Homework 2

Problem 1: (Order Statistics) All the unstated logs are in base 2.

- **(a)** First to sort the numbers, we can use **merge sort** or **heap sort** instead of quicksort or insertion sort which have **$\Theta(n^2)$** worst-case running time. Because those two algorithms (merge, heap sorts) take **$\Theta(n \log n)$** worst-case running time.
  - Then, we can access the k^{th} largest numbers directly in the sorted set, it takes **$\Theta(k)$** time.
    - Worst case running time of the algorithm: **(We know that; k<=n)**
      - **$T = \Theta(n \log n + k) = \Theta(n \log n)$**
- **(b)** First to find the k^{th} largest number, we can use **Selection algorithm** which can take **$\Theta(n)$** worst-case running time.
  - Then, to get k largest numbers, we can partition around that number in **$\Theta(n)$** worst-case running time.
    - As same as part (a) we can sort the k largest numbers by using **merge sort** or **heap sort** which can take **$\Theta(k \log k)$** worst-case running time.
    - Worst case running time of the algorithm:
      - **$T = \Theta(n + k \log k)$**
- **In terms of running time both methods are asymptotically same as each other. Because in the worst case both of them have the same running time. But I rather to choose the second one because as a worst case we assume that k=n, but if it is not true then in this case second method might have a better running time than first one.**

Problem 2: (Linear-time sorting) All the unstated logs are in base 2.

- **(a)** To use the radix sort, length of the objects that we want to make a comparison between have to have same length. Our base for radix sort will contain all letters and "*" character.
  - To satisfy that constrain we have to make an adjustment to shorter strings until all objects in a given set have the same length. It can be done by adding "*" at the end (to at least significant character) of the strings.
- **(b)** After adjustments done as explained at part (a), we assume that list of strings like this at initial position: ["VEYSEL", "EGE***", "SELIN*", "YASIN*"]
  - **First Iteration:** ["EGE", "SELIN", "YASIN", "VEYSEL"] **(comparison done between least significant character)**
  - **2^{nd} Iteration:** ["EGE", "VEYSEL", "SELIN", "YASIN"]
  - **3^{th} Iteration:** ["EGE", "SELIN", "VEYSEL", "YASIN"]
  - **4^{th} Iteration:** ["EGE", "SELIN", "YASIN", "VEYSEL"]
  - **5^{th} Iteration:** ["YASIN", "SELIN", "VEYSEL", "EGE"]
  - **Last Iteration:** ["EGE", "SELIN", "VEYSEL", "YASIN"] **(comparison done between most significant character)**
- **(c)** For a set of N objects worst-case running time of the radix sort, which have base B, is: **$\Theta(N*B)$**
  - Since we know that and same iterations in use for the strings, worst case running time of the radix sort for string doesn't change:
    - For N objects in base B; **$T = \Theta(N*B)$**
    - For the case at part (b) there are 3 elements in the given set (N = 3) and base contain capital letters and "*" character (A to Z and *).