Programming Homework #2

Q1. **Quick Sort** is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. Implement the **partition()** and **quicksort()** functions to sort the input array A. (*Use pivot with the last element of the input array)

(1) Use your quicksort() for 1000 randomly generated integers and measure the running time. e.g., input = np.random.randint(0, 1000, 1000)

<5 points>

(2) Use your quicksort() for 1000 sorted (in ascending order) integers and measure the running time. e.g., input = np.arange(0, 1000)

<5 points>

(3) Use your quicksort() for 1000 sorted (in descending order) integers and measure the running time. e.g., input = np.arange(1000, 0, -1)

<5 points>

(4) Explain the reason that the running time of Q1-(2) and Q1- (3) is greater than the running time of Q1-(1).

<5 points>

(5) Can you improve the running time of Q1-(2) and Q1-(3) cases by modifying your quicksort algorithm? Show your answer, e.g., running time with your improved quicksort for Q1-(2) and Q1-(3).

<5 points>

Q2. **Heap sort** is implemented by completing functions **heapify()** and **buildHeap().** Generate 20 random inputs and sort in ***<u>descending order</u>*** them by using your **heapSort()**. (You may use min-heap).

<20 points>

Q3. Given an unsorted integer array A, find the **smallest missing positive integer**. (*Hint: you may use the partition concept in Q1, but not required.)

```
A = [-100, 10, 5, 6, -62, 23, 14, 4, 7, -78, 3, -12, 94, 97, -32, 1, 2]
```

- You must implement an algorithm that runs in $O(n)$ time.
- Do not use Python built-in sort functions.

<15 points>

Q4. Max-Min problem is to find a maximum and minimum element from the given array. We can effectively solve it using the divide and conquer approach. Implement an algorithm to find max and min from sequence A (in Q3) using the ***divide-and-conquer*** approach.

- Do not use Python built-in sort functions.

*Hint: divide and conquer approach for Max. & Min. problem works in three stages as follows:

> 1. If a_1 is the only element in the array, a_1 is the maximum and minimum.
> 2. If the array contains only two elements a_1 and a_2, then the single comparison between two elements can decide the minimum and maximum of them.
> 3. If there are more than two elements, the algorithm divides the array from the middle and creates two subproblems. Both subproblems are treated as an independent problem and the same recursive process is applied to them. This division continues until the subproblem size becomes one or two.

- It is not required to follow the above stages but your algorithm should exploit a divide-and-conquer approach.

<10 points>

Q5. Generate a random input array by using “input = np.random.randint(0, 100, 10**3)”. Let’s assume that the *input* is the daily price of an item, e.g., input[0] is the price of day 0, input[1] is the price of day 1, etc. We hope to maximize the profit by buying and selling an item. Find the maximum profit for a given input(price). (Calculate the change of the price with the given integer array *input*, then find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.)

(1) Write a code for the brute-force method and measure its running time.

<10 points>

(2) Propose *any algorithm*, e.g, using the divide-and-conquer approach, which can improve the running of finding *maximum subarray problems*. Measure its running time and compare it with the result of Q5-(1).

<15 points>

(3) Compare the complexity of a brute-force method Q5-(1) vs. your algorithm Q5-(2) by using asymptotic notations.

<5 points>