Homework assignment 6:
Due date: Sunday, October 25 2020 at 11:59pm

1- Find the
   a. 2^{nd} least element
   b. 4^{th} least element
   using the random find statistics algorithm (Quickselect).

   - 2, -1, 3, 8, 9, 0, 19, 6, 35, 17, 20
   - 10, 11, 12, 13, 14, 15, 16, 17, 18

2- Calculate the worst-case running time of Quickselect. (e.g. when you are looking for the n^{th} least element in a sorted array with the size of n)
3- Calculate the average-case running time of QuickSelect algorithm. Explain.
4- Explain an algorithm to return the max k numbers from an unsorted array. (The average running time of your algorithm should be *O(n)*)
5- Suppose a student wrote the below code for the previous question (**Maximum Subsequence Sum problem (MSS)**). What is the running time of this algorithm?

```
max_sum = 0;
for(i=0; i< n; i++)
        {
                for(j=i; j< n; j++)
                        {
                                sum=0;
                                for(k=i; k <= j; k++)
                                        sum += a[k];
                                if(sum > max_sum)
                                        max_sum = sum;
                        }
}
return max_sum;
```

6- Suppose another student who is a better programmer wrote the below code for **Maximum Subsequence Sum problem (MSS)**. What is the running time of this algorithm?

```
max_sum = 0;
for(i=0; i< n; i++)
        {
                sum= 0;
                for(j=i; j< n; j++)
                        {
                                sum += a[j];

                                if(sum > max_sum)
                                        max_sum = sum;
                        }
        }
        return max_sum;
```

7- Suppose another student decides to solve the **Maximum Subsequence Sum problem (MSS)** using divide and conquer technique:

Divide: Divide the array into two halves.

Conqure: Recursively find MSS of the two sub-arrays, each of size n/2,

Combine: Now in order to combine the sub-arrays you need to know that MSS can lie in one of the following places:

1- Entirely in the left sub-array

2- Entirely in the right sub-array

3- Intersects both halves:

(We can easily find a maximum sub-array crossing the midpoint in time **linear** in the size of the sub-array. This problem is *not* a smaller instance of our original problem, because it has the added restriction that the sub-array it chooses must cross the midpoint. Any sub-array crossing the midpoint is itself made of two sub-arrays a[i…mid] and a[mid+1..j], where *i* is an *index* in the left sub-array, and *j* is an *index* in the right sub-array. Therefore, we just need to find maximum sub-arrays of the form a[i..mid] and a[mid+1..j] and then combine them.)

Therefore, to choose MSS as the final answer you need to find the max between these three.

What is the running time of this algorithm? (Note: You can read the complete description of MSS from your textbook: page 70-73)

8- Suppose an extraordinary student decides to solve the **Maximum Subsequence Sum problem (MSS)** using the below code. What is the running time of this algorithm?

```
max_sum = 0;
sum = 0;
for(i=0; i< n; i++)
        {
                sum += a[i];

                if(sum > maxSum)
                        max_sum = sum;
                else if(sum < 0)
                        sum = 0;
        }
return max_sum;
```

9- Find the MSS of the below array using solutions for Q.7 and Q.8:

a. 7, 2, -14, 38, 52, -37, 4, 12, -4, 6, 3, 2

b. 3, 6, -20, 11, -15, 26, -43, 10, -14, 27, 0, 39