Instructions

- **Answer all questions on the yellow paper.**
- **One question per page.**
- **Use only one side of the yellow paper.**

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) Re-write the following HeapSort class and fix all 10 **logical** errors:

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

    public void sort(T[] arrayToSort) {
        this.heap = arrayToSort;
       this.heapSize = 1;
       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);

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

   private void siftDown(int node) {

       while (node < this.heapSize) {
          int leftChild = (2 * node);
          int rightChild = (2 * node) + 1;
          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) Given the following BinarySearchTreeInterface, TreeItem, and TreeNode implementations. Write the complete Java class for the BinarySearchTree which implements the given BinarySearchTreeInterface.

```
public interface BinarySearchTreeInterface
              <K extends Comparable<? super K>, V> {

   // returns the root node of the binary search tree
   public TreeNode<K, V> getRoot();

   // set the root node of the binary search tree
   public void setRoot(TreeNode<K, V> root);

   // returns the TreeItem that is in the root node or
   // null if the binary search tree is empty
   public TreeItem<K,V> getRootItem();

   // return true if the binary search tree
   // is empty, false otherwise
   public boolean isEmpty();

   // makes the tree empty
   public void makeEmpty();

   // find and returns the TreeItem with the given key.
   // Returns null if key is not found
   public TreeItem<K,V> find(K key);

   // insert the given TreeItem into the
   // binary search tree.
   public void insert(TreeItem<K,V> treeItem);

   // compute and return the height of the
   // binary search tree
   public int height();

   // returns true if the binary search tree is
   // balanced, false otherwise
   public boolean isBalanced();
}
```

```
public class TreeItem<K extends
                      Comparable <? super K>, V> {

   private K key;
   private V value;

   public TreeItem(K key, V value) {
      this.key = key;
      this.value = value;
   }

   public K getKey() {
      return key;
   }

   public V getValue() {
      return value;
   }

   public void setValue(V value) {
      this.value = value;
   }
}
```

```
public class TreeNode<K extends
                      Comparable <? super K>, V> {

   private TreeItem<K,V> treeItem;
   private TreeNode<K,V> leftChild;
   private TreeNode<K,V> rightChild;
   private TreeNode<K,V> parent;

   public TreeNode(TreeItem<K,V> treeItem) {
      this.treeItem = treeItem;
      this.leftChild = null;
      this.rightChild = null;
      this.parent = null;
   }

   public TreeNode<K, V> getLeftChild() {
      return leftChild;
   }

   public void setLeftChild(TreeNode<K, V> leftChild) {
      this.leftChild = leftChild;
   }

   public TreeNode<K, V> getRightChild() {
      return rightChild;
   }

   public void setRightChild(TreeNode<K, V> rightChild) {
      this.rightChild = rightChild;
   }

   public TreeNode<K, V> getParent() {bv
      return parent;
   }

   public void setParent(TreeNode<K, V> parent) {
      this.parent = parent;
   }

   public TreeItem<K, V> getTreeItem() {
      return treeItem;
   }

   public void setTreeItem(TreeItem<K, V> treeItem) {
      this.treeItem = treeItem;
   }
}
```

4. (20 Points) Given the following list of numbers: 30, 20, 10, 15, 5, 35, 40, 60, 50, 45, 55, 75, 65, 70, 80 being inserted in the given order.
    a. (5 Points) Draw the resulting Binary Search Tree.
    b. (5 Points) Draw the resulting 2-3 Tree.
    c. (5 Points) Draw the resulting 2-3-4 Tree.
    d. (5 Points) What order should the numbers be inserted in order to obtain a Full Binary Search Tree?