorithm, it applies the same dividing and merging steps whether dataset is sorted or unsorted.Therefore complexity of merge sort is stable we can not improve or degrade the complexity.

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

I have created a ascending coefficient -1 or 1 ,with respect to boolean ascending variable, to reverse the order if its descending.Therefore space complexity increased with constant, therefore performance of my algorithm did not affected. However it could depend on implementation details.