Assignment 2

Gizem Topsakal, 27924

November 8, 2022

1. **(Order statistics)**

   (a) To find the kth smallest element in a list that contains n elements, we need to find the most efficient comparison based algorithm. We can choose the merge sort, since it's worst case running time is $\theta(n*logn)$. We can also calculate the running time from recurrence of merge sort which is T(n) = 2T(n/2) + $\theta(n)$. From master theorem case 2 we can also see that the running time is $\theta(n * logn)$.
   Now our set is order and we can find the kth smallest element by iterating every element until kth element. This will take $\theta(k)$ time. So in total, sorting and iterating until the kth element will take $\theta((n * logn) + k)$ time.

   (b) With the order statistic algorithm I will follow these steps:
   1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
   2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
   3. Partition around the pivot x. Let k = rank(x).
   4.if i=k then return x
   elseif i<k
   then recursively SELECT the ith smallest element in the lower part
   else recursively SELECT the (i–k)th smallest element in the upper part.

   Now to develop the recurrence, we will calculate the algorithm step by step.
   The first step dividing n elements to 5 will take $\theta(n)$ time.

The second step finding the median of n/5 groups will take $\theta(T(n/5))$ time.
The third step partitioning will take $\theta(n)$ time.
The last step will take $\theta(T(3n/4))$ time as explained in the lecture.
Hence the total complexity is:
T(n) = $\theta(T(n/5)) + \theta(T(3n/4)) + \theta(n)$
We can calculate the complexity with substitution method and we will have T(n) = $\theta(n)$. Now we are going to sort these k elements. We can use merge sort since it is an efficient algorithm. Sorting k elements will take $\theta(k * logk)$ time. So in total, the running time is $\theta((k * logk) + n)$.

When I compare these algorithms I would choose the order-statistic algorithm. It is because we know that the k cannot be bigger than n. In the first algorithm, n*logn will dominate and we will end up with $\theta(n * logn)$. The second one will stay as $\theta((k * logk) + n)$. So since we know n*logn dominates n and k can't be bigger than n, in anyways the running time of the first algorithm will be bigger. We can conclude that the efficient algorithm would be the second one.

2. **(Linear-time sorting)**

(a) So in the radix sort for integers, we sort them by first looking at their least significant bits. Then we iterate through the number till we get to the most significant bit. So with integers, when their size are not equal, we simply put 0 in front of them because any number that has three digits are bigger than any number that has two digits. But with letters we cannot put zero because being a longer word does not mean that it should come before the shorter word when sorting. So the first modification we will make to prevent this is to add a number or a special character to the smaller string's ends, and make every string equal in character number. The reasons we should and a number or special character is that we know they dominates letters so this will prohibit any confusion when sorting the words.

In the integer radix sort, we had 10 characters, so we were generating an array of 10 to put the numbers in. Now we have to generate an array of 27, because the English alphabet has 26 letters, and we are adding 1 to that because we will use a special character to make the string lengths equal. Let that special character be "-".

(b) Our initial array is:
["BATURAY", "GORKEM", "GIRAY", "TAHIR", "BARIS"]
So let's apply our modifications. The longest word is BATURAY, hence we will arrange the words according to that. This will make our new array:
["BATURAY", "GORKEM−", "GIRAY−−", "TAHIR−−", "BARIS−−"]
Now we will start by comparing the right most character and iterate.
Step1: ["GORKEM−", "GIRAY−−", "TAHIR−−", "BARIS−−", "BATURAY"]
Step2: ["GIRAY−−", "TAHIR−−", "BARIS−−","BATURAY", "GORKEM−"]
Step3: ["GORKEM−,"TAHIR−−", "BATURAY", "BARIS−−", "GIRAY−−""]
Step4: ["GIRAY−−", "BARIS−−", "TAHIR−−", "GORKEM−", "BATURAY"]
Step5: ["TAHIR−−", "GIRAY−−","BARIS−−", "GORKEM−", "BATURAY"]
Step6: ["TAHIR−−", "BARIS−−","BATURAY", "GIRAY−−", "GORKEM−"]
Step7: ["BARIS−−", "BATURAY", "GIRAY−−", "GORKEM−","TAHIR−−"]

(c) In the integer radix sort, running time was input size n times length of the longest number k, so $\theta(n * k)$. So in this case with the strings, we have an array of 5 so our n is 5, and the longest word has 7 letters so that is our k. So the running time will still be $\theta(n * k)$ but this time k is the length of the longest word.