Lab 12: Binary Search Tree and Heap

Question 1

Create a package called BST and implement a node class called `TreeNode<E>` and Binary Search Tree Class called `BST<E>`. Both the `BST<E>` and `TreeNode<E>` classes extends Comparable.

a) Include necessary declaration in the `BST<E>` and `TreeNode<E>` classes.

b) Implement the following methods in class BST<E>:

i. `public boolean search(E e)`
Returns true if the element is in the tree

ii. `public boolean insert(E e)`
Insert element o into the binary tree and return true if the element is inserted successfully

iii. `public int getSize()`
Get the number of nodes in the tree

iv. `public int height()` and `private int height(TreeNode<E> node)`
Returns the height of the BST

v. `public E getRoot()`
Returns the root of the BST

vi. `public E minValue()`
Returns the minimum value of the BST

vii. `public E maxValue()`
Returns the maximum value of the BST

viii. `public java.util.ArrayList<TreeNode<E>> path(E e)`
Returns a path from the root leading to the specified element

ix. `public boolean delete(E e)`
Delete an element from the binary tree. Return true if the element is deleted successfully, and return false if the element is not in the tree

x. `public boolean clear()`
Remove all elements from the tree

xi. `protected void inorder(TreeNode<E> root)`
Display inorder traversal from a subtree

xii. `protected void postorder(TreeNode<E> root)`
Display postorder traversal from a subtree

xiii. `protected void preorder(TreeNode<E> root)`
Display preorder traversal from a subtree

c) Write a test program called `TestBST` in the BST package. Using the appropriate methods you implemented in `BST<E>`, produce the following output:

```
Input Data: 45, 88, 54, 76, 98, 1, 2, 20, 6, 53, 42, 100, 86, 32, 28, 65, 14
Inorder (sorted): 1 2 6 14 20 28 32 42 45 53 54 65 76 86 88 98 100
Postorder: 14 6 28 32 42 20 2 1 53 65 86 76 54 100 98 88 45
Preorder: 45 1 2 20 6 14 42 32 28 88 54 53 76 65 86 98 100
Height of BST: 6
Root for BST is: 45
Check whether 10 is in the tree? false
Delete 53
Updated Inorder data (sorted): 1 2 6 14 20 28 32 42 45 54 65 76 86 88 98 100
Min Value :1
Max Value :100
A path from the root to 6 is: 45 1 2 20 6
```

Question 2

Convert the following maxHeap code to minHeap code. Test your minHeap code.

```
public class Heap<E extends Comparable<E>> {
  private java.util.ArrayList<E> list = new java.util.ArrayList<E>();

  /** Create a default heap */
  public Heap() {
  }

  /** Create a heap from an array of objects */
  public Heap(E[] objects) {
    for (int i = 0; i < objects.length; i++)
      add(objects[i]);
  }

  /** Add a new object into the heap */
  public void add(E newObject) {
    list.add(newObject); // Append to the heap
    int currentIndex = list.size() - 1; // The index of the last node

    while (currentIndex > 0) {
      int parentIndex = (currentIndex - 1) / 2;
      // Swap if the current object is greater than its parent
```

```
      if (list.get(currentIndex).compareTo(
          list.get(parentIndex)) > 0) {
        E temp = list.get(currentIndex);
        list.set(currentIndex, list.get(parentIndex));
        list.set(parentIndex, temp);
      }
      else
        break; // the tree is a heap now

      currentIndex = parentIndex;
    }
  }
  /** Remove the root from the heap */
  public E remove() {
    if (list.size() == 0) return null;

    E removedObject = list.get(0);
    list.set(0, list.get(list.size() - 1));
    list.remove(list.size() - 1);

    int currentIndex = 0;
    while (currentIndex < list.size()) {
      int leftChildIndex = 2 * currentIndex + 1;
      int rightChildIndex = 2 * currentIndex + 2;

      // Find the maximum between two children
      if (leftChildIndex >= list.size()) break; // The tree is a heap
      int maxIndex = leftChildIndex;
      if (rightChildIndex < list.size()) {
        if (list.get(maxIndex).compareTo(
            list.get(rightChildIndex)) < 0) {
          maxIndex = rightChildIndex;
        }
      }
      // Swap if the current node is less than the maximum
      if (list.get(currentIndex).compareTo(
          list.get(maxIndex)) < 0) {
        E temp = list.get(maxIndex);
        list.set(maxIndex, list.get(currentIndex));
        list.set(currentIndex, temp);
        currentIndex = maxIndex;
      }
      else
        break; // The tree is a heap
    }
    return removedObject;
  }

  /** Get the number of nodes in the tree */
  public int getSize() {
    return list.size();
  }
}
```