Data Structures and Algorithms LAB – Spring 2022
(BS-IT-F20 Morning & Afternoon)

Lab # 12

Instructions:

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1 *(Max Time: 20 Minutes)*

You implemented the **`StudentBST`** class in the previous lab. Now, you are required to implement the following public member function of the **`StudentBST`** class:

`void displayInRange (int rollNoStart, int rollNoEnd)`

This function will search the BST for those students whose roll number is between **`rollNoStart`** and **`rollNoEnd`** (both inclusive). The records (all details) of all such students should be displayed in **ascending order** of their **Roll Numbers**.

- ***Hint*:** You may need to implement one or more **recursive** helper functions.
- ***Important Note:*** Your function should be **as efficient as possible** i.e. it should accomplish its objective in as few recursive calls as possible.

Also, modify the menu so that the user can see the list of students whose Roll Number lie within a specific user-given range.

Task # 2.1 *(Max Time: 10 Minutes)*

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 StudentMaxHeap;

class Student {
      friend class StudentMaxHeap;
  private:
      int rollNo;        // Student's roll number
      double cgpa;       // Student's CGPA
};

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 stored in the heap
  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
};
```

First of all, implement the **constructor**, **destructor**, **`isEmpty`** and **`isFull`** functions shown above in the class declaration.

Task # 2.2 *(Max Time: 10 Minutes)*

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:

```
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 $\boldsymbol{O(lg\ n)}$. 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 number(s).

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

Task # 2.3 *(Max Time: 10 Minutes)*

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:

```
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 reference parameters. 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 $\boldsymbol{O(lg\ n)}$.

Task # 2.4 *(Max Time: 10 Minutes)*

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

`void levelOrder ();`

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

`int height ();`

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

Task # 2.5 *(Max Time: 10 Minutes)*

Now, write a menu-based driver function to illustrate the working of different functions of the **`StudentMaxHeap`** class. The menu should look like:

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

Enter your choice:
```

Task # 2.6 *(Max Time: 15 Minutes)*

Implement the following two public member functions of the **`StudentMaxHeap`** class:

`void heapify (int i)`

This function will convert the subtree rooted at index **`i`** into a Max-Heap. This function will assume that the left-subtree and the right-subtree of index **`i`** are already valid Max-Heaps.

`void buildHeap (Student* st, int n)`

This function will take as array of students (**`st`**) and its size (**`n`**) as parameters. This function should build a Max-Heap containing the records of all **`n`** students using the algorithm **`buildHeap`** that we have discussed in lecture.

Task # 2.7 *(Max Time: 10 Minutes)*

Implement the following **global** function:

`void heapSort (Student* st, int n)` *// must be implemented as a global function*

This function will take as array of students (**`st`**) and its size (**`n`**) as parameters. This function should sort the array of students (**`st`**) into **increasing order** (according to **CGPA**), using the **Heap sort** algorithm that we have discussed in lecture. If two (or more) students have the same CGPA then their records should be sorted in a way that the record of the student having **smaller Roll number** should come before the record of the student having **larger Roll number**.

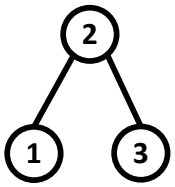
Also write a driver main function to test the working of the above function.

Task # 3 *(Max Time: 20 Minutes)*

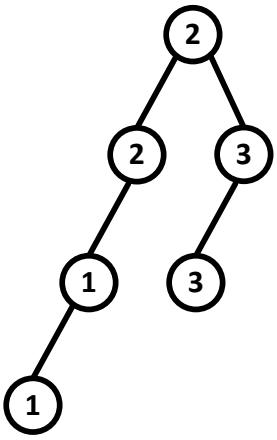
Implement a function named **`doubleTree`** (determine the exact function prototype yourself) of the **`BST`** class, which for each node in the BST, creates a new duplicate node, and inserts the duplicate as the left child of the original node. See the following example.

Example:

BST before calling **`doubleTree`** function:



BST after calling **`doubleTree`** function:



***Hint 1*:** Do some paperwork. Think recursively. You may implement a recursive helper function.

***Hint 2*:** Think about post-order.

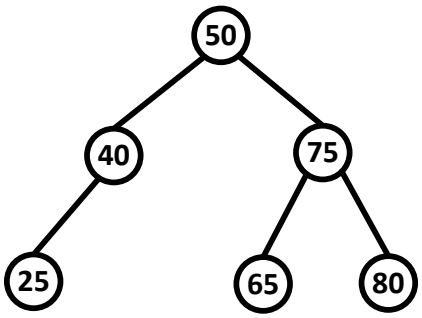
Note: When implementing the above function, you will assume that the left subtree of any node (in the resulting BST) can contain values which are **smaller than or equal to** the node's data.

Task # 4 *(Max Time: 30 Minutes)*

Implement a public member function of the **BST** class which prints **all the root-to-leaf paths** of a given BST. The prototype of your function will be:

```
void printAllPaths ()
```

For example, if we call this function on the following BST:



It should print the following paths (in this order):

50 -> 40 -> 25
50 -> 75 -> 65
50 -> 75 -> 80

***Hint 1*:** You will need to implement a recursive helper function. Think about the prototype of this recursive helper function.

***Hint 2*:** You need some way to communicate the list of paths from one function call to another.

***Hint 3*:** You can assume that the maximum length of a root-to-leaf path will be 100.

***Note*:** Do NOT use any global or static variables in your implementation.