Assignment 1

Mehmet Arda Müftüoğlu

November 6, 2022

Student ID: 29547

Problem 1

I would use order statistics because:

If I use merge sort for finding k smallest numbers:
Time complexity of sorting would be $\Theta(nlogn)$ because the recurrence relation of merge sort is $T(n) = 2T(n/2) + c_1 n$. When we use Master Theorem, $n^{log_2 2} = n^1 = n$ and $f(n)$ which is $c_1 n = O(n)$. So the time complexity of merge sort is $\Theta(nlogn)$, and the space complexity of merge sort is $O(n)$ since we store a separate array.
Since we need k smallest numbers, we need to iterate k times to get the numbers which makes time complexity $O(nlogn + k)$.

If we use order statistics to choose k smallest element, we need to partition the given numbers to find k smallest element and then take the subset of terms smaller than k^{th} element and the k^{th} element. Then sort them with merge sort.
At first, set is divided into 5 subsets and their medians are found which has $\Theta(n)$ time complexity. Then the median of the medians are found and it is the pivot of the subset. If pivot is the k^{th} element then the subset which is going to be sorted is the elements smaller than pivot and the pivot. But if the pivot's place in subset is bigger than k, the same steps will be applied on the smaller subset recursively. If pivot's place in subset is smaller than k, then same steps will be applied on the greater subset but with $k = k-$ pivot's order recursively.
When the set is partitioned by pivot, it is known that $\frac{\lfloor \frac{n}{5} \rfloor}{2}$ is smaller than pivot. So in the worst case, the k^{th} element is on the other subset. That's why in the worst-case scenario, the recurrence relation will be

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + \Theta(n)$$

To prove this recurrence relation is $\Theta(n)$ we use substitution method:
$T(n) \leq c_1 n$
Assume $T(k) \leq c_1 k$ for all $k < n$

$$\begin{aligned} T(n) &= T(\frac{n}{5}) + T(\frac{7n}{10}) + \Theta(n) \\ &\leq c_1 \frac{n}{5} + c_1 \frac{7n}{10} + \Theta(n) \\ &= c_1 \frac{9n}{10} + \Theta(n) \\ &= c_1 n - (c_1 \frac{n}{10} - \Theta(n)) \end{aligned}$$

When c_1 dominates $\Theta(n)$ we get desired - something positive so we can say that $T(n) = O(n)$.
To prove $T(n) = \Omega(n)$ which means $T(n) \geq c_2 n$
Assume $T(k) \geq c_2 k$ for all $k < n$

$$\begin{aligned} T(n) &= T(\frac{n}{5}) + T(\frac{7n}{10}) + \Theta(n) \\ &\geq c_2\frac{n}{5} + c_2\frac{n}{10} + \Theta(n) \\ &= c_2\frac{9n}{10} + \Theta(n) \\ &= c_2 n + (\Theta(n) - c_2\frac{n}{10}) \end{aligned}$$

When $\Theta(n)$ dominates c_2 we get desired + something positive so we can say that $T(n) = \Omega(n)$ and since $T(n) = O(n)$ and $T(n) = \Omega(n)$, $T(n) = \Theta(n)$

This means that finding k^{th} smallest element and subset includes numbers smaller than k^{th} smallest element has time complexity of $\Omega(n)$. After this, by using merge sort with complexity found by Master Theorem previously whole time complexity will be $\Theta(n + klgk)$.

Problem 2

a. To sort strings by using radix algorithm, we can use the ASCII codes of letters. First of all we can make all letters uppercase and then we can subtract ASCII code of A from ASCII code of letters to make sure counting array is size of 26, length of English alphabet. So the algorithm would be

   (a) Fill the words that has length shorter than the maximum length with 'A'

   (b) Starting from the maximum length - 1, use counting sort for each index do these operations

      i. Create a count array with the size of 26 and initial values are 0 for all index
      ii. Increment values by 1 when corresponding letter in alphabet is current in words
      iii. Iterate over count array to perform count[i] += count[i - 1] operation
      iv. By using indexes from count array, put words in their sorted order, starting from the last word

b. When words are filled with 'A' string list becomes ["BATURAY", "GORKEMA", "GIRAYAA", "TAHIRAA", "BARISAA"].

