CS301 - HW2

Tahir Turgut - 27035

1 Problem 1

(a) An asymptotically optimal comparison-based algorithm would be enough to sort those array, basically. Let that algorithm be mergesort algorithm which gives us $\Theta(nlogn)$ time complexity (will be explained further). Then, selecting the kth smallest number is just taking the kth element of *sorted* list.

First of all, mergesort algorithm divides the array into two parts and calls another mergesort function for each of the parts (in divide and conquer manner). It will repeat that part until there is only one element -already sorted-. After that, it merges those two parts by traversing them with two distinct pointers, one by one. Since the recurrence relation of mergesort is $T(n) = 2T(n/2) + n$, by applying master theorem, it can be derived that average mergesort is $O(nlogn)$. Also, in both best and worst cases, there is no difference, mergesort still divides array into two parts an merges them. Hence mergesort is $\Theta(nlogn)$.

Secondly, selecting the kth smallest elements can be performed by just iterating the first k elements of the sorted array, like it is mentioned. Any element with given index can be reached in linear time in arrays ($\Theta(1)$). Iterating that k times gives us $\Theta(k)$.

Thus, as a conclusion, finding the kth smallest number by comparison-based sorting algorithm is sorting the array + getting the first kth elements.

Complexity = $\Theta(nlogn) + \Theta(k)$

Since $nlogn$ dominates -even when $k = n$-, total complexity will be same as sorting: $\Theta(nlogn)$

(b) It can be handled by using rand-select basically, but its worst case complexity is $\Theta(n^2)$, in a case that division is issued "unluckily". Thus, to guarantee worst case to be linear time, division should be held carefully. First, the array of n elements is divided into groups of 5 (maybe the last group with less elements). Then, sort those groups of 5, find the medians of them and recursively find the median of the medians of that group to decide pivot, to assure a good partition. Finally, use the rand-select: partition around the well-found pivot, recursively repeating that algorithm (if pivot is less than k the left part, else right part) until pivot is equal to kth smallest number.

After the kth smallest number is found, continue the partition based on that number. Then, sort the left group to find a proper array of kth smallest numbers with merge sort (mentioned in above).

Complexity of that program would be:

Dividing the array into groups of 5: $\Theta(n)$, since there is $\lceil n/5 \rceil$ groups.

Recursively selecting the median of those groups: $T(n/5)$

Partition around the median of medians: $\Theta(n)$, since iterating over each element only once.

And the crucial part rand-select: At least $3\lfloor n/10 \rfloor$ elements are smaller than median, since there is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor$ medians smaller than median and each of those groups there is 3 elements smaller of equal to pivot. That is why, a group may be maximum $n - 3\lfloor n/10 \rfloor \leq 3n/4$. To make calculation simple, recursion of that part: $T(3n/4)$.

Finally solve the recurrence which $T(n) = T(n/5) + T(3n/4) + \Theta(n)$

By substitution: $T(n) \geq \frac{1}{5}cn + \frac{3}{4}cn + \Theta(n)$

$$= \frac{19}{20}cn + \Theta(n)$$
$$= cn - (\frac{1}{20}cn - \Theta(n))$$
$$\leq cn$$

If c is chosen big enough, complexity is $\Theta(n)$

It is more efficient to use second method, since the best asymptotic complexity is better than the first one's. That is why second method should be used in order to find *kth* smallest elements in an unordered list.

2 Problem 2

(a)

(*I assumed that a string can only be consisted of ASCII characters and can be in any length)*

In the integer case, the shorter integers filled with leading 0's (logically) and 0 is prior to any other digit. To sort strings shorter to longer (Ali should be more prior than Aliye, because of the length), it can be said that shorter strings filled with trailing white spaces which are prior than any character. By applying that modification, it is guaranteed that longer string that fully includes shorter one will appear lately (Aliye fully includes Ali in itself).Actually, that can be achieved by a inital check whether there is an index overflow. If there is, the string is shorter and should be prioritize, else its character can be compared.

Another modification would be comparing chars, not single integer. Some languages are able to compare chars with their ASCII values by default, however if it is not guaranteed, chars should be compared with their ASCII values, so an conversion between types may be needed.

(b)

"AYSU "	"AYSU_ "	"ESRA"	"ILAYDA"	"SELIN"	"AYSU"
"BERK "	"BERK_ "	"SELIN"	"SELIN"	"BERK"	"BERK"
"ESRA "	"ESRA_ "	"BERK"	"ESRA"	"ILAYDA"	"ESRA"
"SELIN "	"ILAYDA"	"AYSU"	"BERK"	"ESRA"	"ILAYDA"
"ILAYDA"	"SELIN "	"ILAYDA"	"AYSU"	"AYSU"	"SELIN"

(c) Let define n as element number, l as maximum length of a word.

Counting sort takes $\Theta(n + k)$, where k is the range that a piece can go up maximum. In our case, it is 256 -ASCII characters.

There would be total $l/8$ iterations -string is l byte or char, hence our complexity:

$\Theta(\frac{l}{8}(n + 256))$

256 and 1/8 is constant, so they can be ignored, thus final complexity:

$\Theta(n \cdot l)$, where n is word number and l is maximum length of strings