NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR CS1020
Semester 2: AY2012/2013

CS1020 – Data Structures and Algorithms I

April 2013 Time allowed: 2 hours

Matriculation number:									

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **FIFTEEN (15)** questions and comprises **TWENTY THREE (23)** 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. Fill in your **Matriculation Number** above clearly with a pen.
4. Answer all questions.
5. For MCQs (Q1 to Q8), use the OCR form provided. Shade and write down your matriculation number on the OCR form. You must use 2B pencil to shade/write on the OCR form.
6. For short questions (Q9 to Q15), write your answers in the space provided. You may use pencil to write your answers.
7. You must submit both the OCR form and this document. It is your responsibility to ensure that you have submitted both to the invigilator at the end of the examination.

FOR EXAMINER'S USE ONLY			
Section/Questions	**Possible**	**Marks**	**Check**
A. MCQs 1 – 8	**24**		
B. Q 9	**7**		
B. Q 10	**6**		
B. Q 11	**14**		
B. Q 12	**12**		
B. Q 13	**12**		
B. Q 14	**16**		
B. Q 15	**9**		
Total	**100**		

SECTION A (8 Multiple Choice Questions: 24 marks)

Each question has one correct answer. Shade your answers on the OCR form. Three (3) marks are awarded for each correct answer; no penalty for wrong answer.

1. What is the output of the following program?

```
   Queue q = new Queue();
   Stack s = new Stack();
   s.push(new Integer(5));
   s.push(new Integer(6));
   s.push(s.peek());
   s.push(new Integer(7));
   q.enqueue(s.pop());
   q.enqueue(new Integer(5));
   q.enqueue(new Integer(6));
   System.out.print(q.peek());
   s.push(q.dequeue());
   System.out.print(s.pop());
   s.pop();
   System.out.print(s.pop());
```

   A. **567**
   B. **765**
   C. **775**
   D. **766**
   E. None of the above

2. Which of the following is true about arithmetic expressions?

   A. Parentheses are needed in all arithmetic expressions.
   B. Infix expressions do not require precedence rules.
   C. Postfix and prefix expressions do not require precedence rules.
   D. Fully parenthesized expressions are difficult to recognize and evaluate.
   E. None of the above.

3. Consider the implementation of the Queue using an array. What would go wrong if we try to keep all the items at the front of a partially-filled array (so that data[0] is always the front of the queue)?

   A. The constructor would require linear time.
   B. The offer (enqueue) method would require linear time.
   C. The isEmpty method would require linear time.
   D. The poll (dequeue) method would require linear time.
   E. The peek method would require linear time.

4. An efficient algorithm to determine if an array *A* with *n* elements contains duplicate elements is:

   A. O(*n*) whether or not *A* is sorted.
   B. $O(n^2)$ whether or not *A* is sorted.
   C. $O(\log_2 n)$ if *A* is sorted, or $O(n \log_2 n)$ otherwise.
   D. O(*n*) if *A* is sorted, or $O(n \log_2 n)$ otherwise.
   E. O(*n*) if *A* is sorted, or $O(n^2)$ otherwise.

5. Which of the following sorting algorithms (as introduced in lectures) is both in-place and stable?

   A. Selection sort
   B. Merge sort
   C. Quick sort
   D. Radix sort
   E. None of the above.

6. Mr. Qian decides to use a circular linked list to represent FIFO queue. He lets a pointer variable Q point to the node on the circular list that contains the item at the front of the queue. Ms. Hou thinks that a better design would be to have Q points to the node on the circular linked list that contains the rear item in the queue instead.

   Is Mr. Qian's approach or Ms. Hou's approach better?

   A. Mr. Qian's approach is better.
   B. Ms. Hou's approach is better.
   C. Both are the same.
   D. Both are no good. Instead, Q should point to the middle node of the linked list.
   E. Circular linked list cannot be used to represent FIFO queue.

7. What does the following method compute, assuming that **a** and **b** are positive integers?

```
// Pre-cond: a, b are positive integers
public static int m(int a, int b) {
    if (a < b) return 0;
    else return 1 + m(a–b, b);
}
```

   A. m(a, b) returns a – b
   B. m(a, b) returns a % b
   C. m(a, b) returns a / b
   D. m(a, b) returns a * b
   E. None of the above.

8. What is the output of the following code?

```
class Guess {
    private static int value1 = 64;
    private int value2;

    public void update() {
        value1++; value2 += 3;
    }

    public void update(int v) {
        value1++; value2 += v;
    }

    public void update(int v1, int v2) {
        value1 += v1; value2 += v2;
    }

    public int getValue1() { return value1; }

    public int getValue2() { return value2; }
}

class DemoGuess {
    public static void main(String[] args) {
        Guess guess1 = new Guess();
        Guess guess2 = new Guess();

        guess1.update(20);
        guess2.update();
        guess2.update(4,7);
        System.out.println(guess1.getValue1() + " " +
                           guess1.getValue2() + " " +
                           guess2.getValue1() + " " +
                           guess2.getValue2());
    }
}
```

A. **70 20 70 10**
B. **65 20 65 10**
C. **69 10 73 20**
D. **73 10 73 20**
E. **84 10 84 10**

SECTION B (7 Short Questions: 76 marks)

