Home Work № 8

CSCI B505 Applied Algorithms 11/07/2022

Instructions

1. Under any circumstances the deadline will NOT be extended.
2. Code should be written in python language in a Jupyter notebook .
3. Unless mentioned in the questions, feel free to use built-in python functions
4. Upload Jupyter notebook and html file of the Jupyter notebook in canvas. No other forms submissions is considered as valid submission. Make sure to have the output of the required cells of the jupyter notebook and its html version before making submission.
5. Plagiarism is unacceptable and we have ways to find it. So do not do it.
6. There are six problems. Problem 4 is worth 25 points, rest of the five problems worth 15 points each.
7. The code should readable with variables named meaningfully.
8. Write test cases wherever required so that they cover all scenarios.
9. students are expected to use the function signature given in the question or they'll get a penalty.

Problem 1

A timesheet is maintained for employees containing the clock in and clock out times. Given a clock in and clock out times lists, find the points where maximum employees are present in the company and return the tuple (a,b) where a represents the max no of employees and b represents the count of no of such points (i.e. times). Note that if clock in and clock out time coincides, the clock in time is preferred over the clock out time. Expected time complexity: O(nlogn) where n represents the total no of employees in the company. Do not use the built in python sort method.

Input:
clock in times = [1, 2, 4, 7, 8, 12] clock out times = [3, 7, 8, 12, 10, 15]
Explanation:

Time	Clock in/Clock out	Max Employees
1	Clock in	1
2	Clock in	2
3	Clock out	1
4	Clock in	2
7	Clock in	3
7	Clock out	2
8	Clock in	3
8	Clock out	2
10	Clock out	1
12	Clock in	2
12	Clock out	1
15	Clock in	2

```
Maximum no of employees are 3 at time points 7 and 8, Therefore the output
to be returned is Output: (3, 2)
```

Following the below given method signature

```

def maxEmployees(list1, list2):
    ### Logic ###
    ### Logic ###
    ### Logic ###
   return (a,b)
```

Problem 2

Given an integer n, return a largest integer that is smaller than n, that can be created using exchanging the position of any one pair of two digits of n. Return the same given integer if no such integer can be formed. You may assume that none of the digits of n is 0. Expected time complexity is O(N), where N is the number of digits in the given integer.

```
Example:

input: 572
output 527
Explanation: 527 is the largest integer smaller than n that can be formed by swapping
```

```
the position of digit 7 & 2.

input: 18256
output: 16258
Explanation: 16258 is the largest integer smaller than n that can be formed by swapping
the position of digit 8 & 6.
```

Use following function signature:

```python
def find_largest_integer(num : integer):
    #### logic ####
    pass
```

Problem 3

M once told D that being able to implement a bug-free in-place quicksort is a testament to the quality of one's computer science education. Help D implement a bug-free in-place quicksort.

```
Examples:

>>> quicksort([4,3,1,2])
    [1,2,3,4]

>>> quicksort([7,7,3,2,1])
    [1,2,3,7,7]
```

```python
def quicksort(arr):
    pass
```

Problem 4

Let A =[a1, a2, a3, ...an] be a given sequence of integers. The range quantile query RQQ(k, l,r) over A returns the k-th smallest integer in the range A[l. . . .r].
For example, on the given A[1. . . 10] = [6, 2, 0, 7, 9,3 ,1,8,5,4] sequence , RQQ(5,3,9) returns 5th smallest integer in the range A[3. . . . . . 9] = [0, 7, 9, 3, 1, 8, 5]
Implement the range quantile queries with the wavelet tree data structure

```
## This is how the usage looks like

wv_tree = Wavelet_Tree([6, 2, 0, 7, 9, 3, 1, 8, 5, 4])
wv_tree.print()

Level 0: 1001100110
Level 1: 00101, 00110
Level 2: 100, 01, 010, 10
Level 3: 01, X, X, X, 10, X, X, X

wv_tree.RQQ(5, 3, 9)

Level 0: (5,3,9)
Level 1: (2,2,5)
Level 2: (2,2,3)
Level 3: (1,1,1)
```

Create wavelet tree class with two methods print and RQQ

```python
class Wavelet_Tree:
   def __init__(self, A:List[int]):
     pass
   def print(self):
     pass
   def RQQ(self, k:int, left:int, right:int):
     pass
```

Problem 5

Given a list of strings where no string is a substring of another, find the shortest string that contains each string in the list as a substring. The problem needs to be solved using Greedy with $O(n^3)$ complexity. Write down the greedy assumption as comments in the program and provide whether the greedy solution will achieve optimal solution or not.

```
Input:  [CATGC, CTAAGT, GCTA, TTCA, ATGCATC]
Output: The shortest superstring is GCTAAGTTCATGCATC
```

GCTAAGTT**CATGC**ATC

G**CTAAGT**TCATGCATC

GCTAAGTTCATGCATC

GCTAAG**TTCA**TGCATC

GCTAAGTTC**ATGCATC**

Use the following function signature:

```python
def shortest_superstring( A : List[string]):
    ##### logic #####
    pass
```

Problem 6

There exist two circular arrangements of integers circle A and circle B each of size n where n(>3) is an integer not perfectly divisible by 3. If circle A = [a1,a2,a3,a4,. . . . .an-1,an] where a1 and an are adjacent to each other and all the elements are positive. Now circle B is initially formed such that B = [an+a1+a2, a1+a2+a3, a2+a3+a4, ..., an-2+an-1+an,an-1+an+a1]. You are given the values of circle B. Your task is to find the list which represents circle A. Expected time complexity - O(n).

```
Example:

Input: [179,129,62,144,182]
Output: [79,9,41,12,91]
```

Use the following function prototype.

```python

def find_circle_A(B):
# your code goes here
```