Problem 1 (Order statistics) **Efe Sencan**

(a) For any comparison based sorting algorithm, $\Omega(nlogn)$ is the lower bound for sorting. On order to prove that any decision tree that can sort n elements must have the height $\Omega(nlogn)$.
Proof:
- If there are n elements in our decision tree, there should be at least n! leaves which denotes the permutation of our elements.
- Since the decision tree is a binary tree with height h, it can have at most 2^h leaves. Therefore,

$n! \leq 2^h$
$log(n!) \leq h$
$log((n/e)^n) <=$
$nlogn - nloge$
= $\Omega(nlogn)$

- Merge Sort have one of the best asymptotic time complexity with O(nlogn) among the comparison based sorting algorithms. Recurrence relation of merge sort:
$T(n) = 2T(n/2) + \theta(n)$ for n >1
$T(n) = 2T(n/2) + cn$ where c >0 by the Master Method:
Case 2: a = 2 b = 2 c = 1 $f(n) = \theta(n^{(log_b a)}log^k n)$ for some constant $k \geq 0$
Thus T(n) = $\Theta(nlogn)$, All in all: We get O(nlogn) for sorting and O(k) for getting k elements, since $k \leq n$ we have = O(k + nlogn) = O(nlogn) time complexity.

(b) I would use a linear time order statistics method algorithm to find the k'th largest number in O(n) time. Let's prove that this algorithm is O(n):
In order to do that, we first divide the n elements into groups which have 5 elements each.(We can sort and find the medians of groups in constant time but since we have n/5 groups we will get $\Theta(n)$) time complexity.We will recursively repeat these steps and select the medians of the medians until we have 5 or lesser elements.(T(n/5)) After we find our median with that method,

this will be our pivot and we will do partitioning around that pivot which takes $\Theta(n)$. Every time we handle with the $\frac{3n}{4}$ elements in the worst case because at least $\frac{3n}{10}$ would be smaller than the median each time. Therefore,
$T(n) = T(n/5) + T(3n/4) + \Theta(n)$
$T(n) \leq T(n/5) + T(3n/4) + \Theta(n)$
= $\frac{19(c.n)}{20}$ + $\Theta(n)$
= c.n - ($\frac{(c.n)}{20}$ - $\Theta(n)$)

Thus it will be $\Theta(n)$) if we pick the wright c values. All in all:
We have $\Theta(n)$ time complexity, for finding the k'th largest element and do partitioning around that element, and we need $\Theta(k)$ time to take k elements from the array. After that, we have to sort these k elements. We can perform this sorting operation in O(klogk) time which we already showed in part a. Totally we have $\Theta(n + klogk)$ time complexity with that algorithm.

I would **prefer** the second algorithm. Because since $k \leq n$, the algorithms perform the same time complexity only k is equal to the n. But elsewhere, the second algorithm has better tight asymptotic time complexity.

Problem 2 (Linear-time sorting) **Efe Sencan**

(a) For sorting integers with radix sort, we use an auxillary array with size our base (for example: for decimal numbers we would need an array of size 10). But for the string case, since our based is changed, we need an array of size base+1 (In this case, it is 27 assuming that we have only uppercase English letters, but our algorithm also works for the all other ascii characters.) Besides that, when comparing integers which have different number of digits, we sort them as if the one with the less number of digit has ”0” in front of the number. Because ”0” in the front does not change the value for the number. But in our string case, we have to add the ”-” symbol(which represents the least valuable character among all the ascii characters) at the of the word instead of placing it in the front. For example (let our letters be ”g” and ”ef”), if we place the that ”-” string in front of the letter ”g”, will be considered to have a smaller index than the ”ef”, which will be wrong. The reason we form an array with size 27 is because, when we see that ”-” character we will increase value of the corresponding(first) index of the array by one.

(b) Since the maximum length of the element in our array is 6, we have to add the ”-” character at the end of the word (although we are not really adding in that ”-” but considering as if there exist a ”-” character) for the words whose length is less than 6.

Step 0: (Initial Step) ”VEYSEL”, ”EGE - - -”, ”SELIN -”, ”YASIN-”

Step 1: ”EGE - - -”, ”SELIN -”, ”YASIN -”, ”VEYSEL”

Step 2: ”EGE - - - ”, ”VEYSEL”, ”SELIN -”, ”YASIN -”

Step 3: ”EGE - - -”, ”SELIN-”, ”YASIN -”, ”VEYSEL”

Step 4: ”EGE - - -”, ”SELIN - ”, ”YASIN - ”, ”VEYSEL”

Step 5: ”YASIN -”, ”SELIN -”, ”VEYSEL”, ”EGE - - -”

Step 6: ”EGE - - -”, ”SELIN -”, ”VEYSEL”, ”YASIN -”

(c) We will have totally k digits for the string case, where k denotes the maximum length of the word in my array, and for each digit, we can apply the counting sort algorithm just as we do in the integer sorting with (n + t) time complexity where t is the number of all possible strings that could be written in one digit. Since, we have k digits, we would apply (n + 27(from our assumption, otherwise our base + 1)) time complexity , k times. As a result, we get O(n.k + 27.k) time complexity. When the n value is be grater than the 27(actually base + 1) value, than our overall time complexity would be O(n.k).