CMP-338 Name:_____

Midterm Spring 2015 Solutions

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);
}
```

The correct array-based implementation:

```
public class ArrayList implements ListInterface {
    private int tail = 0;
    private Object[] arrayList = new Object[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 Object 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 true;
       }
       return false;
    }
}
```

8. (40 Points) The correct BinarySearchTree implementation is:

```
public class BinarySearchTree implements
BinarySearchTreeInterface {
   private TreeNode root = null;

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

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

   @Override
   public void insert(TreeItem item) {
      root = insertItem(root, item);
   }

   private TreeNode insertItem(TreeNode node,
                               TreeItem newItem) {
      TreeNode newSubtree;

      if (node == null) {
         node = new TreeNode(newItem);
         return node;
      }

      if (newItem.compareTo(node.getItem()) < 0) {
         newSubtree =
insertItem(node.getLeftChild(), newItem);
         node.setLeftChild(newSubtree);
         return node;
      } else {
         newSubtree =
insertItem(node.getRightChild(), newItem);
         node.setRightChild(newSubtree);
         return node;
      }
   }

   @Override
   public TreeItem retrieve(int key) {
      return retrieveItem(root, key);
   }

   private TreeItem retrieveItem(TreeNode node,
                                 int key) {
      TreeItem treeItem;
      TreeItem compareItem = new TreeItem(key);
      if (node == null) {
         treeItem = null;
      } else if
(compareItem.compareTo(node.getItem()) == 0) {
         treeItem = node.getItem();
      } else if
(compareItem.compareTo(node.getItem()) < 0) {
         treeItem =
retrieveItem(node.getLeftChild(), key);
      } else {
         treeItem =
retrieveItem(node.getRightChild(), key);
      }
      return treeItem;
   }
}
```

CMP-338 Name:_____

Midterm Spring 2015 Solutions

Version 1

9. (20 Points) The correct method for merging two arrays:

```
public void merge(T[]a, T[]b, T[]c) {
        int firstA = 0;
        int lastA  = a.length - 1;
        int firstB  = 0;
        int lastB  = b.length - 1;

        int cIindex = 0;

        while ((firstA <= lastA)  && (firstB  <= lastB)) {
                if (a[firstA].compareTo(b[firstB]) < 0)  {
                        c[cIindex]  = a[firstA++];
                } else {
                        c[cIindex]  = b[firstB++];
                }
                cIindex++;
        }   // end while

        while (firstA <= lastA)  {
                c[cIindex++] = a[firstA++] ;
        }   // end while


