Lab 10

Task 01: Array Based Heap Implementation

In this task, you are going to implement a class **StudentMaxHeap**. Each node of this Max Heap will contain the Roll number, and CGPA of a student. **The heap will be organized on the basis of students' CGPAs** *i.e. the student having the maximum CGPA will be at the root of the heap*. The class definitions will look like:

```
class Student:
    def __init__(self, rollNo, cgpa):
        self.rollNo = rollNo
        self.cgpa = cgpa

class StudentMaxHeap:
    def __init__(self, size):
        self.maxSize = size   # Maximum number of students that can be
stored in the heap
        self.currSize = 0     # Current number of students present in
the heap
        self.student = [None] * size  # Array of students which will be
arranged like a Max Heap



    def isEmpty(self):  # Checks whether the heap is empty or not
        return self.currSize == 0

    def isFull(self):  #Checks whether the heap is full or not
        return self.currSize == self.maxSize

    def insert(self, student):

    def removeBestStudent(self):

    def levelOrder(self):

    def height(self):

    def _heapify_up(self, index):

    def _heapify_down(self, index):
```

1. Insert in Heap:

Implement a public member function of the **`StudentMaxHeap`** class which inserts the record of a new student (with the given roll number and CGPA) in the Max Heap. The prototype of your function should be:

```
def insert(self, student):
```

This function should return **true** if the record was successfully inserted in the heap and it should return **false** otherwise. The worst-case time complexity of this function should be $\boldsymbol{O}(\boldsymbol{lg\ n})$.

You can assume that Roll numbers of all students will be unique (different).

2. Remove from Heap:

Now, implement a public member function to remove that student's record from the Max Heap which has the **highest CGPA**. The prototype of your function should be:

```
def removeBestStudent(self):
```

It should return a student object which has a highest CGPA. The worst-case time complexity of this function should also be $\boldsymbol{O(lg\ n)}$.

3. Student MaxHeap

Now, implement the following two public member functions of the **`StudentMaxHeap`** class:

```
def levelOrder(self):
```

This function will perform a level order traversal of the **`StudentMaxHeap`** and display the roll numbers and CGPAs of all the students.

```
def height(self): 0
```

This function will determine and return the height of the **`StudentMaxHeap`**. The worst-case time complexity of this function should be **constant** i.e. $\boldsymbol{O(1)}$.

Driver program:

```
heap = StudentMaxHeap(10)
heap.insert(Student(1, 3.8))
heap.insert(Student(2, 3.9))
heap.insert(Student(3, 3.7))
heap.insert(Student(4, 4.0))
heap.levelOrder()


s = heap.removeBestStudent()
print(f"Removed Student - Roll No: {s.rollNo}, CGPA: {s.cgpa}")

s = heap.removeBestStudent()
print(f"Removed Student - Roll No: {s.rollNo}, CGPA: {s.cgpa}")

heap.levelOrder()
print(f"Height of the heap: {heap.height()}")
```

The output of the following program is:

```
Roll No: 4, CGPA: 4.0
Roll No: 2, CGPA: 3.9
Roll No: 3, CGPA: 3.7
Roll No: 1, CGPA: 3.8
Removed Student - Roll No: 4, CGPA: 4.0
Removed Student - Roll No: 2, CGPA: 3.9
Roll No: 1, CGPA: 3.8
Roll No: 3, CGPA: 3.7
Height of the heap: 1
```

Task 02: Maximum Product of Two Elements in an Array

Given the array of integers nums, you will choose two different indices *i* and *j* of that array. Return the maximum value of (nums[i]-1)*(nums[j]-1).

Example 1:
Input: nums = [3,4,5,2] **Output:** 12
Explanation: If you choose the indices i=1 and j=2 (indexed from 0), you will get the maximum value, that is, (nums[1]-1)*(nums[2]-1) = (4-1)*(5-1) = 3*4 = 12.

Example 2:
Input: nums = [1,5,4,5] **Output:** 16
Explanation: Choosing the indices i=1 and j=3 (indexed from 0), you will get the maximum value of (5-1)*(5-1) = 16.

Example 3:
Input: nums = [3,7] **Output:** 12

Constraints:
2 <= nums.length <= 500
1 <= nums[i] <= 10^3

Task 03: Sort Character By Frequency

Given a string s, sort it in **decreasing order** based on the **frequency** of the characters. The frequency of a character is the number of times it appears in the string.
Return the sorted string. If there are multiple answers, return any of them.

Example 1:
Input: s = "tree" **Output:** "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

Example 2:
Input: s = "cccaaa" **Output:** "aaaccc"
Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.
Note that "cacaca" is incorrect, as the same characters must be together.

Example 3:
Input: s = "Aabb" **Output:** "bbAa"
Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.

Constraints:
1 <= s.length <= 5 * 105
s consists of uppercase and lowercase English letters and digits.