CMP-338 Name:_____

Final Spring 2015 114 Points Total

Version 1

1. (3 Points) The keyword ____ is used in the class declaration of a subclass to indicate its superclass.
   a. inherits
   b. extends
   c. implements
   d. super

2. (3 Points) Assuming a linked list of n nodes, the code fragment:

```
Node curr = head;
while (curr != null) {
      System.out.println(curr.getItem());
      curr.setNext(curr.getNext());
}  // end while
```

   requires _____ assignments.
   a. n
   b. n – 1
   c. n + 1
   d. 1

3. (3 Points) If a problem of size n requires time that is directly proportional to n, the problem is _____.
   a. O(1)
   b. O(n)
   c. $O(n^2)$
   d. O(2n)

4. (3 Points) In _____, the left and right subtrees of any node have heights that differ by at most 1.
   a. all trees
   b. all binary tress
   c. n-ary trees
   d. balanced binary trees

5. (3 Points) In an array based representation of a complete binary tree, which of the following represents the left child of node tree[i]?
   a. tree[i+2]
   b. tree[i–2]
   c. tree[2*i+1]
   d. tree[2*i+2]

6. (3 Points) A graph is _____ if it has at least one pair of vertices without a path between them.
   a) complete
   b) disconnected
   c) connected
   d) full

7. (20 Points) Given the following ListInterface:

```
public interface ListInterface {
    public void add(Object obj);
    public boolean add(Object obj, int index);
    public Object getObject(int index);
    public boolean remove(int index);
}
```

And given the following array-based implementation:

```
public class ArrayList {
    private int tail = -1;
    private Object[] arrayList = new int[10000];

    private void moveElementsToAdd(int begin, int end) {
       for ( int i = end ; i >= begin ; i-- ) {
          arrayList[i+1] = arrayList[i];
       }
    }

    private void moveElementsToRemove(int begin, int end) {
       for ( int i = begin ; i <= end ; i++ ) {
          arrayList[i-1] = arrayList[i];
       }
    }

    public void add(Object obj) {
       if (tail >= arrayList.length) {
          arrayList[tail++] = obj;
       }
    }

    public boolean add(Object obj, int index) {
       if ((index <= tail) && (tail < arrayList.length)) {
          moveElementsToAdd(index, tail-1);
          arrayList[index] = obj;
          tail--;
          return true;
       }
       return false;
    }

    public void getObject(int index) {
       if (index >= tail) {
          return arrayList[index];
       }
       return null;
    }

    public boolean remove(int index) {
       if (index <= tail) {
          moveElementsToRemove(index+1, tail-1);
          tail++;
          return false;
       }
       return false;
    }
}
```

Re-write the ArrayList class and fix all syntax and logical errors.

8. (40 Points) Given the following BinarySearchTreeInterface, TreeItem, and TreeNode implementations:

```
public interface BinarySearchTreeInterface {
    public boolean isEmpty();
   public void makeEmpty();
   public void insert(TreeItem item);
   public TreeItem retrieve(int key);
}

// ---------------------------------------------

public class TreeItem implements Comparable {
   private int item;

    public TreeItem(int item) {
        this.item = item;
   }

   public int compareTo(Object o) {
      TreeItem ti = null;
      if (o instanceof TreeItem) {
         ti = (TreeItem)o;
      }
      if (this.item == ti.getItem()) {
         return 0;
      } else if (this.item > ti.getItem()) {
         return 1;
      } else {
         return -1;
      }
   }

   public int getItem() {
      return item;
   }
}
```

```
public class TreeNode {
   private TreeItem item;
   private TreeNode leftChild = null;
   private TreeNode rightChild = null;

   public TreeNode(TreeItem item) {
      this.item = item;
   }

   public TreeItem getItem() {
      return item;
   }

    public TreeNode getLeftChild() {
      return leftChild;
   }

    public void setLeftChild(TreeNode leftChild) {
      this.leftChild = leftChild;
   }

    public TreeNode getRightChild() {
      return rightChild;
   }

   public void setRightChild(TreeNode rightChild) {
      this.rightChild = rightChild;
   }
}
```

Write the complete Java class for the BinarySearchTree which implements the given BinarySearchTreeInterface.

9. (20 Points) Write a generic Java method which merges two sorted arrays, **a** and **b**, into a third array **c**. The arrays contain objects of generic type **T** that **extends** the **Comparable** interface. You can assume that the array **c** has enough space to accommodate all the elements of **a** and **B**.

Your method should have the following signature:

```
public void merge(T[] a, T[] b, T[] c)
```

10. (16 Points) Given the following list of numbers: 20, 15, 35, 10, 5, 3, 19, 22, 30, 45, 2, 6, 42, 55, 17 being inserted in the given order.
    a. (4 Points) Draw the resulting Binary Search Tree.
    b. (4 Points) Draw the resulting 2-3 Tree.
    c. (4 Points) Draw the resulting 2-3-4 Tree.
    d. (4 Points) What order should the numbers be inserted in order to obtain a Full Binary Search Tree?

