CS 301 – Algorithms

Homework 2 – 18/03/2019

Atilla Alpay Nalcaci - 19510

Problem 1. Order Statistics

A set of n numbers with the goal of finding the k largest numbers in sorted order.

(a) Relevant comparison-based sorting algorithms by means of returning the k largest numbers in this set are explained below;

- Merge Sort: Described as a divide and conquer algorithm Merge Sort algorithm divides the input array into equal halves and then combines them in a sorted manner. Takes $\theta(k.\log k)$ worst-case asymptotic running time ($\theta(k.\log k)$ in this case).
- Heap Sort: Considered as an improved version of Selection Sort, Heap Sort algorithm uses a binary heap data structure and has a running time of $\theta(n.\log n)$, like merge sort. Additionally, similar to insertion sort, heap sort algorithm sorts in-place, so no extra space is needed during the sorting process.

Considering order-statistics algorithms, in order to acquire the k largest numbers, making use of `get max` and `perlocate down` operations of priority queues will result in k largest numbers as an output. For obvious reasons, `build heap` operation must be performed prior to these operations (namely `get max` and `perlocate down`) and the operations must be performed k times in order to return k results (which will ultimately be our k largest numbers).

Ultimately, we can analyze the overall running time of the algorithm in terms of n and k as;

`build heap` operation → $\theta(n)$

`get max` and `perlocate down` operations → $\theta(k.\log k)$ +

$\theta(n + k.\log k)$

(b) By using Randomized Selection, we can find the k^{th} largest number with $O(n)$ complexity. Following that, as question suggest, partitioning around the k^{th} largest number will take $O(n)$ asymptotic running time as well. Ultimately, sorting these k largest numbers through a comparison-based sorting algorithm, let's say Merge Sort, will take $O(k.\log k)$ asymptotic running time.

Thus, the overall running time of the algorithm will be $O(n + k.\log k)$.

Which method would you use? Please explain why.

Considering the above methods of finding the k largest numbers in a set, I would make use of ***method (b)*** due to the fact that the asymptotic worst-case running time takes less time than both comparison-based sorting algorithms in ***method (a)*** (namely merge sort and heap sort).

Problem 2. Linear-Time Sorting

(a) By nature, Radix Sort algorithm treats an integer as a string of digits, so it is really a sorting algorithm. Counting Sort technique (counting the number of objects having distinct key values [kind of hashing] and then doing some arithmetic to calculate the position of each object in the output sequence.) is used from the least significant digit (LSD) to the most significant digit (MSD) and once the MSD is sorted, the input collection becomes sorted and operation completes.

In order to make it sort string values, we need to define the base accordingly to the elements and lexical ordering of the set. Since we are sorting words, our base will consist of all letters 'a'-'z' inclusively. The algorithm uses a '*' symbol for unoccupied digits which is labeled as the smallest element in value.

Thus, our algorithm will function by comparing the LSD of the given strings with each other until reaching the last MSD of the largest element. In particular;

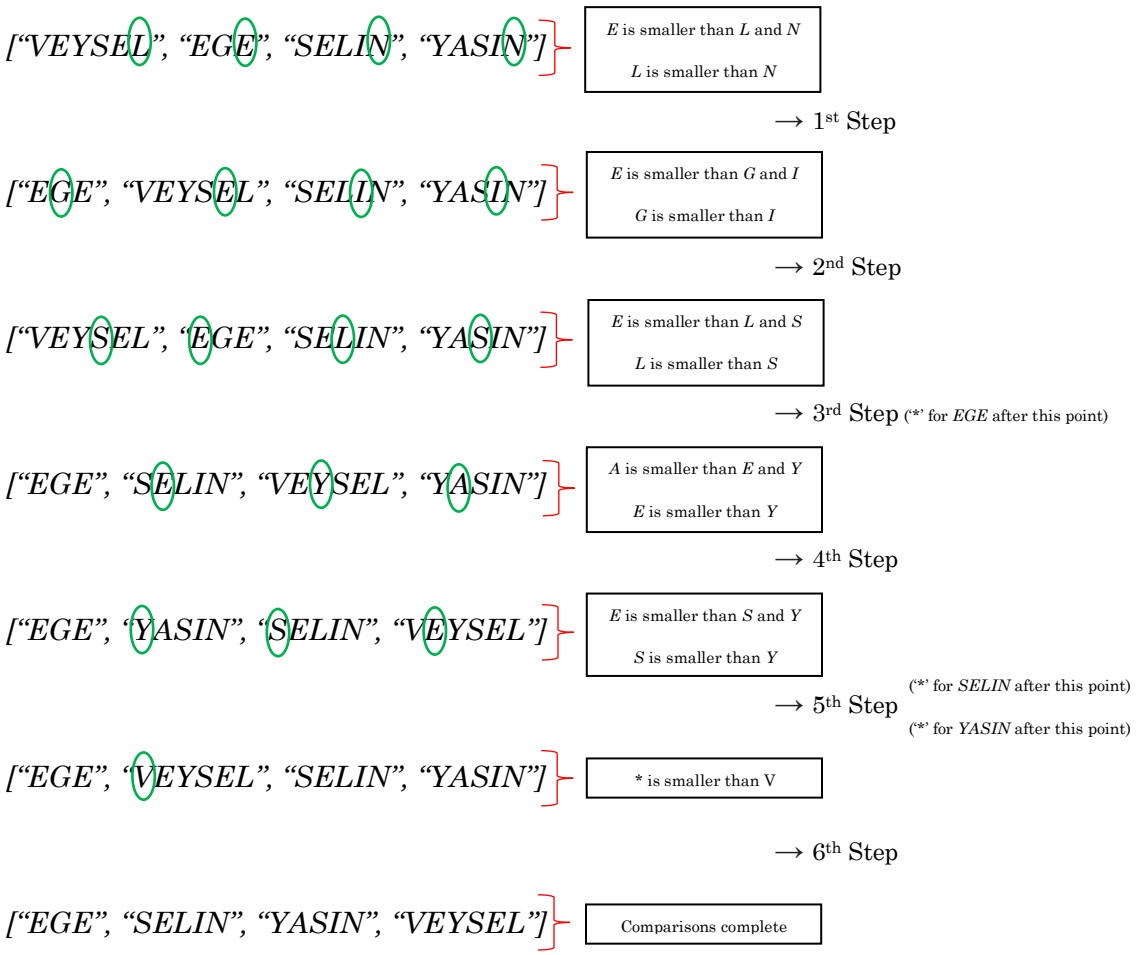
→ Initiate an i value from the LSD to MSD.

→ Sort the given array by comparing the i^{th} digit at each step until the last MSD.

(b) The given list of strings is;

["VEYSEL", "EGE", "SELIN", "YASIN"]

Applying our algorithm from ***part (a)*** we get the following;



Ultimately, our algorithm sorts the given list of strings as;

["EGE", "SELIN", "YASIN", "VEYSEL"]

(c) First thing that can be observed upon analysis is that the number of comparisons being equal to the length of the longest string. For k being the length of the longest string, our complexity becomes $O(k)$. Following that, since we are using a modified base from 'a'-'z' which is equal to 26, our complexity now becomes $O(26k)$. Finally, considering that there are n elements, our complexity becomes $O((n + 26)k) = O(nk + 26k)$.

So the running time of the modified algorithm becomes $O(nk + 26k)$.