   (a) Looking for index 7
   "BATURAY"'s 7^{th} index is 'Y' so increment count[25] by 1
   "GORKEMA", "GIRAYAA", "TAHIRAA", "BARISAA" 7^{th} index is 'A' so increment count[1] by 1, 4 times
   So after count[i] += count[i-1]: count = [4,4,...,5,5]
   "BARISAA" goes to index 4 in output → [, , , "BARISAA",], count = [3, 4, ..., 5, 5]
   "TAHIRAA" goes to index 3 in output → [, , "TAHIRAA", "BARISAA",], count = [2, 4, ..., 5, 5]
   "GIRAYAA" goes to index 2 in output → [, "GIRAYAA", "TAHIRAA", "BARISAA",], count = [1, 4, ..., 5, 5]
   "GORKEMA" goes to index 1 in output → ["GORKEM", "GIRAYAA", "TAHIRAA",

”BARISAA”,], count = [0, 4, ..., 5, 5]
”BATURAY” goes to index 5 in output → [”GORKEM”, ”GIRAYAA”, ”TAHIRAA”, ”BARISAA”, ”BATURAY”], count = [0, 4, ..., 4, 5]

(b) Looking for index 6
”GORKEM”’s 6^{th} index is ’M’ so increment count[13] by 1
“BATURAY”, “GIRAYAA”, “TAHIRAA”, “BARISAA” 6^{th} index is ’A’ so increment count[1] by 1, 4 times
After count[i] += count[i-1]: count = [4,4,...,5,...,5]
”BATURAY” goes to index 4 in output → [, , , ”BATURAY”,], count = [3,4,...,5,...,5]
”BARISAA” goes to index 3 in output → [, , ”BARISAA”, ”BATURAY”,], count = [2,4,...,5,...,5]
”TAHIRAA” goes to index 2 in output → [, ”TAHIRAA”, ”BARISAA”, ”BATURAY”,], count = [1,4,...,5,...,5]
”GIRAYAA” goes to index 1 in output → [”GIRAYAA”, ”TAHIRAA”, ”BARISAA”, ”BATURAY”,], count = [0,4,...,5,...,5]
”GORKEMA” goes to index 5 in output → [”GIRAYAA”, ”TAHIRAA”, ”BARISAA”, ”BATURAY”, ”GORKEMA”], count = [0,4,...,4,...,5]

(c) Looking for index 5
”GIRAYAA”’s 5^{th} index is ’Y’ so increment count[25] by 1
”TAHIRAA”, ”BATURAY” 5^{th} index is ’R’ so increment count[18] by 1, 2 times
”BARISAA”’s 5^{th} index is ’S’ so increment count[19] by 1
”GORKEMA”’s 5^{th} index is ’E’ so increment count[5] by 1
After count[i] += count[i-1]: count = [0,...,1,...,3,4,...,5,5]
”GORKEMA” goes to index 1 in output → [”GORKEMA”, , , ,], count = [0,...,0,...,3,4,...,5,5]
”BATURAY” goes to index 3 in output → [”GORKEMA”, , ”BATURAY”, ,], count = [0,...,2,4,...,5,5]
”BARISAA” goes to index 4 in output → [”GORKEMA”, , ”BATURAY”, ”BARISAA”,], count = [0,...,2,3,...,5,5]
”TAHIRAA” goes to index 2 in output → [”GORKEMA”, ”TAHIRAA”, ”BATURAY”, ”BARISAA”,], count = [0,...,1,3,...,5,5]
”GIRAYAA” goes to index 5 in output → [”GORKEMA”, ”TAHIRAA”, ”BATURAY”, ”BARISAA”, ”GIRAYAA”], count = [0,...,1,2,...,4,5]

(d) Looking for index 4
”GIRAYAA”’s 4^{th} index is ’A’ so increment count[1] by 1
”TAHIRAA”, ”BARISAA” 4^{th} index is ’I’ so increment count [9] by 1, 2 times
”GORKEMA”’s 4^{th} index is ’K’ so increment count[11] by 1
”BATURAY”’s 4^{th} index is ’U’ so increment count[21] by 1
After count[i] += count[i-1]: count = [1,...,3,3,4,...5,...]
”GIRAYAA” goes to index 1 in output → [”GIRAYAA”, , , ,], count = [0,...,3,3,4,...,5,...]
”BARISAA” goes to index 3 in output → [”GIRAYAA”, , ”BARISAA”, ,], count = [0,...,2,3,4,...,5,...]
”BATURAY” goes to index 5 in output → [”GIRAYAA”, , ”BARISAA”, , ”BATURAY”], count = [0,...,2,3,4,...,4,5,...]
”TAHIRAA” goes to index 2 in output → [”GIRAYAA”, ”TAHIRAA”, ”BARISAA”, , ”BATURAY”], count = [0,...,1,3,4,...,4,5,...]
”GORKEMA” goes to index 4 in output → [”GIRAYAA”, TAHIRAA”, ”BARISAA”, ”GORKEMA”,

”BATURAY”], count = [0,...,1,3,3,...,4,5,...]