        while (firstB <= lastB)  {
                c[cIindex++] = b[firstB++];
        }   // end while
}
```

CMP-338 Name:_____

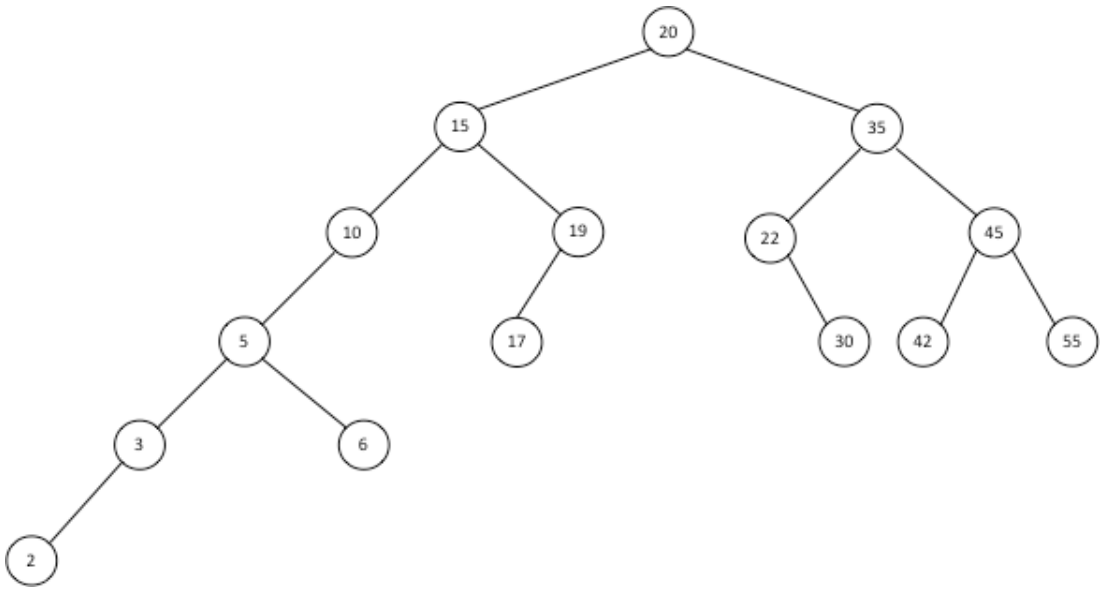
Midterm Spring 2015

Solutions

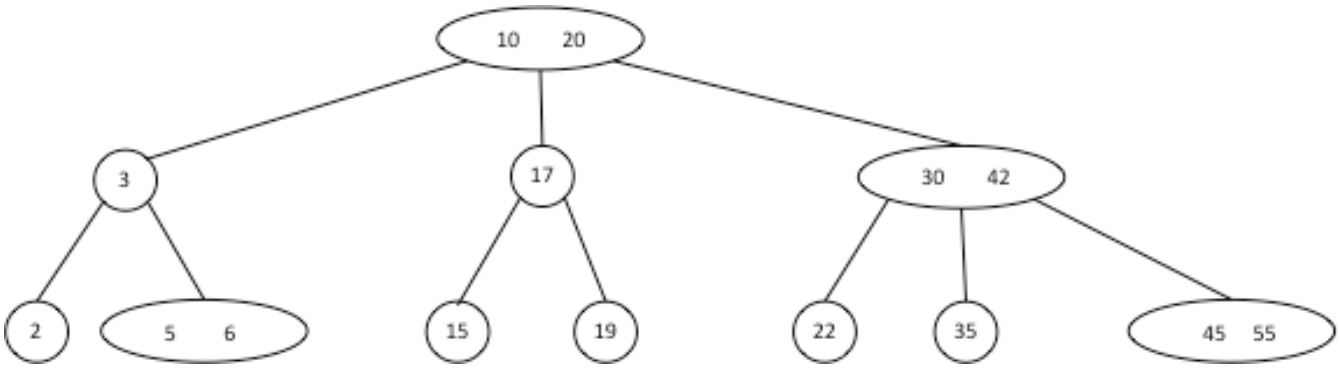
Version 1

10. (16 Points) The following list of numbers: 20, 15, 35, 10, 5, 3, 19, 22, 30, 45, 2, 6, 42, 55, 17 inserted in the given order into:

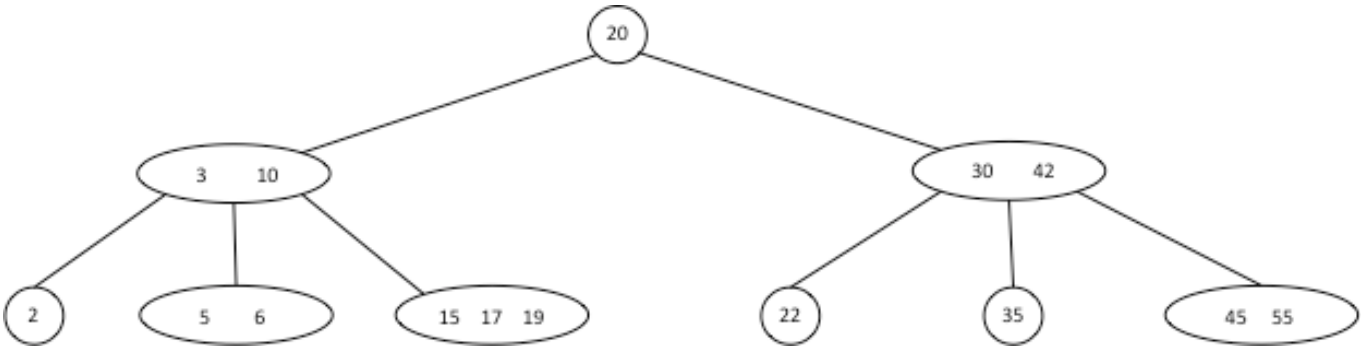
a. (4 Points) Into a Binary Search Tree produces the following tree:



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



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



d. (4 Points) The numbers in the following order produces a full Binary Search Tree:

19, 6, 3, 2, 5, 15, 10, 17, 35, 22, 20. 30, 45, 42, 55

CMP-338 Name:_____

Midterm Spring 2015 Solutions

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
Midterm Spring 2015

Name:_____

Solutions
Version 2

7. (20 Points) Given the following ListInterface, the correct reference-based implementation is:

```
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 implements ListInterface {
   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();
      }
      return obj;
   }

   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 boolean 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;
   }
}
```

8. (40 Points) The correct BinarySearchTree implementation is:

```
public class BinarySearchTree implements
BinarySearchTreeInterface {
   private TreeNode root = null;

