

Each question is worth 10 marks each.

**Question 1** [10 marks]

A *hetero-literal* word pair is defined as a pair of words having no letters in common.

As examples, "TOM" and "CRUISE" form a hetero-literal word pair  
"WAYNE" and "GRETZKY" do not (because of the E and Y in each word).

Write a static Java method called `isHetero()` that has two `String` parameters, `wordA` and `wordB`. The method should return a boolean value: `true` if `wordA` and `wordB` form a hetero-literal word pair, and `false` if they do not. You may use any of the methods in the `String` class. In particular, you may wish to use the method `charAt(n)` to obtain the character at position `n` in the `String`. You may assume that the strings contain only uppercase letters.

**Question 2** [10 marks]

Consider the main method shown below:

```
public static void main (String[] args)
{
    double x1, y1, x2, y2, distance;
    x1 = 2.4;
    y1 = 1.5;
    x2 = 6.7;
    y2 = 3.0;
    distance = Math.sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
    System.out.println( "Distance between points: " + distance );
}
```

**a) [7 marks]**

Write a complete class `CPoint` that could be used by the following main method to give exactly the same results as the main method show above. You should properly hide (i.e., encapsulate) any aspects of your class that are not directly used by the main program below.

```
public static void main (String[] args)
{
    CPoint p1 = new CPoint(2.4,1.5);
    CPoint p2 = new CPoint(6.7,3.0);
    double distance = p1.distanceFrom(p2);
    System.out.println( "Distance between points: " + distance );
}
```

**b) [3 marks]**

Add a method, called `isOnBisector()`, to the `CPoint` class. This method determines whether this point is on the *perpendicular bisector* of a line segment defined by two other `Point` objects, `p1` and `p2`, which are passed as parameters. That is, it should return a boolean value: `true` if this point is *equidistant* from `p1` and `p2`, and `false` otherwise. You may assume that computer arithmetic is perfectly precise.

```
// example usage of isOnBisector() method
CPoint p3 = new CPoint(5.0,1.1);
if( p3.isOnBisector(p1,p2) )
    System.out.println( "Yes, p3 is on the bisector." );
else
    System.out.println( "No, p3 is not on the bisector." );
```

**Question 3** [10 marks]

Assume that a linked list has nodes that have been defined by the following:

```
class Elem
{
    public int value;
    public Elem next;
}
```

Write a Java method that will be invoked by a call of the form `x.printMerged(y)`, where `x` and `y` are both linked lists of type `List`. The contents of each list are already sorted in strictly ascending order. The method should print *in ascending order* all of the values contained by the union of both lists, one value per line. You may assume that the lists have no elements in common. In your solution, *do not sort or modify either list, write/call any other methods except `System.out.println()`, or create any new objects.*

```
class List
{
    private Elem head;
    // fill in rest
}
```

**Question 4** [10 marks]

Lacsap's triangle is a triangular array of integers in which each row begins and ends with the row number, and each number in the interior of the triangle is the sum of two the numbers on either side in the previous row. The diagram below shows the first five rows of Lacsap's triangle.

```

          1
        2  2
       3  4  3
      4  7  7  4
     5 11 14 11 5
```

Write a static Java method `lacsap()`, that takes a single parameter `int n`. It should print the first `n` rows of Lacsap's triangle, with one row per line and one space between the numbers in a row. For simplicity, assume that `n >= 1` and do not try to print the triangle in the symmetrical form shown above.

*Hint:* you will need to use arrays to store your results.

**Question 5** [10 marks]

A polynomial of the form

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_ix^i + \dots + a_nx^n$$

can be evaluated more efficiently using a technique known as *Horner's rule*. Horner's rule avoids computing the numerous  $x^i$  values explicitly, since this is redundant and slow, by rewriting and evaluating  $p(x)$  in the form shown below

$$p(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \dots + x \cdot (a_i + \dots + x \cdot (a_n) \dots) \dots))$$

Write a **recursive** method `horner()`, which evaluates a polynomial using Horner's rule and returns the numeric value of  $p(x)$ . The parameter `double[] a` holds the coefficients  $a_i$  in an array, `double x` is the value at which the polynomial is to be evaluated, and `int i` is a marker you must use to keep track of your recursion. *If you do not use recursion, you will receive a maximum of 5 marks.*

```
public static double horner (double[] a, double x, int i)
```

All questions are worth 10 marks each.

