Data Structures and Algorithms Lab

Lab 12 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

Managing Student Data with a Max Heap Structure

Implement a class for **Max Heap**. Each node of this Max Heap will contain the **roll number** and **CGPA** of a student. The heap will be organized based on students' CGPAs, i.e., the student having the **maximum CGPA** will be at the root of the heap.

```
/** Holds information about a student, including CGPA and roll number. */
class Student
{
	friend class StudentMaxHeap;

private:
	int rollNo;			/** Student's roll number */
	double cgpa;			/** Student's CGPA */
};

/** Manages an array of students arranged like a Max Heap. */
class StudentMaxHeap
{
private:
	Student* arr;		/** Array of students arranged like a Max Heap */
	int curSize;		/** Current number of students present in the heap */
	int maxSize;		/** Maximum number of students that can be present in the heap */

public:
	/** Constructor: Initializes the heap with the given maximum size. */
	StudentMaxHeap(int size);

	/** Destructor: Deallocates the memory used by the heap. */
	~StudentMaxHeap();

	bool isEmpty();			/** Checks whether the heap is empty */
	bool isFull();			/** Checks whether the heap is full */
};
```

Member Functions

Implement the following member functions of the **StudentMaxHeap** class:

1. `void insert(int rollNo, double cgpa)`
   - Inserts the record of a new student (with the given **roll number** and **CGPA**) into the Max Heap.
   - If two students have the same CGPA, the student with the **smaller roll number** should come first.
   - **Time Complexity**: $O(\log N)$
2. `bool remove(int& rollNo, double& cgpa)`
   - Removes the student with the **highest CGPA** from the Max Heap.
   - Stores the **roll number** and **CGPA** of the removed student in the provided arguments.
   - Returns **true** if the removal was successful, otherwise **false**.
   - **Time Complexity**: $O(\log N)$

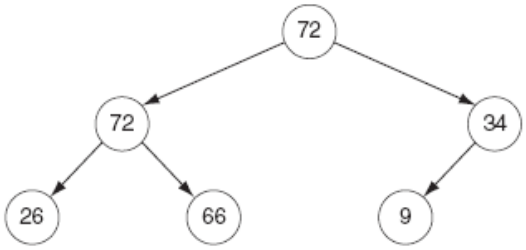
3. `void displayStudentList()`

- Displays the **roll numbers** and **CGPAs** of the students in **descending order** of **CGPA**.
- Does not alter the original heap.
- **Time Complexity**: $O(N \log N)$

4. `void levelOrder()`

- Performs a **level-order** traversal of the heap and displays the **roll numbers** and **CGPAs** of the students in that order.
  - The output displayed by the **level-order traversal** should follow the format shown below, in an abstract manner:

Heap:	**Level order traversal:**

	`72`
	`72 34`
	`26 66 9`

- **Time Complexity**: $O(N)$

Menu-Driven Testing

Write a menu-based driver function (**menu**) to test the functionality of the **StudentMaxHeap** class. The menu should look like this:

```
1. Insert a new student
2. Remove (and display) the student with the Max CGPA
3. Display the list of students (Descending order of CGPA)
4. Display the list of students (Level-order traversal)
5. Exit

Enter your choice:
```

Example Execution

- **Input**:
  - Insert students with the following details <**roll_number**, **CGPA**>:
    - (101, 3.9)
    - (102, 3.8)
    - (103, 3.9)
    - (104, 4.0)
- **Output:**

```
Level Order Traversal:

Roll No: 104, CGPA: 4.0
Roll No: 103, CGPA: 3.9
Roll No: 101, CGPA: 3.9
Roll No: 102, CGPA: 3.8

Descending Order of CGPAs:

Roll No: 104, CGPA: 4.0
Roll No: 101, CGPA: 3.9
Roll No: 103, CGPA: 3.9
Roll No: 102, CGPA: 3.8
```