(e) Looking for index 3
”TAHIRAA”’s 3^{th} index is ’H’ so increment count[8] by 1
”GIRAYAA”, ”BARISAA”, ”GORKEMA” 3^{th} index is ’R’ so increment count[18] by 1, 3 times
”BATURAY”’s 3^{th} index is ’T’ so increment count[20] by 1
After count[i] += count[i-1]: count = [0,...,1,...,4,4,5,...]
”BATURAY” goes to index 5 in output → [, , , , ”BATURAY”], count = [0,...,1,...,4,4,4,5,...]
”GORKEMA” goes to index 4 in output → [, , , ”GORKEMA”, ”BATURAY”], count = [0,...,1,...,3,4,4,5,...]
”BARISAA” goes to index 3 in output → [, , ”BARISAA”, ”GORKEMA”, ”BATURAY”], count = [0,...,1,...,2,4,4,5,...]
”TAHIRAA” goes to index 1 in output → [”TAHIRAA”, , ”BARISAA”, ”GORKEMA”, ”BATURAY”], count = [0,...,0,1,...,2,4,4,5,...]
”GIRAYAA” goes to index 2 in output → [”TAHIRAA”, ”GIRAYAA”, ”BARISAA”, ”GORKEMA”, ”BATURAY”], count = [...,0,1,...,1,4,4,5,...]

(f) Looking for index 2
”TAHIRAA”, ”BARISAA”, ”BATURAY” 2^{th} index is ’A’ so increment count[1] by 1, 3 times
”GIRAYAA”’s 2^{th} index is ’I’ so increment count[9] by 1
”GORKEMA”’s 2^{th} index is ’O’ so increment count[15] by 1
After count[i] += count[i-1]: count = [3,...,4,...,5,...]
”BATURAY” goes to index 3 in output → [, , ”BATURAY”, ,], count = [2,3,...,4,...5,...]
”GORKEMA” goes to index 5 in output → [, , ”BATURAY”, , ”GORKEMA”], count = [2,3,...,4,...,4,...]
”BARISAA” goes to index 2 in output → [, ”BARISAA”, ”BATURAY”, , ”GORKEMA”], count = [1,3,...,4,...,4,...]
”GIRAYAA” goes to index 4 in output → [, ”BARISAA”, ”BATURAY”, ”GIRAYAA”, ”GORKEMA”], count = [1,...,3,...,4,...]
”TAHIRAA” goes to index 1 in output → [”TAHIRAA”, ”BARISAA”, ”BATURAY”, ”GIRAYAA”, ”GORKEMA”], count = [0,...,3,...,4,...]

(g) Looking for index 1
”BARISAA”, ”BATURAY” 1^{th} index is ’B’ so increment count[2] by 1, 2 times
”GIRAYAA”, ”GORKEMA” 1^{th} index is ’G’ so increment count[7] by 1, 2 times
”TAHIRAA”’s 1^{th} index is ’T’ so increment count[20] by 1
After count[i] += count[i-1]: count = [0,2,...,4,...,5,...]
”GORKEMA” goes to index 4 in output → [, , , ”GORKEMA”,], count = [0,2,...,3,...,5,...]
”GIRAYAA” goes to index 3 in output → [, , ”GIRAYAA”, ”GORKEMA”,], count = [0,2,...,2,...,5,...]
”BATURAY” goes to index 2 in output → [, ”BATURAY”, ”GIRAYAA”, ”GORKEMA”,], count = [0,1,...,2,...,5,...]
”BARISAA” goes to index 1 in output → [”BARISAA”, ”BATURAY”, ”GIRAYAA”, ”GORKEMA”,], count = [0,0,...,2,...,5,...]
”TAHIRAA” goes to index 5 in output → [”BARISAA”, ”BATURAY”, ”GIRAYAA”, ”GORKEMA”, ”TAHIRAA”], count = [0,0,...,2,...,4,...]

So at the end the array will be [”BARIS”, ”BATURAY”, ”GIRAY”, ”GORKEM”, ”TAHIR”]

c. At first, we need to find the longest string (say l is the length of it) which takes $O(n)$ time. Then strings with smaller length are filled with 'A' which takes $O(l * n)$ time. So first part takes $O(l * n)$ time.

Then at each iteration, a count array with fixed size k is created and it was 26 in this case. It takes $O(k)$ time. After that, algorithm iterated over all strings so it takes $\Theta(n)$ time. Then, algorithm iterated over count array with size k to perform count[i] += count[i-1] so it takes $O(k)$ time. Then algorithm iterated over all strings from end and it takes $\Theta(n)$ time again. Total time complexity at each iteration then is $\Theta(n + k)$ where n is number of strings and k is the number of count array which is 26.

This process is iterated l times, the maximum length of strings so time complexity becomes $\Theta(l * (n + k))$. (There is also $O(l * n)$ from the first process but it is a lower order term). With big enough n that dominates k which is 26 all the time, the time complexity of algorithm will be $\Theta(n * l)$.