**Question 1** [10 marks]

Write a method to determine whether or not an integer array is already in sorted order. Be efficient. *Do not sort or otherwise modify the array.* Return `true` if the array is already in ascending or descending order, otherwise return `false`.

Note that the arrays `{ }`, `{ 42 }`, `{ 1, 2, 6, 9, 9, 11 }` and `{ 11, 9, 9, 6, 2, 1 }` are considered sorted, however `{ 1, 3, 4, 2, 0 }` is not.

```
public class Question1
{
    public static boolean isSorted( int[] array )
        // fill in rest
}
```

**Question 2** [10 marks]

Write a **non-recursive** method to reverse the order of the elements in a linked list. Note that all elements are to be rearranged. You may not use any other methods, such as `insert()`, to help you. Do not needlessly create new `ListElem` objects.

```
class ListElem
{
    int value;
    ListElem next;
}

class LinkedList
{
    private ListElem head;

    public void reverse()
    {
        // fill in rest
    }
}
```

### Question 3 [10 marks]

On the last page of this test is the code for the QuickSort routine described in class. Do not write anything on that page, but you may remove it from the test and write your answer in the space below.

One change has been made to the QuickSort class: the `sortSubArray()` method now counts how many times it is called while sorting, and `qsort()` must initialize and return this value.

You are to determine how many times the `sortSubArray()` method is called when given the following program. For full marks, you must show how you arrived at your answer.

```
class Question3
{
    public static void main (String[] args)
    {
        int[] a = { 6, 3, 12, 9, 2, -4, 13, 11 };
        int numberOfCalls = QuickSort.qsort( a );
        System.out.print ( "The quicksort routine used " );
        System.out.println( numberOfCalls + " recursive calls." );
    }
}
```

### Question 4 [10 marks]

Some businesses spell out their phone number with letters or a combination of numbers and letters. For example, 310-DELL (i.e., 310-3355) is the phone number for Dell Computer. Using recursion, write a program (including a `main()` method) that maps a phone number to all possible strings.

The **recursive** method must take an integer (i.e., a phone number) as a parameter, plus any additional parameters you need. It must print out *all* possible letter and number combinations, one per line. *Note that the phone number may contain more or less than the usual 7 digits.*

If given the number 3, your program should print “3”, “D”, “E”, and “F” (one per line) as solutions. If given the number 3103355, it must print “310DELL”, “D10DELL”, “D10DDJJ”, etc. (in any order, for a total of  $4 \cdot 1 \cdot 1 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 1024$  unique solutions). It should quit if a number  $\leq 0$  is entered. You do **not** need to print out the dash “-” or any other special characters, e.g. parentheses for (905).

The two-dimensional array containing this mapping is given to you; notice that `mapping[digit][0]` gives you the first possible mapping string, `mapping[digit][1]` give the next one (if possible), etc.

```
class Question4
{
    private static String[][] mapping =
    {
        { "0" },
        { "1" },
        { "2", "A", "B", "C" }, { "3", "D", "E", "F" },
        { "4", "G", "H", "I" }, { "5", "J", "K", "L" }, { "6", "M", "N", "O" },
        { "7", "P", "R", "S" }, { "8", "T", "U", "V" }, { "9", "W", "X", "Y" }
    };

    // fill in rest
}
```

### QuickSort Code for Question 3

You may remove this page from the test. *Do not write on this page.*

```
class QuickSort
{
    private static int numCalls;

    public static int qsort( int[] array )
    {
        numCalls = 0;
        sortSubArray( array, 0, array.length-1 );
        return numCalls;
    }

    private static void sortSubArray( int[] array, int l, int u )
    {
        numCalls++;

        if( l >= u )
            return;

        // find the pivot value
        // use the array[l] point as the pivot
        final int pivot = array[l];

        int m = l;

        // scan the sub-array from l to u,
        //     partitioning it to 2 groups.
        // move values < p to positions <=m in the array
        // keep values >= p to positions >m in the array
        for( int i=l+1; i <= u; i++ ) {
            if( array[i] < pivot ) {
                m++;
                swap( array, m, i );
            }
        }

        // move the pivot to the correct position
        swap( array, l, m );

        // recursively sort the partitions
        sortSubArray( array, l, m-1 );
        sortSubArray( array, m+1, u );
    }

    private static void swap( int[] array, int i, int j )
    {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }
}
```