Lab 13
Data Structures
BS DS Fall 2024 Morning/Afternoon

Instructions:

- **Attempt the following tasks exactly in the given order.**
- Make sure that there are no ***dangling pointers*** or ***memory leaks*** in your programs.
- Indent your code properly.
- Use meaningful variable and function names. Follow the naming conventions.
- Use meaningful prompt lines/labels for all input/output that is done by your programs.

Task 1

In this task, you are going to implement a class for Max Heap. Each node of this Max Heap will contain the roll number, name, 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 {

public:
    double cgpa;  // Student's CGPA
    int rollNo;   // Student's roll number
    Student() {
        cgpa = 0.0;
        rollNo = 0;
    }
    Student(int r, double c) {
        rollNo = r;
        cgpa = c;
    }
};
```

```
class StudentMaxHeap {
private:
    Student* st;   // Array of students which will be arranged like a Max Heap
    int currSize;  // Current number of students present in the heap
    int maxSize;   // Maximum number of students that can be present in the heap

    bool isGreater(const Student& a, const Student& b);
    void heapifyUp(int index);
    void heapifyDown(int index);

public:
    StudentMaxHeap(int size);  // Constructor
    ~StudentMaxHeap();         // Destructor
    bool isEmpty();  // Checks whether the heap is empty or not
    bool isFull();   // Checks whether the heap is full or not
};
```

Implement the constructor, destructor, isEmpty and isFull functions shown above in the class declaration.

Task 1.1

Add a member function to 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:

bool insert (int rollNo, double cgpa);

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 ***O*(lg *n*)**. If two students have the same CGPA then their records should be stored in a way such that at the time of removal if two (or more) students have the **same highest CGPA** then the student with smaller roll number should be removed before the students with larger roll numbers.

Task 1.2

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

bool removeBestStudent (int& rollNo, double& cgpa);

Before removing the student's record, this function will store the roll number and CGPA of the removed student in its two arguments. It should return true if the removal was successful and it should return false otherwise. The worst case time complexity of this function should also be ***O*(lg *n*)**.

Task 1.3

Now add the following two member functions to the **`StudentMaxHeap`** class:

void levelOrder ();

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

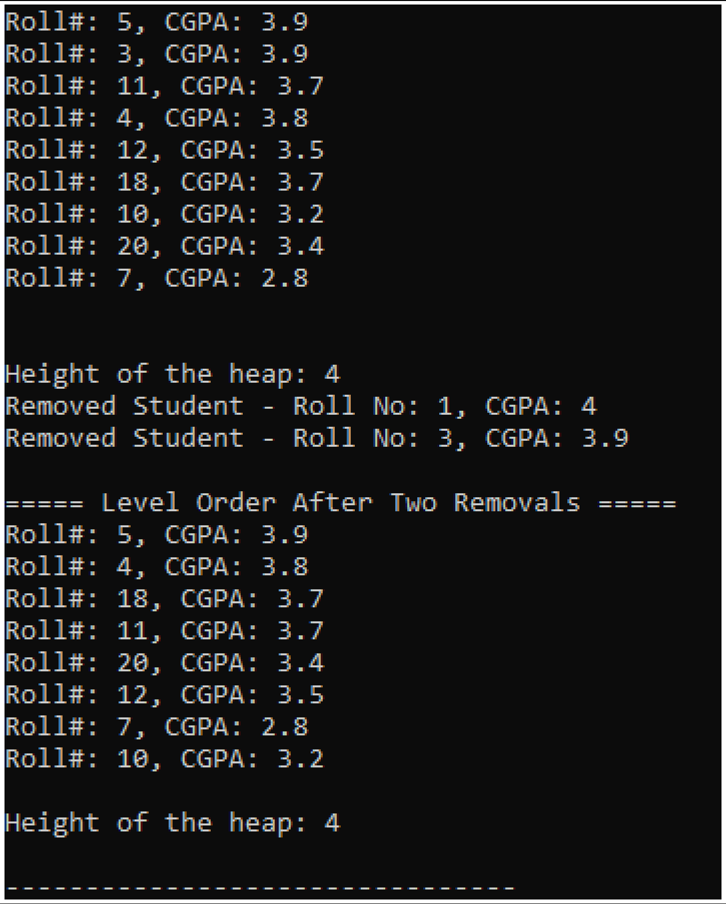
int height ();

This function will determine and return the height of the StudentMaxHeap.

Driver program:

