NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR CS1020
Semester 2: AY2013/2014

CS1020 – Data Structures and Algorithms I

May 2014 Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FOURTEEN (14)** questions and comprises **TEN (10)** printed pages.
2. This is a **CLOSED BOOK** examination. You are allowed to bring in ONE (1) piece of A4 handwritten reference sheet. No photocopies allowed.
3. Calculators are not allowed.
4. Answer all questions in the Answer Booklet provided.
5. You may write your answers in pencil.

SECTION A (6 Multiple Choice Questions: 18 marks)

Each question has one correct answer. Three (3) marks are awarded for each correct answer; no penalty for wrong answer.

1. What is the output of the following code fragment?

```
ArrayList<Integer> arr = new ArrayList<Integer>();

for (int i=0; i<10; i++)
   arr.add(i);

for (int i=0; i<arr.size()/2; i++)
   arr.add(i, arr.remove(arr.size()-1-i));

System.out.println(arr);
```

(A) **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**
(B) **[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]**
(C) **[9, 7, 5, 3, 1, 0, 2, 4, 6, 8]**
(D) **[9, 7, 5, 3, 1, 8, 6, 4, 2, 0]**
(E) **[9, 0, 8, 1, 7, 2, 6, 3, 5, 4]**

2. Consider the following program:

```
public class Test1 {
    public static void main(String[] args) {
         recurse(12);
    }

    public static void recurse(int n) {
         System.out.println("The number is " + n);
         if (n-- > 0) {
             recurse(n--);
         }
    }
}
```

How many lines will be printed in the output?

(A) 5
(B) 6
(C) 7
(D) 12
(E) None of the above

3. Consider the following program:

```
import java.util.*;

public class Test2 {
   public static void main(String[] args) {
      int[] array = new int[] {1,2,3,4,5};

      Stack<Integer> s = new Stack<Integer>();
      for (int i: array) {
         s.push(i);
         if (i%2 == 0) s.pop();
      }

      Queue<Integer> q = new LinkedList<Integer>();
      for (int i: array) {
         q.offer(i);
         if (i%2 == 0) q.poll();
      }
   }
}
```

After the program is run, how many integers in the stack s can also be found in the queue q?

(A) 0
(B) 1
(C) 2
(D) 3
(E) 4

4. Suppose we are sorting an array of eleven integers using **Quick sort** as presented in the lecture. Suppose we have just finished the first partitioning and the pivot swapping to result in the following content in the array:

2, 5, 3, 7, 9, 15, 20, 18, 25, 30, 10

From the above resulting array, there are various possible integers among them that could be the pivot. How many of them are there?

(A) 3
(B) 4
(C) 5
(D) 6
(E) None of the above.

5. Given an *n*×*m* matrix **arr**, you want to create another matrix **newArr** with the elements in **arr** shifted by a shift amount. For example, given the matrix **arr** on the left below, and a shift amount of 3, the matrix **newArr** on the right is created.

arr

1	2	3	4
5	6	7	8
9	10	11	12

$\xrightarrow{\text{Shift 3}}$

newArr

10	11	12	1
2	3	4	5
6	7	8	9

The elements in **arr** are shifted 3 positions in row-major order, and if an element is shifted out of the matrix, it appears at the start of the matrix (see elements 10, 11, 12). One way to do is to "linearise" the element positions in **arr**, and map their respective new positions in **newArr**. For example, the second element in **arr** now appears in the fifth position in **newArr**.

The partial code is shown below:

```
int[][] newArr = new int[arr.length][arr[0].length];

int index, numRows = arr.length, numCols = arr[0].length;
for (int r = 0; r < numRows; r++) {
   for (int c = 0; c < numCols; c++) {

      index = 

      newArr[index/numCols][index%numCols] = arr[r][c];
   }
}
```

Complete the missing part in the dashed box, where **`shiftAmt`** is the shift amount:

(A) **`(r*numCols + c + shiftAmt);`**

(B) **`(r*numCols + c + shiftAmt) % numRows;`**

(C) **`(r*numCols + c + shiftAmt) % numCols;`**

(D) **`(r*numCols + c + shiftAmt) % (numRows*numCols);`**

(E) None of the above.

6. Which of the following statements is true?

(A) Quadratic probing may avoid the clustering problem completely.

(B) A perfect hash function must distribute the keys evenly in the hash table.

