CMP-338 110 Points Total Final Spring 2017

1. (20 Points) Multiple Choice:

A. (2 Points) A _____ is an undirected connected graph without cycles.

   a. tree
   b. multigraph
   c. digraph
   d. connected component

B. (2 Points) A connected undirected graph that has n vertices and exactly n – 1 edges _____.

   a. cannot contain a cycle
   b. must contain at least one cycle
   c. can contain at most two cycles
   d. must contain at least two cycles

C. (2 Points) The sum of the weights of the edges of a path can be called all of the following EXCEPT _____.

   a. length
   b. weight
   c. height
   d. cost

D. (2 Points) Each node in a tree has _____.

   a. exactly one parent
   b. at most one parent
   c. exactly two parents
   d. at most two parents

E. (2 Points) A full binary tree with height 4 has _____ nodes.

   a. 7
   b. 8
   c. 15
   d. 31

F. (2 Points) _____ is the ability of a class to derive properties from a previously defined class.

   a. Encapsulation
   b. Simulation
   c. Inheritance
   d. Polymorphism

G. (2 Points) In an implementation of a queue uses the ADT list, which of the following can be used to implement the operation `enqueue(newItem)`?

   a. `list.add(list.size(), newItem)`
   b. `list.add(list.size()+1, newItem)`
   c. `list.add(newItem.size(), newItem)`
   d. `list.add(newItem.size()+1, newItem)`

H. (2 Points) In the ADT list, items can be added _____.

   a. only at the front of the list
   b. only at the back of the list
   c. either at the front or the back of the list
   d. at any position in the list

I. (2 Points) The last-in, first-out (LIFO) property is found in the ADT _____.

   a. list
   b. stack
   c. queue
   d. tree

J. (2 Points) Which of the following statements is used to insert a new node, referenced by newNode, at the end of a linear linked list?

```
a. newNode.setNext(curr);
   prev.setNext(newNode);
b. newNode.setNext(head);
   head = newNode;
c. prev.setNext(newNode);
d. prev.setNext(curr);
   newNode.setNext(curr);
```

2. (20 Points) The corrected HeapSort Class:

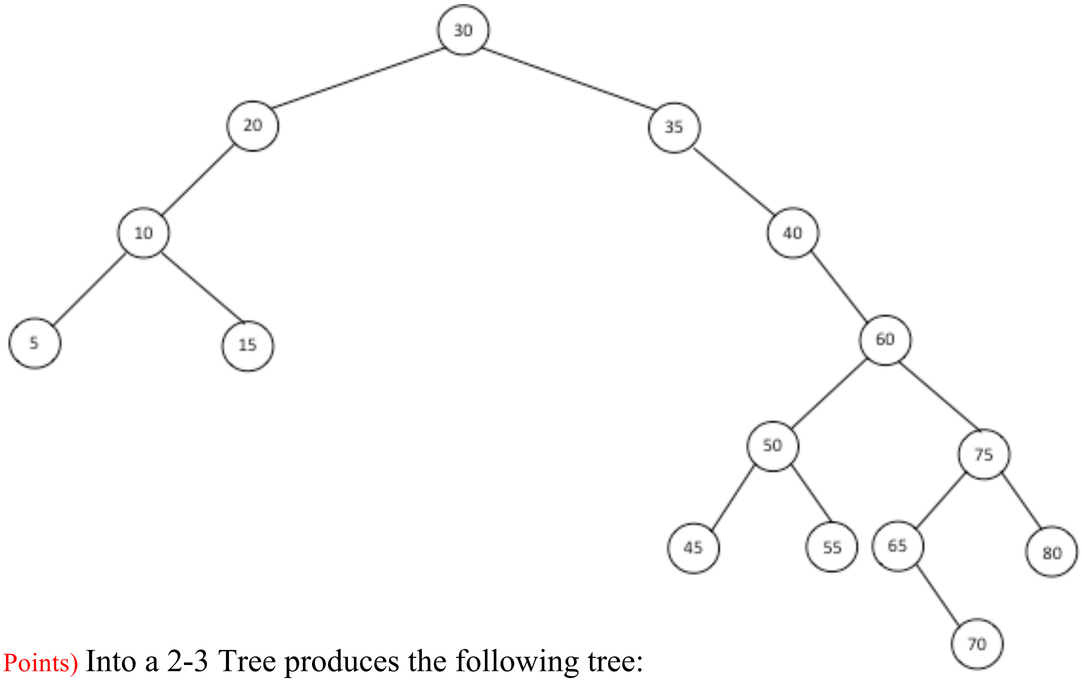
```
public class HeapSort<T extends Comparable<? super T>> {
   T heap[];
   int heapSize;

   public void sort(T[] arrayToSort) {
       this.heap = arrayToSort;
      this.heapSize = this.heap.length;
      this.heapify();

      heapSort();
   }

   private void heapSort() {
      while (this.heapSize >= 1) {
         T temp = this.heap[0];
         this.heap[0] = this.heap[this.heapSize - 1];
         this.heap[this.heapSize - 1] = temp;
         this.heapSize--;
         heapify();
      }
   }

   private void heapify() {
      int last = this.heapSize - 1;
      int parent = (last - 1) / 2;

      while (parent >= 0) {
         siftDown(parent);
         parent = parent - 1;
      }
   }

   private void siftDown(int node) {

      while (node < this.heapSize) {
         int leftChild = (2 * node) + 1;
         int rightChild = (2 * node) + 2;
         int swap = node;

         if ((leftChild < this.heapSize) && (this.heap[node].compareTo(this.heap[leftChild]) < 0)) {
            swap = leftChild;
         }

         if ((rightChild < this.heapSize) && (this.heap[swap].compareTo(this.heap[rightChild]) < 0)) {
            swap = rightChild;
         }

         if (swap == node) {
            return;
         } else {
            T temp = this.heap[node];
            this.heap[node] = this.heap[swap];
            this.heap[swap] = temp;
            node = swap;
         }
      }
   }
}
```