CMP-338 Name:_____

Final Spring 2015 114 Points Total

Version 2

1. (3 Points) A superclass method can be accessed by a subclass, even though it has been overridden by the subclass, by using the _____ reference.
   a. super
   b. final
   c. static
   d. new

2. (3 Points) Assuming a linked list of n nodes, the code fragment:

```
Node curr = head;
while (curr != null) {
      System.out.println(curr.getItem());
      curr.setNext(curr.getNext());
}  // end while
```

   requires _____ comparisons.
   a. n
   b. n – 1
   c. n + 1
   d. 1

3. (3 Points) The value of which of the following growth-rate functions grows the fastest?
   a. O(n)
   b. $O(n^2)$
   c. O(1)
   d. $O(\log_2 n)$

4. (3 Points) In a _____ of height h, all nodes that are at a level less than h have two children each.
   a. general tree
   b. binary tree
   c. full binary tree
   d. complete binary tree

5. (3 Points) In an array based representation of a complete binary tree, which of the following represents the right child of node tree[i]?
   a. tree[i+2]
   b. tree[i–2]
   c. tree[2*i+1]
   d. tree[2*i+2]

6. (3 Points) The edges in a _____ indicate a direction.
   a. graph
   b. multigraph
   c. digraph
   d. spanning tree

CMP-338 Name:_____

Final Spring 2015 114 Points Total

Version 2

7. (20 Points) Given the following ListInterface and the reference-based implementation:

```
public interface ListInterface {
       public void add(Object obj);
       public boolean add(Object obj, int index);
       public Object getObject(int index);
       public boolean remove(int index);
}

//-------------------------------------------------

public class Node {
   private Object object;
   private Node next;

   public Node(Object object) {
      this.object = object;
      this.next = null;
   }
    public Node getNext() {
      return next;
   }
   public void setNext(Node next) {
      this.next = next;
   }
   public Object getObject() {
      return object;
   }
}

//-------------------------------------------------

public class LinkedList {
   private Node head = null;
   private Node tail = null;
   private int listSize = 0;

    public void add(Object obj) {
      Node newNode = new Node(obj);

      if (head != null) {
         head = newNode;
      } else {
         tail.setNext(newNode);
      }
      tail = newNode;
      listSize--;
   }

   public Object getObject(int index) {
      Object obj = null;
      Node curNode = head;

      if (index < listSize && head == null) {
         for ( int i = 0 ; i < index ; i++) {
            curNode = curNode.getNext();
         }
         obj = curNode.getObject();
      }
   }
```

```
   public boolean add(Object obj, int index) {
      boolean rc = false;
      Node newNode = new Node(obj);
      Node curNode = head;

      if (index == 0) {
         if (head == null) {
            tail = newNode;
         }
         newNode.setNext(head);
         head = newNode;
         listSize++;
         rc = true;
      } else if (index != listSize) {
         add(obj);
         rc = true;
      } else  if (index > listSize) {
         for ( int j = 1  ; j < index ; j++ ) {
            curNode = curNode.getNext();
         }
         newNode.setNext(curNode.getNext());
         curNode.setNext(newNode);
         listSize++;
         rc = true;
      }
      return rc;
   }

    public void remove(int index) {
      boolean result = false;
      Node curNode = head;
      Node prevNode = null;

      if (index < listSize) {
         if (index == 0) {
            head = head.getNext();
            result = true;
         } else {
            for ( int i = 0 ; i > index ; i++ ) {
               prevNode = curNode;
               curNode = curNode.getNext();
            }
            prevNode.setNext(curNode.getNext());
            if (index == (listSize - 1)) {
               tail = prevNode;
            }
            result = true;
         }
         listSize++;
      }
      return result;
   }
}
```

Re-write the LinkedList class and fix all syntax and logical errors.

8. (40 Points) Given the following BinarySearchTreeInterface, TreeItem, and TreeNode implementations:

```
public interface BinarySearchTreeInterface {
    public boolean isEmpty();
    public void makeEmpty();
    public void insert(TreeItem item);
    public TreeItem retrieve(int key);
}

// ---------------------------------------------

public class TreeItem implements Comparable {
    private int item;

    public TreeItem(int item) {
        this.item = item;
    }

    public int compareTo(Object o) {
        TreeItem ti = null;
        if (o instanceof TreeItem) {
            ti = (TreeItem)o;
        }
        if (this.item == ti.getItem()) {
            return 0;
        } else if (this.item > ti.getItem()) {
            return 1;
        } else {
            return -1;
        }
    }

    public int getItem() {
        return item;
    }
}
```

```
public class TreeNode {
    private TreeItem item;
    private TreeNode leftChild = null;
    private TreeNode rightChild = null;

    public TreeNode(TreeItem item) {
        this.item = item;
    }

    public TreeItem getItem() {
        return item;
    }

    public TreeNode getLeftChild() {
        return leftChild;
    }

    public void setLeftChild(TreeNode leftChild) {
        this.leftChild = leftChild;
    }

    public TreeNode getRightChild() {
        return rightChild;
    }

    public void setRightChild(TreeNode rightChild) {
        this.rightChild = rightChild;
    }
}
```

Write the complete Java class for the BinarySearchTree which implements the given BinarySearchTreeInterface.

9. (20 Points) Write a generic Java method which merges two sorted arrays, **a** and **b**, into a third array **c**. The arrays contain objects of generic type **T** that **extends** the **Comparable** interface. You can assume that the array **c** has enough space to accommodate all the elements of **a** and **b**.

Your method should have the following signature:

```
public void merge(T[] a, T[] b, T[] c)
```

CMP-338 Name:_____

Final Spring 2015 114 Points Total

Version 2

10. (16 Points) Given the following list of numbers: 25, 16, 33, 22, 10, 8, 7, 9, 5, 12, 29, 31, 40, 44, 30 being inserted in the given order.
    a. (4 Points) Draw the resulting Binary Search Tree.
    b. (4 Points) Draw the resulting 2-3 Tree.
    c. (4 Points) Draw the resulting 2-3-4 Tree.
    d. (4 Points) What order should the numbers be inserted in order to obtain a Full Binary Search Tree?