(C) It is effective (i.e. there are fewer collisions) to use direct addressing table when the keys are not dense.

(D) The probe sequence of modified linear probing covers all positions in the hash table.

(E) None of the above.

SECTION B (82 marks)

7. [8 marks]
The **ListNode** class is defined as usual as follows:

```
class ListNode <E> {
   private E element;
   private ListNode <E> next;

   public ListNode(E element) { this(element, null); }

   public ListNode(E element, ListNode <E> next) {
      this.element = element;
      this.next = next;
   }

   public ListNode <E> getNext() { return this.next; }

   public E getElement() { return this.element; }

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

A **MyLinkedList** class is defined for linked lists containing objects of **ListNode**. In the **MyLinkedList** class, we want to include an instance method **searchAndSwitch(E item)** to search for the item in the linked list. If it is found and the node that contains item is not the last node, the method returns true and switches the node containing the item with its next node. If the linked list does not contain the item, or the node that contains the item is the last node, the method should return false. The switching must be done by manipulation of the pointers, not by merely swapping the contents of the nodes.

For example, given the original linked list **names** as follows:

[Abe, Gigi, Flora, Andy, Jack, Zoe]

calling **names.searchAndSwitch("Andy");** will result in the following linked list:

[Abe, Gigi, Flora, Jack, Andy, Zoe]

The method **searchAndSwitch(E item)** is partially given. Complete it without changing any part that is given. You may add appropriate comments in your code.

8. [12 marks]
A **MyPolygon** class has been defined as follows for *n*-sided polygons (*n* ≥ 3) whose vertices are **Point** objects, arranged in clockwise manner.

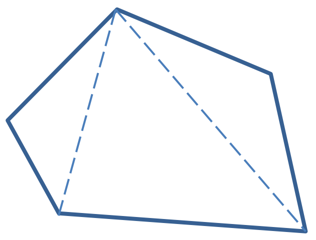
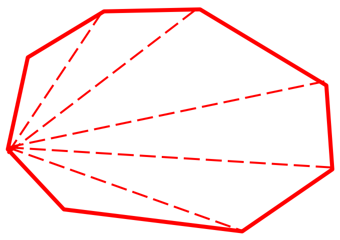
```
import java.awt.Point;
import java.util.ArrayList;

class MyPolygon {
  private int size; // number of vertices
  private ArrayList<Point> vertices;

  public MyPolygon(int num, ArrayList<Point> pts) {
       size = num;
       vertices = new ArrayList<Point>(pts);
  }

  // Other methods not shown as they are not required
  // for this question
}
```

Here, we assume that polygons are convex. A convex polygon is one in which if we pick any point inside the polygon and draw a line from this point to any vertex of the polygon, that line will not cut any of the sides of the polygon. The diagrams below show two convex polygons, one is 5-sided and the other 8-sided.

In computer graphics and computational geometry, a common technique called **triangulation** is to split a convex polygon into a number of triangles. In the above examples, the two polygons are triangulated with dashed lines as shown.

Write the following method, with appropriate comments, in the **MyPolygon** class:

```
public ArrayList<MyPolygon> triangulate()
```

This method triangulates a **MyPolygon** object into an ArrayList of triangles. As triangles are 3-sided polygons, hence the result is an ArrayList of MyPolygon objects.

Appendix A contains relevant information about **ArrayList** class.

9. Given a stack of integers, write **efficient** Java method(s) or algorithm(s), with appropriate comments, to remove all the maximum values in the stack. For example, if the stack contains 3, 8, 4, 8, 5, 4, then the two 8's are to be removed. What is the time complexity of your algorithm?

   You are not allowed to use additional data structures except a few temporary variables for computations in this question. Only the stack's **push()**, **pop()**, **empty()**, and **peek()** methods are allowed. [12 marks]

10. Write an **efficient recursive** Java method (or algorithm) called **iSearch(…)**, with appropriate comments, that returns the value *i* such that *A*[*i*] = *i* in an array *A*[] of increasing unique integers. For example, if *A* contains { -7, 0, 1, 3, 8, 12 }, then the method returns 3. If no such *i* is found, the method returns -1. If there is more than one correct answer, you may return anyone of them.

    You may write helper method if necessary. No mark is awarded if recursion is not used.

    What is the time complexity of your code/algorithm? [10 marks]

11. Given an array of 6 integers: **1, 3, 2, 4, 6, 5**, how many passes and key comparisons are required to sort the array using the following sorting methods? [9 marks]

    (A) Improved Bubble Sort

    (B) Selection Sort

    (C) Insertion Sort

12. What is the Big-O complexity of the following code fragment, assuming that statement1, statement2, statement3 and statement4 are basic statements that take O(1) time?

    [10 marks]

