Duygu Tümer 24842

CS 301 ASSIGNMENT-1

Problem-1)

a) For this algorithm, I would use merge sort or heap sort algorithms. Because if I would use a quick sort algorithm, the worst case time could be $\Theta(n^2)$. Since, I use merge sort (or heap sort), the worst case time is $\Theta(nlogn)$ for sorting these n elements. When we perform a merge sort algorithm, we will have the sorted elements at the end. We can put these k smallest elements in an array. Putting k smallest numbers in the array will take $\Theta(k)$time. So at the end, our algorithm takes $\Theta(k + nlogn)$ time. Since $k \leq n$, the best asymptotic worst-case running time is $\Theta(nlogn)$.

b) I use the SELECT algorithm to find the k'th largest number.This takes $\Theta(n)$ time. Partition around that number to get the k smallest numbers, takes also $\Theta(n)$ time. Now, I have to use a comparison-based sorting algorithm to sort these k smallest numbers. I can use merge sort or heap sort for this purpose. Since we will sort k smallest numbers (which means that I have k elements), with merge sort (or heap sort) this takes $\Theta(klogk)$ time. At the end our algorithm takes $\Theta(n + klogk)$ time. Since $k \leq n$, the best asymptotic worst-case running time is $\Theta(nlogn)$.

I would use the second algorithm (part b) because when k is a small value, this algorithm will take $\Theta(n)$time (nearly). For the first algorithm, for smaller k values it will take $\Theta(nlogn)$ time. So, the second algorithm is always asymptotically better than the first algorithm.

Problem-2)

a) For the radix sort algorithm for integers, we sort digit by digit. For the first iteration, we need to look for the least-significant digit. For the last iteration, we need to look for the most significant digit. For example if we have [3,10, 22, 100], for the first iteration, we will have [10, 100, 22, 3]. For the second iteration, we will have [100, 3, 10, 22]. For the last iteration, we will have [3, 10, 22, 100].
If we want to implement this algorithm for strings, we will do the same logic with a little difference. We can sort characters by using their ascii values. But what if their length will be different? Since for integers, we think that for the previous example above [003, 010, 022, 100] to sort the digits. For strings, if their lengths are different, then we can use '*' character as an empty character. For example if we have ["CS", "COURSE"] we can think as ["CS**", "ANTH"]. So, at the last iteration, we have ["ANTH", "CS**"] since 'A' becomes first.

b) [“AYSU”, “BERK”, “ESRA”, “ILAYDA”, “SELIN”] think as:

[“AYSU**”, “BERK**”, “ESRA**”, “ILAYDA”, “SELIN*”]

For the first iteration we have:
[“AYSU**”, “BERK**”, “ESRA**”, “SELIN*”, “ILAYDA”]

For the second iteration we have:
[“AYSU**”, “BERK**”, “ESRA**”, “ILAYDA”, “SELIN*”]

For the third iteration we have:
[“ESRA**”, “SELIN*”, “BERK**”, “AYSU**”, “ILAYDA”]

For the fourth iteration we have:
[“ILAYDA”, “SELIN*”, “ESRA**”, “BERK**”, “AYSU**”]

For the fifth iteration we have:
[“SELIN*”, “BERK**”, “ILAYDA”, “ESRA**”,“AYSU**”]

For the last iteration we have:
[“AYSU**”, “BERK**”, “ESRA**”, “ILAYDA”, “SELIN*”]

c) Radix sort takes $O(d * (n + b))$where n is the number of items to sort, d is the number of digits in each item, and b is the number of values each digit can have. Since, we only modified b (for integers, it was 10; for strings, it is 26), complexity will be greater than before but still it takes $\Theta(d * n)$time.