

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
APS105 — Computer Fundamentals
Quiz #2 Solutions
March 12, 1999, 4:10–5:00 p.m.

Exam Type A:

This is a “closed book” examination; no aids are permitted.

Calculator Type 4:

No calculators allowed.

Write your answer in the space that follows each question. If more space is required, continue on the back of the preceding page, and be sure to indicate on the front that you have done so.

The examination has 6 pages.

All questions are worth 10 marks each.

For all questions, you may assume that the following `Stdin` methods are available:

`Stdin.getInt()`, `Stdin.getDouble()`, and `Stdin.getChar()`.

Name _____

Circle your scheduled lab day: **Monday** or **Tuesday**

Student Number _____

MARKS

Question #	1	2	3	4	Total
Marks	/10	/10	/10	/10	/40

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 )
    {
        // first, test if it is ascending
        boolean isAscending = true;
        for( int i=1; i < array.length; i++ ) {
            if( array[i-1] > array[i] ) {
                isAscending = false; // it is not ascending
                break; // maybe it is descending
            }
        }
        if( isAscending )
            return true; // it is sorted

        // it is not ascending. test if it is descending.
        for( int i=1; i < array.length; i++ ) {
            if( array[i-1] < array[i] ) {
                return false; // it is not descending either
            }
        }

        // it is descending
        return true;
    }
}
```

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()
    {

        ListElem revhead = null;

        while( head != null ) {

            // remove the 'head' of original list
            ListElem oldHeadElem = head;
            head = head.next;

            // insert it into the reversed list
            oldHeadElem.next = revhead;
            revhead = oldHeadElem;
        }

        head = revhead;
    }
}
```

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." );
    }
}

qsort
+- sortSubArray( a, 0, 7 )
|   after partitioning:    a = { 6, 3, 2, -4, 12, 9, 13, 11 }
|   after swapping pivot: a = { -4, 3, 2, 6, 12, 9, 13, 11 }
|   m = 3
+- sortSubArray( a, 0, 2 )
|   |   after partitioning:    a = { -4, 3, 2, 6, 12, 9, 13, 11 }
|   |   after swapping pivot: a = { -4, 3, 2, 6, 12, 9, 13, 11 }
|   |   m = 0
|   +- sortSubArray( a, 0, -1 )
|   |   does nothing, returns immediately because l >= u
|   +- sortSubArray( a, 1, 2 )
|   |   after partitioning:    a = { -4, 3, 2, 6, 12, 9, 13, 11 }
|   |   after swapping pivot: a = { -4, 2, 3, 6, 12, 9, 13, 11 }
|   |   m = 2
|   |   +- sortSubArray( a, 1, 1 )
|   |   |   does nothing, returns immediately because l >= u
|   |   +- sortSubArray( a, 3, 2 )
|   |   |   does nothing, returns immediately because l >= u
+- sortSubArray( a, 4, 7 )
|   after partitioning:    a = { -4, 2, 3, 6, 12, 9, 11, 13 }
|   after swapping pivot: a = { -4, 2, 3, 6, 11, 9, 12, 13 }
|   m = 6
+- sortSubArray( a, 4, 5 )
|   |   after partitioning:    a = { -4, 2, 3, 6, 11, 9, 12, 13 }
|   |   after swapping pivot: a = { -4, 2, 3, 6, 9, 11, 12, 13 }
|   |   m = 5
|   |   +- sortSubArray( a, 4, 4 )
|   |   |   does nothing, returns immediately because l >= u
|   |   +- sortSubArray( a, 6, 5 )
|   |   |   does nothing, returns immediately because l >= u
+- sortSubArray( a, 7, 7 )
|   does nothing, returns immediately because l >= u
```

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 $4 * 1 * 1 * 4 * 4 * 4 * 4 = 1024$ unique solutions). It should quit if a number ≤ 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" }
    };

    public static void mapPhoneNumber( int number, String solution )
    {
        if( number == 0 )
            System.out.println( solution );
        else {
            int digit = number % 10;
            for( int i = 0; i < mapping[digit].length; i++ ) {
                mapPhoneNumber( number/10, mapping[digit][i]+solution );
            }
        }
    }

    public static void main( String[] args )
    {
        while( true ) {
            int n = Stdin.getInt();
            if( n <= 0 ) break;
            mapPhoneNumber( n, "" );
        }
    }
}
```

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;
    }
}
```