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

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

   @Override
   public void insert(TreeItem item) {
      root = insertItem(root, item);
   }

   private TreeNode insertItem(TreeNode node,
                               TreeItem newItem) {
      TreeNode newSubtree;

      if (node == null) {
         node = new TreeNode(newItem);
         return node;
      }

      if (newItem.compareTo(node.getItem()) < 0) {
         newSubtree =
insertItem(node.getLeftChild(), newItem);
         node.setLeftChild(newSubtree);
         return node;
      } else {
         newSubtree =
insertItem(node.getRightChild(), newItem);
         node.setRightChild(newSubtree);
         return node;
      }
   }

   @Override
   public TreeItem retrieve(int key) {
      return retrieveItem(root, key);
   }

   private TreeItem retrieveItem(TreeNode node,
                                 int key) {
      TreeItem treeItem;
      TreeItem compareItem = new TreeItem(key);
      if (node == null) {
         treeItem = null;
      } else if
(compareItem.compareTo(node.getItem()) == 0) {
         treeItem = node.getItem();
      } else if
(compareItem.compareTo(node.getItem()) < 0) {
         treeItem =
retrieveItem(node.getLeftChild(), key);
      } else {
         treeItem =
retrieveItem(node.getRightChild(), key);
      }
      return treeItem;
   }
}
```

CMP-338 Name:_____

Midterm Spring 2015 Solutions

Version 2

9. (20 Points) The correct method for merging two arrays:

```
public void merge(T[]a, T[]b, T[]c) {
        int firstA = 0;
        int lastA  = a.length - 1;
        int firstB  = 0;
        int lastB  = b.length - 1;

        int cIindex = 0;

        while ((firstA <= lastA)  && (firstB  <= lastB)) {
                if (a[firstA].compareTo(b[firstB]) < 0)  {
                        c[cIindex]  = a[firstA++];
                } else {
                        c[cIindex]  = b[firstB++];
                }
                cIindex++;
        }   // end while

        while (firstA <= lastA)  {
                c[cIindex++] = a[firstA++] ;
        }   // end while


