Instructions

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

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 Points) 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.getFirstNext());
   - d. prev.setNext(curr.getFirstNext());

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

```java
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, last, pivotIndex - 1);
            quickSort(theVector, pivotIndex + 1, first);
        }
    }

    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(first));
                theVector.set(lastS1, tempItem);
            }
        }
        tempItem = theVector.elementAt(first);
        theVector.set(last, theVector.elementAt(lastS1));
        theVector.set(lastS1, tempItem);
        return lastS1;
    }
}
```
3. (20 Points) Write a method that merges 2 sorted arrays (a and b) of a generic type T into another array (c). You can assume that $<\text{T extends Comparable<}?> \text{super T}>>$ and that c is big enough for the merge. The method should have the following signature:

```
public void merge(T[] a, T[] b, T[] c) {
}
```
4. (20 Points) Given the following definitions for `ListNode` and `LinkedList` classes. Write the `addSorted` method which adds an element to the `LinkedList` in sorted order. The method should have the following signature:

```
public void addSorted(T element) {
}
```

```java
public class ListNode
    <T extends Comparable <? super T>> {
    private T element;
    private ListNode<T> previous;
    private ListNode<T> next;

    public ListNode(T element) {
        this.element = element;
        this.previous = null;
        this.next = null;
    }

    public T getElement() {
        return element;
    }

    public void setElement(T element) {
        this.element = element;
    }

    public ListNode<T> getPrevious() {
        return previous;
    }

    public void setPrevious(ListNode<T> previous) {
        this.previous = previous;
    }

    public ListNode<T> getNext() {
        return next;
    }

    public void setNext(ListNode<T> next) {
        this.next = next;
    }
}
```

```java
public class LinkedList
    <T extends Comparable <? super T>> {
    private ListNode<T> head;
    private ListNode<T> tail;
    private int size;

    public LinkedList() {
        this.head = null;
        this.tail = null;
        this.size = 0;
    }

    public int size() {
        return this.size;
    }

    public boolean isEmpty() {
        return (this.size == 0);
    }

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

        if (head == null) {
            head = newNode;
            tail = newNode;
            size = 1;
        } else {
            tail.setNext(newNode);
            newNode.setPrevious(tail);
            tail = newNode;
            size++;
        }
    }

    public T getElement(int index) {
        T element = null;

        if (index < size) {
            ListNode<T> cur = head;
            while (index > 0) {
                curNode = cur.getNext();
                index--;
            }
            element = cur.getgetElement();
        }

        return element;
    }

    public T getAndRemoveElement(int index) {
        T element = null;

        if (index < size) {
            ListNode<T> cur = head;
            while (index > 0) {
                curNode = cur.getNext();
                index--;
            }
            element = cur.getgetElement();
            curNode.getPrevious().setNext(curNode.getNext());
            curNode.getNext().setPrevious(curNode.getPrevious());
            size--;
        }

        return element;
    }
}
```
5. **(20 Points)** A heap can be represented by an array `heapArray`.

   An element at index `n` in `heapArray` has its left child at index `2n + 1` and its right child at index `2n + 2`. Also, an element at index `m` in `heapArray` has its parent at index `(m - 1) / 2`.

   A Max-Heap is a heap where every node in the heap is bigger than its children.

   Given a random array `heapArray`, write the `heapify()` method that converts `heapArray` into a Max-Heap. You can assume the following:

   ```java
   T heapArray[];
   
   And
   
   <T extends Comparable<? super T>>
   
   The method should have the following signature:
   
   ```java
   private void heapify() {
   }
   ```
6. **(20 Points)** Given the following list of numbers: 50, 70, 60, 15, 25, 80, 5, 35, 40, 75, 90, 10, 20, 30, 45 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?