```
int main() {
     StudentMaxHeap heap(20);

    // ---- Add multiple sample students ----
    heap.insert(10, 3.2);
    heap.insert(5, 3.9);
    heap.insert(12, 3.5);
    heap.insert(3, 3.9);   // Same CGPA as roll 5 (tie-break: smaller
roll first)
    heap.insert(7, 2.8);
    heap.insert(18, 3.7);
    heap.insert(1, 4.0);   // Highest CGPA
    heap.insert(20, 3.4);
    heap.insert(11, 3.7);  // Same CGPA as roll 18 (tie-break)
    heap.insert(4, 3.8);

    cout << "===== Level Order After Insertions =====\n";
    heap.levelOrder();
    cout << endl;
    cout << "\nHeight of the heap: " << heap.height() << endl;


    // ---- Remove top student twice ----
    int roll;
    double cg;

    if (heap.removeBestStudent(roll, cg)) {
        cout << "Removed Student - Roll No: " << roll
             << ", CGPA: " << cg << endl;
    }

    if (heap.removeBestStudent(roll, cg)) {
        cout << "Removed Student - Roll No: " << roll
             << ", CGPA: " << cg << endl;
    }

    cout << "\n===== Level Order After Two Removals =====\n";
    heap.levelOrder();

    cout << "\nHeight of the heap: " << heap.height() << endl;

    return 0;
}
```

The output of the following program is:



```
Roll#: 5, CGPA: 3.9
Roll#: 3, CGPA: 3.9
Roll#: 11, CGPA: 3.7
Roll#: 4, CGPA: 3.8
Roll#: 12, CGPA: 3.5
Roll#: 18, CGPA: 3.7
Roll#: 10, CGPA: 3.2
Roll#: 20, CGPA: 3.4
Roll#: 7, CGPA: 2.8


Height of the heap: 4
Removed Student - Roll No: 1, CGPA: 4
Removed Student - Roll No: 3, CGPA: 3.9

===== Level Order After Two Removals =====
Roll#: 5, CGPA: 3.9
Roll#: 4, CGPA: 3.8
Roll#: 18, CGPA: 3.7
Roll#: 11, CGPA: 3.7
Roll#: 20, CGPA: 3.4
Roll#: 12, CGPA: 3.5
Roll#: 7, CGPA: 2.8
Roll#: 10, CGPA: 3.2

Height of the heap: 4

--------------------------------
```

Task 2

You are given a vector of integers. Your task is to find and return the first non-repeating element in the vector. If all elements repeat, return -1. You are not allowed to use any additional data structures, solve this problem using only a vector<int>. **You have to solve it in linear time.**

Constraints:

- The vector can contain up to 1,000,000 elements
- The elements in the vector can range from -1000 to 1000

Sample 1:

Input: 4, 5, 6, 7, 4, 6, 5, 6, 4, 5, 5, 6

Output: 7

Sample 2:

Input: 1, 5, 2, 5, 4, 1, 2, 3, 4, 3, 5

Output: -1

Task 3

You are given a string that contains only uppercase English alphabets (A-Z). Your task is to print each letter in ascending order of how many times it appears in the string. If multiple letters appear the same number of times, print them in alphabetical order.

The string can contain up to **1,000,000 characters**, so the solution must run in **linear time**. If you sort the characters in the string, it will take at least **NLogN** time.

Sample 1:

Input: DACDACCDCBC

Output: BAADDDCCCCC

Sample 2:

Input: ZXZXYYZZYYXYYZZZWWQQQQVVVVVVVVVWXX

Output: WWWQQQQXXXXXYYYYYZZZZZZZVVVVVVVVV

Hint:

Use `vector<pair<int, int>> freq(26, pair<int, int>(0, 0));`

In first part of pair, store letter code, in second part store the frequency.

sort vector using following code:

```
sort(freq.begin(), freq.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
return a.second < b.second;
});
```

Print the letter as per frequency using the code in the first part of the pair.