```
for (int i=1; i<n; i=i*3) {
    statement1;
    for (int j=0; j<i; j++) {
        statement2;
        for (int k=n-1; k>0; k--)
            statement3;
        for (int h=n-1; h>0; h=h/2)
            statement4;
    }
}
```

13. [15 marks]

(A) Consider a hash table of length m = 13 with the hash function:

```
h(key) = key mod m
```

and use double hashing with the following second hash function to resolve collisions:

```
h2(key) = 11 - (key mod 11)
```

(i) Write a possible sequence for inserting keys 17, 24, 26, 37 and 52 such that the resulting content of the hash table is as follows: [3 marks]

52	26			17			24				37	
0	1	2	3	4	5	6	7	8	9	10	11	12

(ii) Then we insert the key 115. What are the keys accessed/probed in this insertion? [3 marks]

(iii) Then we delete all the keys that are not accessed/probed in the insertion in (ii), and search for the key 63. What is the probe sequence for this search? [3 marks]

(B) You are given the following mapping table between letters and keys:

Letter	a	b	c	d	e	f	g	h	i	j	k	l	m
Key	0	1	2	3	4	5	6	7	8	9	10	11	12

Letter	n	o	p	q	r	s	t	u	v	w	x	y	z
Key	13	14	15	16	17	18	19	20	21	22	23	24	25

Insert the letters in “lazydog” in sequence into a hash table of length m = 11 with the hash function **`h(key) = key mod m`**, show the content of the hash table after inserting all the values using the following collision resolution techniques.

(i) Separate chaining. [2 marks]

(ii) Quadratic probing. [2 marks]

(iii) What are the load factors of the two hash tables? If more keys are to be inserted, which collision resolution technique allows the load factor to become greater than 1? [2 marks]

14. An *m*-dimensional vector {v_0, v_1, …, v_{m-1}} is considered sparse if at most *n* (*n* << *m*) elements in it are non-zero. In practice, it is very inefficient to represent sparse vectors using an array of size *m* since most of the elements in the array would contain 0.

    Describe an alternative representation for a sparse vector which meets the following time complexity requirements: [6 marks]

Operation	Definition	Time Complexity
set(*i*, *value*)	Set the element at index *i* to be the given positive integer *value.*	O(*n*)
unset(*i*)	Set the element at index *i* in the vector to be 0.	O(1)
get(*i*)	Return the value of the element at index *i*.	O(1)
printInSequence()	Print the non-zero elements in the vector in ascending index order.	O(*n*)

    You need to justify in your answer how the time complexity requirements are satisfied.

Appendix A: java.util.ArrayList <E>

Constructors

`ArrayList``()` Constructs an empty list with an initial capacity of ten.
`ArrayList``(`**`Collection`**`<? extends `**`E`**`> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayList``(int initialCapacity)` Constructs an empty list with the specified initial capacity.

Methods

Modifier and Type	**Method and Description**
`boolean`	**`add`**`(`**`E`**` e)` Appends the specified element to the end of this list.
`void`	**`add`**`(int index, `**`E`**` element)` Inserts the specified element at the specified position in this list.
`void`	**`clear`**`()` Removes all of the elements from this list.
`boolean`	**`contains`**`(`**`Object`**` o)` Returns `true` if this list contains the specified element.
`E`	**`get`**`(int index)` Returns the element at the specified position in this list.
`int`	**`indexOf`**`(`**`Object`**` o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
`boolean`	**`isEmpty`**`()` Returns `true` if this list contains no elements.
`int`	**`lastIndexOf`**`(`**`Object`**` o)` Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
`E`	**`remove`**`(int index)` Removes the element at the specified position in this list.
`boolean`	**`remove`**`(`**`Object`**` o)` Removes the first occurrence of the specified element from this list, if it is present.
`boolean`	**`removeAll`**`(`**`Collection`**`<?> c)` Removes from this
`E`	**`set`**`(int index, `**`E`**` element)` Replaces the element at the specified position in this list with the specified element.
`int`	**`size`**`()` Returns the number of elements in this list.

**~~~ END OF PAPER ~~~**