**National University of Computer and Emerging Sciences**



**Laboratory Manual**

*for*

# Data Structures Lab

|  |  |
| --- | --- |
| Course Instructor | Mr. Uzair Naqvi |
| Lab Instructor | Mr. Durraiz Waseem |
| Semester | Fall 2024 |

# Department of Computer Science

FAST-NU, Lahore, Pakistan

**Objectives:**

In this lab, students will practice **Heap Trees**

**Problem 1: Max Heap  
  
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, 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 above.  
  
**2.** 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 (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.  
  
**3.** 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 (𝒍𝒈 𝒏).  
  
**4**. 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 Min-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 Min-Heap containing the records of all n students using the algorithm.

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 MaxHeap

int currSize; // Current number of students present in the heap

int maxSize; // Maximum no. of students that can be present in 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

};

**Problem 2**: **Min Heap**  
  
**1**. **Find the k Smallest Elements in a Min-Heap:** Given a min-heap, write a function to find the k smallest elements without modifying the heap.   
Example:  
**Input**: heap = [1, 3, 6, 5, 9, 8], k = 3  
**Expected Output**: [1, 3, 5]

**2. Merging Two Min-Heaps:** Given two min-heaps, implement a function to merge them into a single min-heap.  
Example:  
**Input**: heap1 = [1, 3, 6], heap2 = [2, 5, 7]  
**Expected Output**: merged\_heap = [1, 2, 3, 5, 6, 7]

**3. Convert Min-Heap to Max-Heap**: Given a min-heap, write a function to convert it to a max-heap without changing the structure.  
Example:  
**Input**: min\_heap = [1, 3, 6, 5, 9, 8]  
**Expected Output** (max-heap): [ 9, 5, 8, 1, 3, 6]