3. (50 Points) The correct BinarySearchTree implementation:

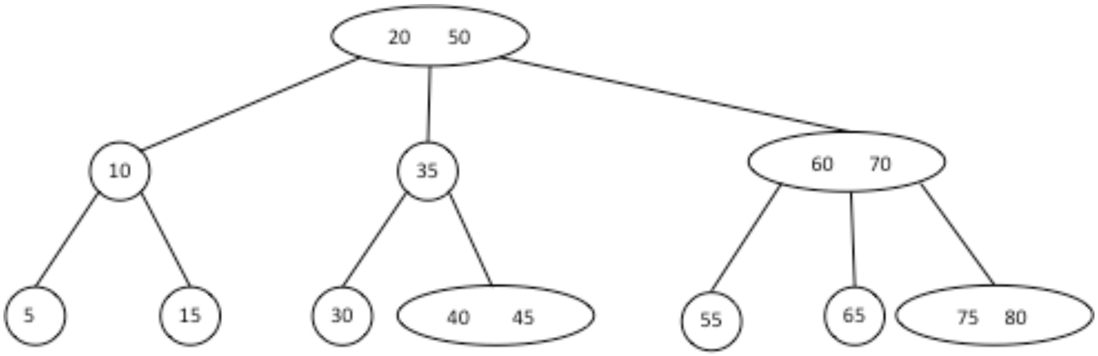
```java
public class BinarySearchTree
               <K extends Comparable<? super K>, V>
               implements BinarySearchTreeInterface<K,V> {

   private TreeNode<K,V> root = null;

   @Override
   public TreeNode<K, V> getRoot() {
      return root;
   }

   @Override
   public void setRoot(TreeNode<K, V> root) {
      this.root = root;
   }

   @Override
   public TreeItem<K,V> getRootItem() {
      if (this.root == null) {
         return null;
      } else {
         return this.root.getTreeItem();
      }
   }

   @Override
   public boolean isEmpty() {
      return (root == null);
   }

   @Override
   public void makeEmpty() {
      this.root = null;
   }

   @Override
   public TreeItem<K,V> find(K key) {
      return findItem(this.root, key);
   }

   private TreeItem<K,V> findItem(TreeNode<K,V> node, K key) {
      if (node == null) {
         return null;
      } else if (node.getTreeItem().getKey().compareTo(key) == 0) {
         return node.getTreeItem();
      } else if (node.getTreeItem().getKey().compareTo(key) > 0) {
         return findItem(node.getLeftChild(), key);
      } else {
         return findItem(node.getRightChild(), key);
      }
   }

   @Override
   public void insert(TreeItem<K,V> treeItem) {
      this.root = insertItem(this.root, null, treeItem);
   }

   private TreeNode<K,V> insertItem(TreeNode<K,V> node,
                                    TreeNode<K,V> parent,
                                    TreeItem<K,V> treeItem) {
      if (node == null) {
         node = new TreeNode<K,V> (treeItem);
         node.setParent(parent);
      } else if (node.getTreeItem().getKey().compareTo(treeItem.getKey()) > 0) {
         node.setLeftChild(this.insertItem(node.getLeftChild(), node, treeItem));
      } else {
         node.setRightChild(this.insertItem(node.getRightChild(), node, treeItem));
      }
      return node;
   }

   @Override
   public int height() {
      return treeHeight(this.root);
   }

   private int treeHeight(TreeNode<K,V> node) {
      int height = 0;

      if (node != null) {
         int lHeight = treeHeight(node.getLeftChild());
         int rHeight = treeHeight(node.getRightChild());
         height = Math.max(lHeight, rHeight) + 1;
      }
      return height;
   }

   @Override
   public boolean isBalanced() {
      return isBalancedSubtree(this.getRoot());
   }

   private boolean isBalancedSubtree(TreeNode<K,V> node) {
      boolean answer = true;
      int lHeight, rHeight;

      if (node == null) {
         return answer;
      }

      lHeight = treeHeight(node.getLeftChild());
      rHeight = treeHeight(node.getRightChild());

      answer = (Math.abs(lHeight - rHeight) <= 1) &&
            isBalancedSubtree(node.getLeftChild()) &&
            isBalancedSubtree(node.getRightChild());

      return answer;
   }
}
```

