Lab 9 (Binary Search Trees and Heaps)

CSC 172 (Data Structures and Algorithms)
Spring 2018
University of Rochester
Due Date: Sunday, April 15 @ 11:59 pm

Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Every student must hand in their own work but also list the name of their lab partner (if any) on all labs.

Objectives

This lab is designed to give you a chance to explore in details implementation details of Binary Search Trees and Heaps—two distinct but crucial variations of binary trees. This lab is closely related to Workshop #9.

Problem 1: Binary Search Tree

Assume the following BSTNode class is given.

```
public class BSTNode
{
  int key;
  BSTNode parent;
  BSTNode left;
  BSTNode right;

  // Add constrcutor and other methods if required.

}
```

1. (5 pts) Create a new class `BSTree` which represents a Binary Search tree (consist of instances of `BSTNode`).
2. (45 pts) Implement the following three functions:

```
// 10 pts
// Insert a key.
// If the key is already present, return false, otherwise return true
 boolean insert(int key)

// 25 pts
// Remove the key if present and return true; otherwise, return false;
// (Consider cases where the node to be deleted has 0, 1, or 2 children)
boolean remove (int key)

// 10 pts
// Search for a specific key. If found, return true; otherwise, return false;
boolean search(int key)
```

3. (10 pts) Create another class `BSTreeTest` which tests the functionality of these three functions. Steps to be followed:
   (a) Create an empty Binary Search Tree (BST) and then insert 5, 18, 3, 25, 27, 45, 97, 88, 26, 15, 17, 16 into that BST.
   (b) Perform an inorder traversal of the BST and print each key. The output should be in sorted order.
   (c) Search for 3, 88, 27, and 28. Print if each search was successful
   (d) Now delete the following entries from the BST (in order): 88, 18, 5, 30. Print 'Item not found' if a particular key was not found for deletion.
   (e) Print the inorder traversal after each deletion

Note: For getting full credits for part 2, your part 3 must work properly.

Problem 2: Heap

1. Create a new class `Heap` This class must implement the following two methods:
   (a) (15 pts) `heapify` which takes an array of integers as input and converts it into a max heap. (i.e., implement Heapify algorithm). Your algorithm must be in-place.
   Method signature:
   ```
   void heapify(int[] arr);
   ```
   (b) (15 pts) `heapsort` which takes an array and sorts it (ascending order)
   Method signature:
   ```
   void heapsort(int[] arr);
   ```
   For heap sort, You can/should call heapify method internally
2. (10 pts) Create another class `HeapTest` for testing You must follow the following steps:
   (a) Create an array `arr` containing the following elements:
   5, 18, 3, 25, 27, 45, 97, 88, 26, 16, 49, 67
   (b) Call `heapify(arr)`
   (c) Print elements in arr
   (d) Create another array `arr2` containing the following elements:
   15, 99, 3, 77, 27, 45, 7, 88, 26, 5
   (e) call `heapsort(arr2)`
   (f) Print elements in `arr2`

Note: For getting full credits for part 1, your part 2 must work correctly.

Submission

Create a zip file Lab9.zip containing your source code and a README file. Submit this file at the appropriate location on the Blackboard system at `learn.rochester.edu`. The README file should state your and your team member's name and any other pertinent information. You should include five Java files in your submission:

- BSTNode.java
- BSTree.java

- BSTreeTest.java
- Heap.java
- HeapTest.java

Grading **(100 pts)**

Problem 1: 60 pts
Problem 2: 40 pts
Not following the direction would cause 10 pts deduction.