1. (10 Points) Multiple Choice:

A. (1 Point) The midpoint of a sorted array can be found by _____, where first is the index of the first item in the array and last is the index of the last item in the array.
   a. first / 2 + last / 2
   b. first / 2 – last / 2
   c. (first + last) / 2
   d. (first – last) / 2

B. (1 Point) A(n) _____ is an instance of a class.
   a. method
   b. data field
   c. interface
   d. object

C. (1 Point) In Java, a class can extend _____.
   a. at most 1 class
   b. at most 16 classes
   c. at most 32 classes
   d. as many classes as required

D. (1 Point) If a linked list is empty, the statement head.getNext() will throw a(n) _____.
   a. IllegalAccessException
   b. ArithmeticException
   c. IndexOutOfBoundsException
   d. NullPointerException

E. (1 Point) Which of the following statements deletes the node that curr references?
   a. prev.setNext(curr);
   b. curr.setNext(prev);
   c. curr.setNext(curr.getNext());
   d. prev.setNext(curr.getNext());

F. (1 Point) Which of the following is the postfix form of the infix expression: (a + b) * c / d
   a. a b + c * d /
   b. a b * c / d +
   c. a + b * c / d
   d. a b + c d * /

G. (1 Point) Inheritance should only be used when a(n) _____ relationship exists between the superclass and the subclass.
   a. is-a
   b. has-a
   c. has-many
   d. similar-to

H. (1 Point) Each node in a binary tree has _____.
   a. exactly one child
   b. at most one child
   c. exactly two children
   d. at most two children

I. (1 Point) A tree with n nodes must contain _____ edges.
   a. n
   b. n – 1
   c. n – 2
   d. n / 2

J. (1 Point) A graph is _____ if each pair of distinct vertices has a path between them.
   a. complete
   b. disconnected
   c. connected
   d. full

2. (20 Points) The corrected QuickSort Class:

```
import java.util.Vector;

public class QuickSort <T extends Comparable<? super T>> {

	public void quickSort(Vector<T> theVector,
				int first, int last) {
		if (first < last) {
			int pivotIndex = partition(theVector, first, last);
			quickSort(theVector, first, pivotIndex - 1);
			quickSort(theVector, pivotIndex + 1, last);
		}
	}

	public void choosePivot(Vector<T> theVector,
				int first, int last) {

		// The pivot will be the middle value of first, mid and last
		int mid = (first + last) / 2;
		T temp = theVector.elementAt(first);
		T f = theVector.elementAt(first);
		T m = theVector.elementAt(mid);
		T l = theVector.elementAt(last);

		if (((f.compareTo(m) <= 0) && (l.compareTo(m) >= 0)) ||
		    ((f.compareTo(m) >= 0) && (l.compareTo(m) <= 0))) {
			theVector.set(first, theVector.elementAt(mid));
			theVector.set(mid, temp);
		} else if (((f.compareTo(l) <= 0) && (m.compareTo(l) >= 0)) ||
		           ((f.compareTo(l) >= 0) && (m.compareTo(l) <= 0))) {
			theVector.set(first, theVector.elementAt(last));
			theVector.set(last, temp);
		}
	}

	public int partition(Vector<T> theVector,
				int first, int last) {
		T tempItem;
		choosePivot(theVector, first, last);
		T pivot = theVector.elementAt(first); // reference pivot
		int lastS1 = first; // index of last item in S1
		for (int firstUnknown = first + 1; firstUnknown <= last; ++firstUnknown) {
			if (theVector.elementAt(firstUnknown).compareTo(pivot) < 0) {
				++lastS1;
				tempItem = theVector.elementAt(firstUnknown);
				theVector.set(firstUnknown, theVector.elementAt(lastS1));
				theVector.set(lastS1, tempItem);
			}
		}
		tempItem = theVector.elementAt(first);
		theVector.set(first, theVector.elementAt(lastS1));
		theVector.set(lastS1, tempItem);
		return lastS1;
	}
}
```

3. (20 Points) The method to merge 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++;
   }

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


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

4. (20 Points) The correct addSorted method:

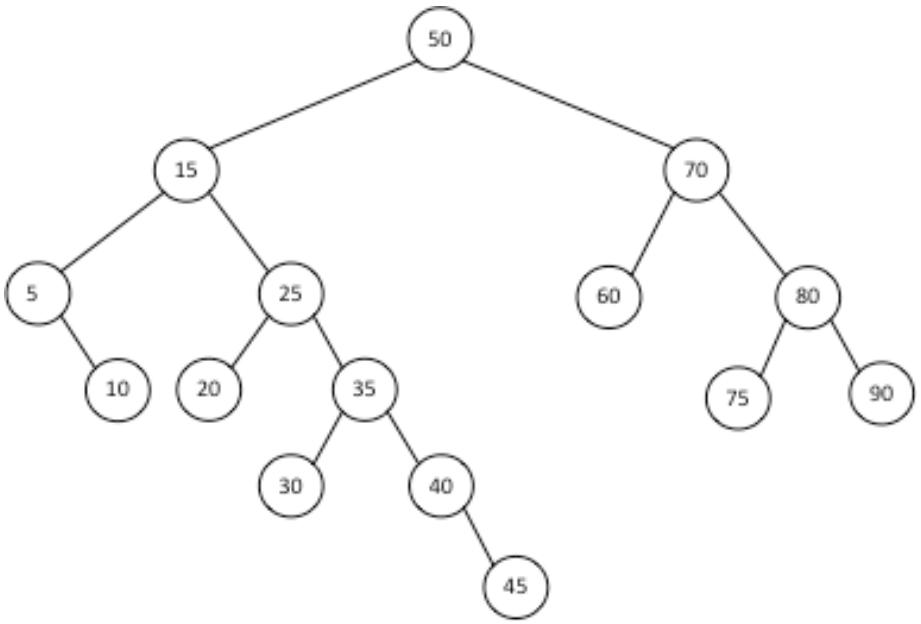
```
public void addSorted(T element) {
   ListNode<T> newNode = new ListNode<T>(element);

   if (head == null) {
      // list is empty, add the element
      this.add(element);
   } else {
      ListNode<T> curNode = head;
      while (curNode != null) {
         if (curNode.getElement().compareTo(element) <= 0) {
            curNode = curNode.getNext();
         } else {
            break;
         }
      }

      if (curNode == null) {
         // got to end of list, add element there
         this.add(element);
      } else if (curNode == head) {
         // the new element is smaller that the head
         // insert it before the head
         newNode.setNext(curNode);
         curNode.setPrevious(newNode);
         head = newNode;
         size++;
      } else {
         // curNode points to node containing an element
         // larger than the new element being added
         curNode.getPrevious().setNext(newNode);
         newNode.setPrevious(curNode.getPrevious());
         curNode.setPrevious(newNode);
         newNode.setNext(curNode);
         size++;
      }
   }
}
```

5. (20 Points) The correct heapify method:

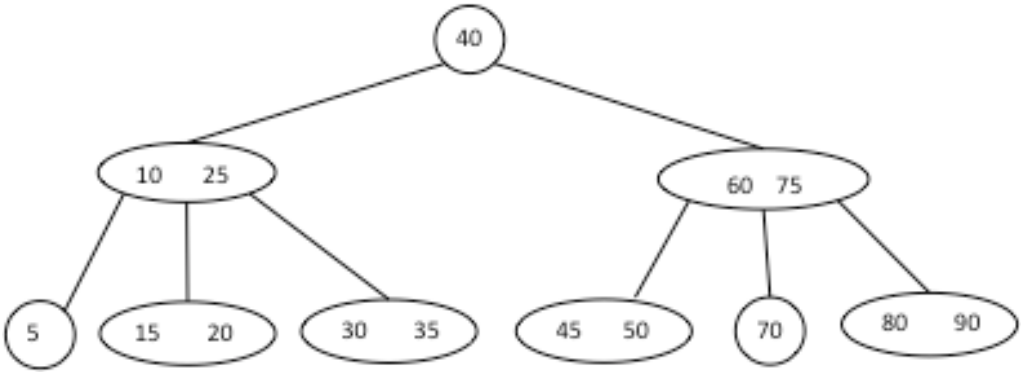
```
private void heapify() {
   int last = this.heapArray.length - 1;
   int parent = (last - 1) / 2;

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

private void siftDown(int node) {

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

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

      if ((rightChild < this.eapArray.length) &&
          (this.heapArray[swap].compareTo(this.heapArray[rightChild]) < 0)) {
         swap = rightChild;
      }

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

6. (20 Points) The following list of numbers: 50, 70, 60, 15, 25, 80, 5, 35, 40, 75, 90, 10, 20, 30, 45 inserted in the given order:

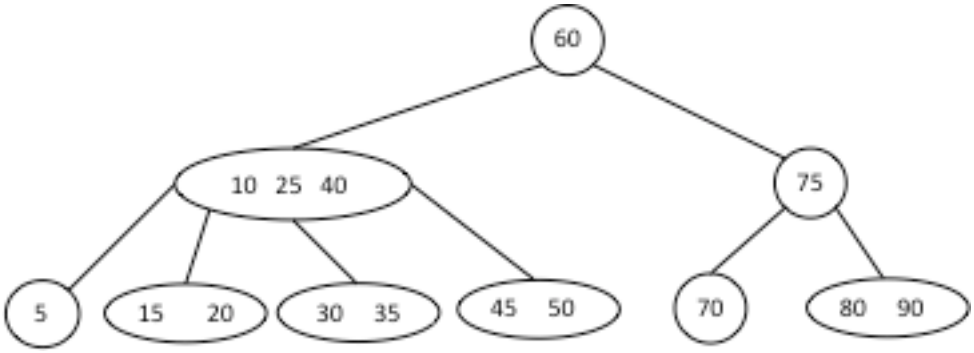
   a. (5 Points) Into a Binary Search Tree produces the following tree:



   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:

40, 20, 70, 10, 30, 50, 80, 5, 15, 25, 35, 45, 60, 75, 90