4. (20 Points) The following list of numbers: 40, 30, 55, 20, 10, 50, 70, 65, 5, 15, 4, 60, 85, 54, 8 inserted in the given order:

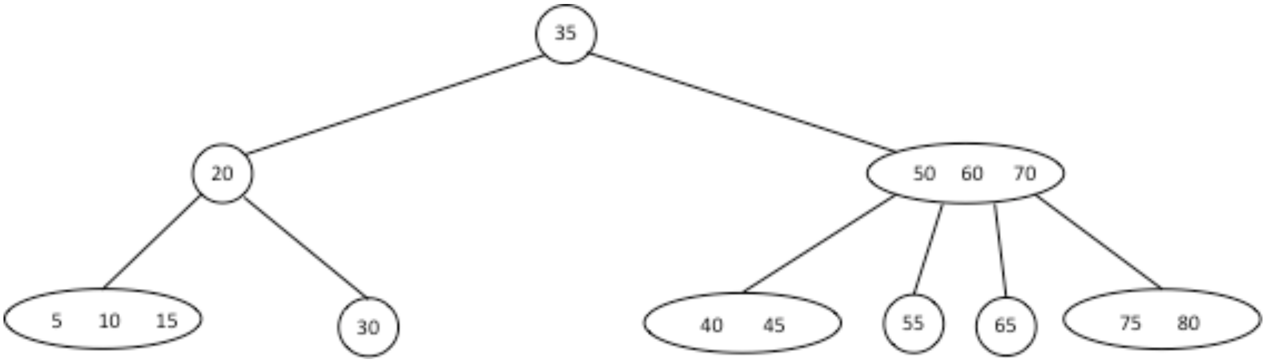
   a. (5 Points) Into a Binary Search Tree produces the following tree:



   b. (5 Points) Into a 2-3 Tree produces the following tree:



   c. (5 Points) Into a 2-3-4 Tree produces the following tree:



   d. (5 Points) The numbers in the following order produces a full Binary Search Tree:

45, 20, 10, 5, 15, 35, 30, 40, 65, 55, 50, 60, 75, 70, 80