Analysis of Algorithms

BLG 335E

Project 2 Report

Melike Beşparmak
besparmak22@itu.edu.tr
150220061

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 25/11/2024

1. Implementation

1.1. Sort the Collection by Age Using Counting Sort

In this section, the collections were sorted by age using the Counting Sort algorithm. Counting Sort works in linear time, making it particularly suitable for this case due to the limited range of ages in the dataset. The algorithm works by "counting" the occurrences of each element and mapping them to a secondary vector based on their indices.

To handle multiple items with the same age while keeping stability, a vector of lists was used. The list data structure ensures that elements with the same age are preserved in their original order.

1.2. Calculate Rarity Scores Using Age Windows (with a Probability Twist)

Rarity scores were calculated in this section using the formula provided in the assignment file. The calculation involves iterating through all items and comparing each with every other item to determine their similarity. This process requires $O(n^2)$ time complexity due to the nested loops since each item should be compared to every other item.

1.3. Sort by Rarity Using Heap Sort

To sort the collection by rarity scores, Heap Sort was implemented. Heap Sort is an algorithm with a time complexity of $O(n \log n)$, using the heap data structure. At each step, the maximum (or minimum) element is extracted and swapped with a leaf node. This extraction violates the heap's structure, so we call the "heapify" function to reorganize the heap. Heapify function compares the sub-heap's root with its children and makes the necessary swaps for each element.

1.4. Scenario

The scenario was straightforward. First, the items were sorted by age attribute, and then the rarity scores were calculated. Finally, the items were sorted once again by rarity scores.

2. Results

2.1. Measured Execution Times and Comparions

	large	medium	small
Counting Sort	11 ms	6 ms	2 ms
Heap Sort	27 ms	36 ms	5 ms
Rarity Calculation	23509 ms	5941 ms	897 ms

Table 2.1: Comparison of two sorting algorithms and one calculation on input data (Different Size).

The results are as expected. As the dataset grows, the execution times increase. The algorithms work in $O(n)$, $O(n \log n)$ and $O(n^2)$ respectively.

3. Discussion

The increasing execution times across the algorithms highlight the importance of choosing the right algorithm for the task and dataset size. For small to medium-sized datasets, all three methods perform within acceptable limits. However, as the dataset size grows, the quadratic complexity of the rarity score calculation becomes significant. Further optimizations may be needed to keep efficiency.