        while (firstB <= lastB)  {
                c[cIindex++] = b[firstB++];
        }   // end while
}
```

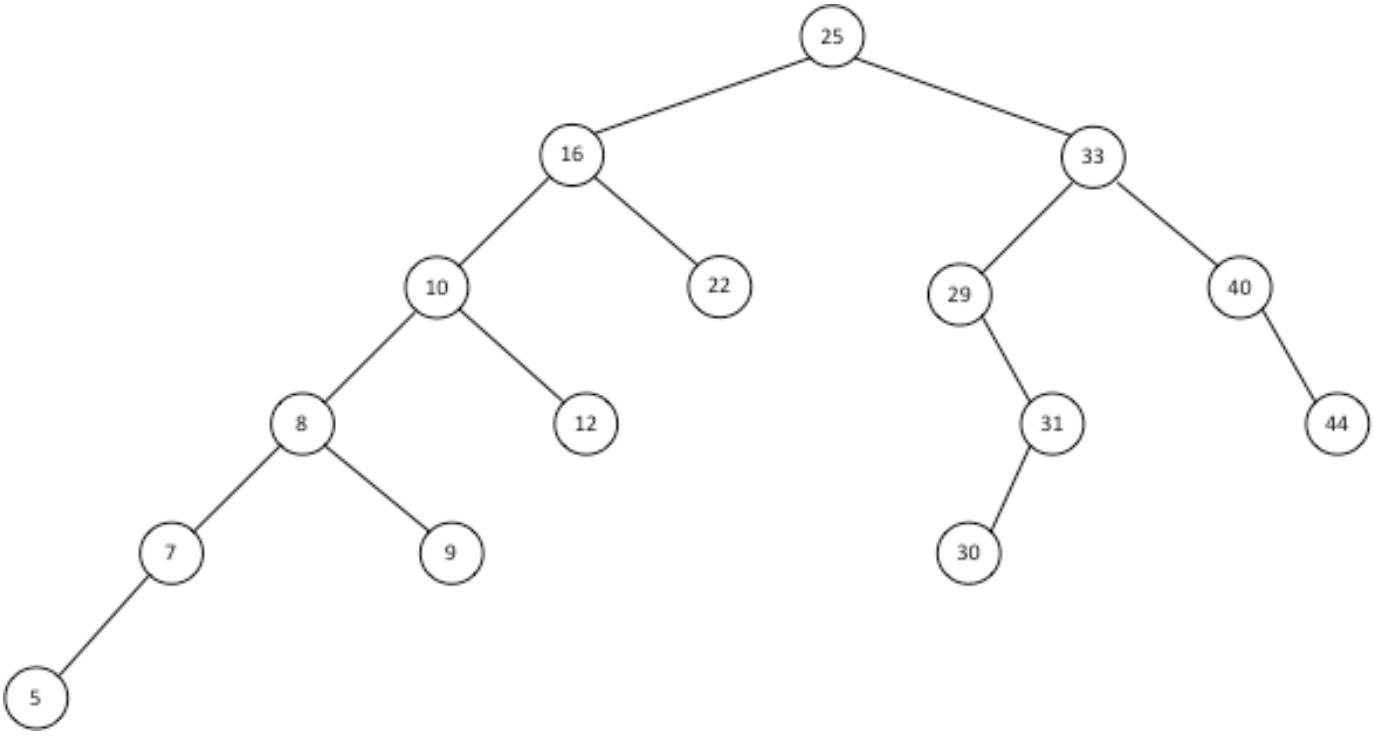
CMP-338 Name:_____

Midterm Spring 2015 Solutions

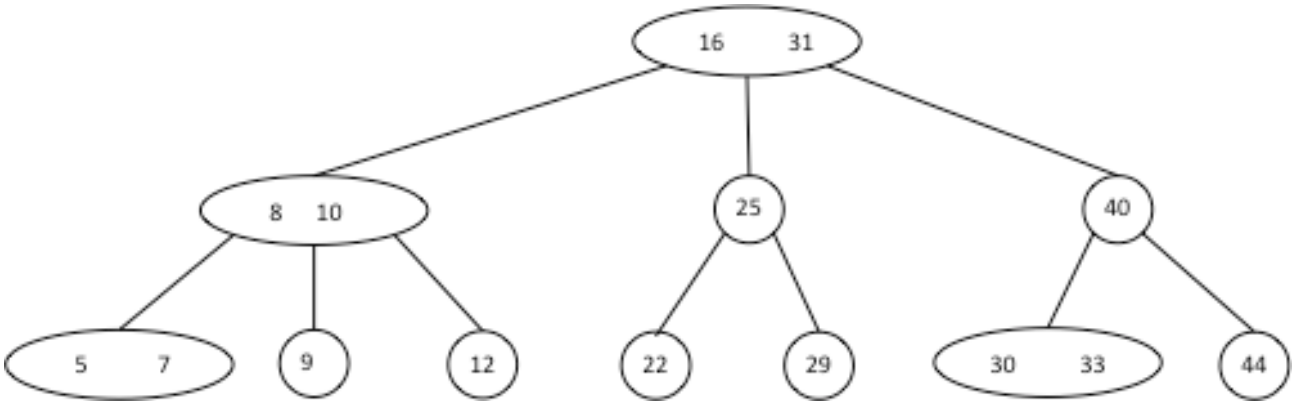
Version 2

10. (16 Points) The following list of numbers: 25, 16, 33, 22, 10, 8, 7, 9, 5, 12, 29, 31, 40, 44, 30 inserted in the given order into:

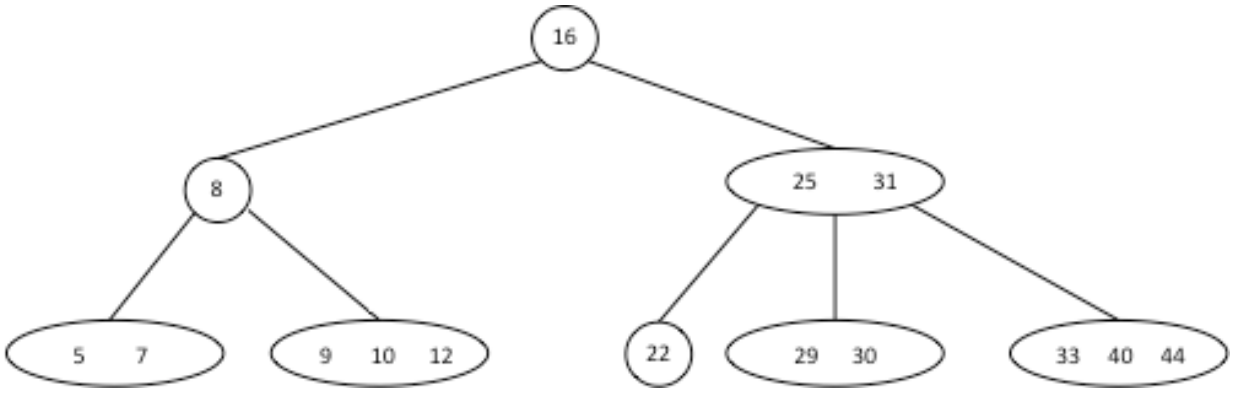
a. (4 Points) Into a Binary Search Tree produces the following tree:



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



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



d. (4 Points) The numbers in the following order produces a full Binary Search Tree:

22, 9, 7, 5, 8, 12, 10, 16, 31, 29, 25, 30, 40, 33, 44