NATIONAL UNIVERSITY OF SINGAPORE

CS1020 – DATA STRUCTURES AND ALGORITHMS I

(Semester 2: AY2014/15)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **THIRTEEN (13)** questions and comprises **THIRTEEN (13)** 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.
6. Please write legibly, or marks will be deducted for unclear writing.
7. The total marks for the paper is **80**.

SECTION A (6 Multiple Choice Questions: 18 marks)

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

1. What is the output of the following program?

```
class A {
  protected static int num1 = 3;
  protected int num2 = 5;

  public A() { num1++; num2++; }

  public A(int v) { num1 = v; num2 = v; }
}

class B extends A {
  protected int num2 = 9;

  public B() { super(); }

  public B(int v1, int v2) { super(v1 + v2); }

  public int getNum1() { return num1; }

  public int getNum2() { return num2; }
}

public class Q1 {
  public static void main(String[] args) {
    B b1 = new B();
    System.out.println(b1.getNum1() + ", " + b1.getNum2());

    B b2 = new B(11,12);
    System.out.println(b2.getNum1() + ", " + b2.getNum2());
  }
}
```

(A) 4, 9
23, 9

(B) 4, 10
23, 10

(C) 4, 6
23, 23

(D) 4, 9
0, 23

(E) None of the above.

2. Study the following 3 code fragments. The API of **ArrayList** class is given in Appendix B.

(i)

```
ArrayList<ArrayList<Integer>> outer =
                                new ArrayList<ArrayList<Integer>>();
int count = 0;
for (int i=0; i<2; i++) {
  ArrayList<Integer> inner = new ArrayList<Integer>();
  for (int j=0; j<3; j++) {
    count++;
    inner.add(count);
  }
  outer.add(inner);
}
System.out.println(outer);
```

(ii)

```
ArrayList<ArrayList<Integer>> outer =
                                new ArrayList<ArrayList<Integer>>();
ArrayList<Integer> inner = new ArrayList<Integer>();

int count = 0;
for (int i=0; i<2; i++) {
  for (int j=0; j<3; j++) {
    count++;
    inner.add(count);
  }
  outer.add(inner);
}
System.out.println(outer);
```

(iii)

```
ArrayList<ArrayList<Integer>> outer =
                                new ArrayList<ArrayList<Integer>>();
ArrayList<Integer> inner = new ArrayList<Integer>();

int count = 0;
for (int i=0; i<2; i++) {
  inner.clear();
  for (int j=0; j<3; j++) {
    count++;
    inner.add(count);
  }
  outer.add(inner);
}
System.out.println(outer);
```

2. (continue)

Which of the above 3 code fragments give(s) the following output?

```
[[4, 5, 6], [4, 5, 6]]
```

(A) Only (i)

(B) Only (ii)

(C) Only (iii)

(D) Only (i) and (iii)

(E) None of the above

3. What is the postfix expression for the infix expression `(a+b*(c-d)-e)`?

(A) `e-d-c*b+a`

(B) `cd-b*a+e-`

(C) `ab+cd-*e-`

(D) `abcd-*+e-`

(E) `ab*+cd-e-`

4. Suppose you have a stack in which the values 1 through 5 must be pushed on the stack in that order, but that an item on the stack can be popped and printed at any time. Which of the following outputs is impossible to be produced by any sequence of pushes and pops that follows the above constraints?

(A) `1 2 3 4 5`

(B) `5 4 3 2 1`

(C) `1 4 5 2 3`

(D) `1 3 5 4 2`

(E) None of the above

5. Which algorithm does the following **`mystery()`** method actually implement?

```
// maxVal() returns the maximum element value in the part of the
// array from a[startIndex] to a[endIndex], inclusive.

public static int maxVal(int[] a, int startIndex, int endIndex) {
   // Code not shown.
}


public static void mystery(int[] a) {
   for (int i = a.length-1; i >= 1; i--) {
      int max = maxVal(a, 0, i)
      a[i] = max;
   }
}
```

(A) Selection sort

(B) Insertion sort

(C) Bubble sort

(D) A correct sorting algorithm not shown in lecture

(E) An incorrect sorting method

6. Suppose we have the following recurrence relation (recursive definition) for $f(n)$:

$$f(n) = 1, \qquad \text{for } n = 0;$$

$$f(n) = 2f(n-1) + 3, \qquad \text{for } n > 0.$$

The closed-form formula for $f(n)$ is

(A) $2^n + 3$

(B) $2^{(n+2)} - 3$

(C) $n^2 + 1$

(D) $2n - 1$

(E) None of the above

SECTION B (62 marks)

7. [3 marks]

Using the definition of big O notation, show that $\log_{10} n = O(n^{0.01})$.

