CS301 ASSIGNMENT 2

Dilara Nur Tekinoğlu (ID: 27868)

September 28, 2021

1 Problem-1 (Order statistics)

Suppose that you are given a set of n numbers. The goal is to find the k smallest numbers in this set, in sorted order. For each method below, identify relevant algorithms with the best asymptotic worst-case running time (e.g., which sorting algorithm? which order-statistics algorithm?), and analyze the running time of the overall algorithm in terms of n and k.

(a) First sort the numbers using a comparison-based sorting algorithm, and then return the k smallest numbers.

For sorting n numbers in the set, we can use HeapSort algorithm which is a comparison-based algorithm that runs in $O(n * logn)$ time in the worst case. Besides, since it sorts the numbers in place, no additional memory is required. After sorting, listing k smallest numbers takes $O(k)$ time. So the time complexity will be $O(n * logn + k)$.

(b) First use an order-statistics algorithm to find the k'th smallest number, then partition around that number to get the k smallest numbers, and then sort these k smallest numbers using a comparison-based sorting algorithm.

We can apply Select Algorithm to find k'th smallest number and partition the set around it. Select Algorithm runs in $O(n)$ in the worst-case. After parti-

tioning is completed, we can use HeapSort Algorithm to sort these k smallest numbers in $O(k*logk)$ time. So, the total time complexity will be $O(n+k*logk)$.

Which method would you use? Please explain why.

I might prefer different methods based on different values of n and k. If k is large - close to n- both algorithms runs in $O(n*logn)$ time. In this case, I would choose the first method because Select Algorithm runs slow in practice due to large constant factor in front of n.

When k is not large:

- If n is not a large number, I would again choose the first method because of the reason I've just mentioned (large constant factor in Select Algorithm).
- If n is a large number, I would apply the second algorithm since it runs faster than the first one.

2 Question-2

(a) How can you modify the radix sort algorithm for integers, to sort strings? Please explain the modifications.

- If the length of strings to be sorted are different, I first add some character (lets say "*") at the end of shorter strings to make all lengths equal. I specify that the added character is considered smaller than character "a".(e.g."*" = 0)
- Then, starting from "a" = 1 and "A" =1, I give values to all letters ("b" ="B" = 2 and so on). I do not prefer to simply use ASCII values because sorting will be case-sensitive (not alphabetically correct) in such a case. (Note: I assume that the strings consist of only letters. If this is not the case, we should give integers values to other included characters as well.)
- After that, I use Radix Sort Algorithm by comparing characters with respect to their integer values I defined.
- Finally, I remove added characters (*) from the strings.

(b) Illustrate how your algorithm sorts the following list of strings ["AYSU", "BERK", "ESRA", "ILAYDA", "SELIN"]. Please show every step of your algorithm.

We first add "*" characters at the end of strings to make their lengths equal. Then we apply stable sorting on strings starting from the last character. We pass one character left at each iteration. After all iterations are done, we simply remove "*" characters we add at the beginning.

AYSU**	AYSU**	AYSU**	ESRA**	ILAYDA	SELIN*	AYSU**	**AYSU**
BERK**	BERK**	BERK**	SELIN*	SELIN*	BERK**	BERK**	**BERK**
ESRA**	ESRA**	ESRA**	BERK**	ESRA**	ILAYDA	ESRA**	**ESRA**
ILAYDA	SELIN*	ILAYDA	AYSU**	BERK**	ESRA**	ILAYDA	**ILAYDA**
SELIN*	ILAYDA	SELIN*	ILAYDA	AYSU**	AYSU**	SELIN*	**SELIN**

(c) Analyze the running time of the modified algorithm.

In order to analyze the running time, let us first write the pseudocode of the modified algorithm (d = length of the longest string and n = number of strings to be sorted).

Initialize * = 0	$O(1)$
Add necessary number of *s at the end of shorter strings.	$O(d * n)$
Initialize a = A = 1, b= B = 2, ... z = Z = 26	$O(1)$
For i in range length(string)-1 to 0:	$O(d) * O(n)$
Apply stable sorting with respect to i'th character	

Initializing integer values of characters takes constant amount of time. Detecting shorter strings and adding * character at the end of them takes $O(d * n)$ time since there are n strings, len() function takes $O(1)$ in Python, and difference in length can be d-1 in the worst case. If we apply Counting Sort as the stable sorting method, each iteration will take $\theta(n + k)$ where k = 27 (of values that a character can take). Since k is a constant, n will dominate. Therefore we use $\theta(n)$ time for each iteration. Since there are d iterations in total, overall worst-case complexity of the algorithm will be $\theta(d * n)$.