Analysis of Algorithms

BLG 335E

Project 2 Report

Taha TEMİZ

temizt21@itu.edu.tr

Faculty of Computer and Informatics Engineering

Department of Computer Engineering

Date of submission: 26/11/2024

1. Implementation

1.1. Sort the Collection by Age Using Counting Sort

For the counting sort part, my implementation first get the max age using the getmax function. It creates a vector with size of maxage+1 (because of 0) vector named counts. The counts vector keeps the counts of age of each item in items vector. It checks every element of items vector and stores the count number of them into corresponding value of counts vector. After counting the elements, the algorithm changes every element of counts vector with the sum of the current element with the previous ones. Because we need to get the minimum proper index for every age from 0 to max. For example, if there is 2 zeros and 3 ones than an item with age of 2 can be in minimum index 5(0,0,1,1,1,2). After the summation, the algorithm starts inserting from the end of the items list. Because after inserting, we should decrement the corresponding counts vector element one. If we start from the beginning than the numbers would be mistaken. According the ascending boolean value, the algorithm starts inserting from the left side or the right size of the final vector. I could implement another count vector for descending order but I think it is more efficient this way. The counting sort algorithm checks every item from the given item list and the count list. So the time complexity of counting sort should be O(items.size+ maxage).Durations: small=3631microseconds,medium=3835microseconds,large=3566microseconds). I do not know why the duration for the largest list is shorter than medium.

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

First,the algorithm gets the max age from the item list. Than for all the items in the item vector, it compares the current item with others, if it is in the range then counttotal is incremented by one. If the counted element is also with the same type and the origin with the current item then countsimilar is inceremented by one. After counting, it calculates the rarityscore according to the given formula in the assignment pdf. The critical part is the static cast<double> part. Because if I do not write this part then the algorithm does an integer dividing so the rarityscores would be mistaken. Moreover, there is an interesting part in the formula. If counttotal <=0 then the probabilty is zero. But I think this is impossible according to my implementation. Because while counting, the algorithm counts also the current element. So counttotal and countsimilar are minimum 1. Maybe, my implementation is mistaken but it passed the given tests. The time complexity of this algorithm is O(n2) because it checks every item for every item. My durations are: 2450microseconds for small list, 2048 microseconds for medium list, 2251microseconds for large list. I do not know why.

1.3. Sort by Rarity Using Heap Sort

For the heaping sort, first I implemented a heapify method. This method compares roots with child. For a given vector it also takes a root index. Compares the root with its left and right child. According to the order, it checks if a swap is needed or not. If a swap is needed it swaps the root with the proper child. Then heapifies again. There is also a method named heapsort. This method starts from the middle n/2-1 th index of the given vector. Because after this index, the elements are leafs. It starts from this element and heapifies until the beginning backwards. Then the first element should be the biggest or the smallest. Then it swaps the first and last element.