8. [6 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; }
}
```

Complete the following Java method **printCircularLinkedList()** that takes as input the reference to the head node of a circular linked list, and prints out all the data elements of the circular linked list exactly once. Assume that the **toString()** method of the data element class has already been properly defined.

```
public static
void printCircularLinkedList(ListNode <E> head) {
   // Fill in the code.
}
```

9. [8 marks]

The **ListNode** class is defined as usual as in Question 8.

Complete the following **recursive** Java method **reverseLinkedList()** that takes as input the reference to the head node of a linked list, reverses the linked list and returns the reference to the new head node. Your method should not create any new **ListNode** instance, and should not modify the data element in each **ListNode** object.

To reverse a linked list referenced by **head**, we call the method this way:

```
head = reverseLinkedList(head, null);
```

Marks will only be awarded for correct and meaningful use of recursion in your method.

```
public static
ListNode <E> reverseLinkedList(ListNode <E> head,
                               ListNode <E> revHead) {
   // Fill in the code.
}
```

10. [7 marks]

Complete the following Java method **printInOrder()** that takes as inputs an array **num** of **double**s and an array **rank** of **int**s, and prints out all the numbers in array **num** in *descending order*. The value of **rank[i]** is the *rank* of the corresponding element **num[i]** in array **num**. The largest element in array **num** has the rank 1, the second largest has rank 2, and so on. Assume all the numbers in array **num** are unique.

Your method should have linear time complexity (with respect to the length of array **num**), assuming that printing a number takes O(1) time. Your method should not modify the input arrays. You can create a temporary array in your method.

```
public static void printInOrder(double[] num, int[] rank) {
   // Fill in the code.
}
```

11. [10 marks]

For this question you are not supposed to use additional data structures other than the **LinkedList** class provided in the Java API, and only the **LinkedList**'s **`size()`**, **`offer()`**, **`poll()`**, and **`peek()`** methods are allowed.

Complete the following Java method **mergeSortedQueues()** that takes as input two sorted queues (where the smallest integer is at the front of each queue), and returns a new sorted queue that is the result of merging all the elements in the two input queues. The two input queues must be empty after the merging.

```
import java.util.*;

public static LinkedList<Integer> mergeSortedQueues(
   LinkedList<Integer> queue1, LinkedList<Integer> queue2) {
     // Fill in the code.
}
```

12. [20 marks]

A bead is a circular disc with a hole in it. The images on the right show a few examples.

Each bead can be represented by two circles: a bigger circle representing the disc and a smaller circle the hole.

Each circle is represented by its centre, which is a **Point** object, and its radius. The class **Circle** is given below:

```
import java.awt.Point;
import java.text.DecimalFormat;

class Circle {
  private Point _centre;
  private double _radius;
  private double _area;

  public Circle(int x, int y, double radius) {
    this(new Point(x, y), radius);
  }

  public Circle(Point centre, double radius) {
    _centre = centre;
    _radius = radius;
    _area = computeArea();
  }

  private double computeArea() {
    return Math.PI * _radius * _radius;
  }

  public Point getCentre() { return _centre; }

  public double getRadius() { return _radius; }

  public double getArea() { return _area; }

  public String toString() {
    DecimalFormat df = new DecimalFormat("0.00");
    return "(" + _centre.x + "," + _centre.y + ")|"
           + _radius + "|" + df.format(_area);
  }
}
```

The API of **Point** class is given in Appendix A.

12. (continue)

(a) A bead is considered valid if the hole is contained entirely inside the disc. The following shows two sample runs of a client program (which you do not need to write). The input data are shown in bold.

```
Enter x- and y-coord of centre and radius of disc: 1 2 13.5
Enter x- and y-coord of centre and radius of hole: 3 -4 2.8
Bead created: (1,2)|13.5|572.56^(3,-4)|2.8|24.63
```

```
Enter x- and y-coord of centre and radius of disc: 1 2 13.5
Enter x- and y-coord of centre and radius of hole: 12 2 3.1
Invalid bead
```

Complete the **Bead** class. You are not allowed to add any other methods. **[10 marks]**

```
import java.awt.Point;

class Bead {
  private Circle _disc, _hole;
  private double _area;        // area of the bead

  // discX, discY: x- and y-coordinates of centre of disc
  // discRad: radius of disc
  // holeX, holeY: x- and y-coordinates of centre of hole
  // holeRad: radius of hole
  public Bead(int discX, int discY, double discRad,
              int holeX, int holeY, double holeRad) {
    // Write a single statement below

  }

  public Bead(Circle disc, Circle hole) {
    // Fill in the code
  }

  public double getArea() {
    // Fill in the code
  }

  public String toString() {
    // Fill in the code
  }

  // Return true if this bead is valid,
  // i.e. the hole is inside the disc, or false otherwise.
  public boolean isValid() {
    // Fill in the code
  }
}
```

12. (continue)

(b) A program has been written to read in a list of beads' information and create an ArrayList of valid bead objects. The company wants to find out among all the pairs of beads, what is the largest difference in the areas of two beads. You may assume that the list has at least one bead. If there is only one bead in the list, the largest difference in the area of two beads is 0.

Write an efficient method **largestDifference(ArrayList<Bead> list)**.

The API of **ArrayList** class is given in Appendix B.

What is the efficiency of your method in big-O notation, assuming that *n* is the size of the list? **[8 marks]**

(c) The company wants to find out among all the pairs of beads, what is the smallest difference in the areas of two beads. Again, you may assume that the list has at least one bead, and if there is only one bead in the list, the smallest difference is 0.

Describe an efficient algorithm (you do not need to write any code) in one or two sentences. What is the efficiency of your algorithm in big-O notation, assuming that *n* is the size of the list? **[2 marks]**

13. [8 marks]

A 2D maze is represented as a *N* x *N* 2D array of **char**s, where each array element **maze[Y][X]** represents the location (*X*, *Y*) in the maze. Each location is marked with either an 'O' if it is an obstacle, or 'E' if it is empty space (not containing any obstacle). The *destination location* is the empty location at (0, 0) (therefore, `maze[0][0] == 'E'`). A *path* in the maze from location (*X1*, *Y1*) to location (*X2*, *Y2*), is a sequence of adjacent empty locations from (*X1*, *Y1*) to (*X2*, *Y2*), inclusive. Two locations are adjacent to each other only if they are connected and one is on the north, south, east, or west direction of the other.

Complete the following **recursive** method **reachable()** that takes as inputs a **maze**, the value ***N***, and a starting location (**locX**, **locY**), and returns **true** iff there exists a path from the starting location to the destination location (0, 0). Assume that the starting location is empty.

Marks will only be awarded for correct and meaningful use of recursion in your method. Your method is allowed to modify the content of **maze**, if that is necessary for the computation.

```
public static
boolean reachable(char[][] maze, int N, int locX, int locY) {
   // Fill in the code.
}
```

Appendix A: java.awt.Point

Attributes

```
public int x
public int y
```

Constructors

Point() Constructs and initializes a point at the origin (0, 0) of the coordinate space.
Point(int x, int y) Constructs and initializes a point at the specified (x,y) location in the coordinate space.
Point(**Point** p) Constructs and initializes a point with the same location as the specified Point object.

Methods

Modifier and Type	Method and Description
boolean	**equals**(Object obj) Determines whether or not two points are equal.
Point	**getLocation**() Returns the location of this point.
double	**getX**() Returns the X coordinate of this Point2D in double precision.
double	**getY**() Returns the Y coordinate of this Point2D in double precision.
void	**move**(int x, int y) Moves this point to the specified location in the (x,y) coordinate plane.
void	**setLocation**(double x, double y) Sets the location of this point to the specified double coordinates.
void	**setLocation**(int x, int y) Changes the point to have the specified location.
void	**setLocation**(**Point** p) Sets the location of the point to the specified location.
String	**toString**() Returns a string representation of this point and its location in the (x,y) coordinate space.
void	**translate**(int dx, int dy) Translates this point, at location (x,y), by dx along the x axis and dy along the y axis so that it now represents the point (x+dx,y+dy).

Appendix B: 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.
boolean	**addAll**(**Collection**<? extends **E**> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(**Object** o) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
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.
Iterator<**E**>	**iterator**() Returns an iterator over the elements in this list in proper sequence.
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.
ListIterator<**E**>	**listIterator**() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<**E**>	**listIterator**(int index)

	Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
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
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
boolean	**retainAll**(**Collection**<?> c) Retains only the elements in this list that are contained in the specified collection.
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.
List<**E**>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

~~~ END OF PAPER ~~~