9. **[7 marks]**
Given the following code fragment:

```
for (int i = 1; i <= n; i = i*4) {
   System.out.println("loop 1");
   for (int j = 1; j <= i; j = j + 2) {
      System.out.println("loop 2");
   }
}
```

(a) If n = 64, what is the number of times "loop 1" is printed? [1 mark]

(b) If n = 64, what is the number of times "loop 2" is printed? [2 marks]

(c) What is the running-time complexity of the code in terms of n? [4 marks]

10. **[6 marks]**
A string can contain only letters ‘A’ and ‘X’. However, it cannot contain two ‘X’ next to each other. For example, the string “AAAXAXAA” is valid, but “XAXXA” is not.

Write a recursive method

int numOfStrings(int n)

to return the number of valid strings of length n, where n is a positive integer.

For example, numOfStrings(3) returns 5 because there are 5 valid strings of length 3, namely, “AAA”, “AAX”, “AXA”, “XAA”, and “XAX”.

No mark will be awarded if recursion is not used.

```
int numOfStrings(int n) {













}
```

11. **[14 marks]**

(a) In the infix-to-postfix expression conversion algorithm, given an infix expression:

a + (b – c) – d * ((e – f) / g + h)

(i) Show the stack contents and the current partial postfix expression output when the infix expression is read and processed up till the token "f". [4 marks]

(ii) Continuing the processing till the end of the infix expression, what is the postfix expression of the whole infix expression? [2 marks]

The stack content after token "f" is processed:

The current partial postfix expression output is:

The postfix expression of the whole infix expression is:

This page is left blank intentionally.

11. (continued…)

(b) Write an algorithm or a Java code fragment which implements a queue using two stacks S1 and S2. Both **isEmpty()** and **remove()** methods of the queue implementation must have the complexity of O(1).

(i) Write out the algorithm or code fragment for the **isEmpty()**, **remove()**, and **offer()** methods. [6 marks]

(ii) Show the contents of S1 and S2 after performing the following operations:

offer(1); offer(2); offer(3); remove(); remove(); offer(4); offer(5) [2 marks]

This page is left blank intentionally.

12. **[12 marks]**

Shell sort is a sorting algorithm based on **insertion sort**. It improves insertion sort by comparing and swapping elements separated by a gap of several positions. In this way, out-of-place elements which are far apart can be moved into position faster than merely comparing neighbouring elements. The process of Shell sort consists of multiple passes. Each pass is associated with a fixed gap *h* and insertion sort is applied on all the subsequences formed by every *h*-th element of the sequence.

For example, a Shell sort with gaps 4 and 3 on an 8-element array is shown below. In the first pass, insertion sort is applied on 4 subsequences (a_0, a_4), (a_1, a_5), (a_2, a_6) and (a_3, a_7). In the second pass, insertion sort is applied on 3 subsequences (a_0, a_3, a_6), (a_1, a_4, a_7) and (a_2, a_5).

Array elements	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
Original array	53	17	28	93	15	25	7	47
After 1st pass (h = 4)	15	17	7	47	53	25	28	93
After 2nd pass (h = 3)	15	17	7	28	53	25	47	93

(a) Apply one pass of Shell sort on the given original array with *h* = 2. Show the resulting array and the total number of comparisons required for sorting (using insertion sort) each of the two subsequences. [4 marks]

For example, in insertion sort, given an array 6, 9, 12, 15, 10, 4, and if we are looking at where to put the value 10, the number of comparisons required to put it in the right place is 3.

Resulting array:

Total number of comparisons required for sorting (a_0, a_2, a_4, a_6):

Total number of comparisons required for sorting (a_1, a_3, a_5, a_7):

(b) What is the requirement on the gaps used such that Shell sort always yields an array that is completely sorted? [2 marks]

This page is left blank intentionally.

12. (continued…)
    (c) Write a method **void pass(int[] arr, int h)** which performs <u>one pass</u> of Shell sort on array **arr** with the gap **h**. [6 marks]

```
void pass(int[] arr, int h) {

}
```

This page is left blank intentionally.

13. **[12 marks]**

(a) Consider inserting the keys 14, 45, 28, 38, 1, and 61 in sequence into a hash table of length m = 11 with the following hash function:

h(key) = key mod m

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

h2(key) = (key mod 7) + 1

(i) Fill in the content of the hash table below after all the keys are inserted into it. [2 marks]

0	1	2	3	4	5	6	7	8	9	10

(ii) We then delete the keys 45 and 1. Fill in the content of the hash table below after the deletion and the probe sequence for deleting 1. [3 marks]

0	1	2	3	4	5	6	7	8	9	10

Probe sequence for deleting 1:

(iii) We then insert the key 27. Fill in the content of the hash table below after the insertion. [1 mark]

0	1	2	3	4	5	6	7	8	9	10

(iv) We then search for the key 46. What is the probe sequence for this search? [2 marks]

Probe sequence for searching for 46:

This page is left blank intentionally.

13. (continued…)
    (b) Consider inserting the keys 68, 29, 3, 42, 16, and 55 in sequence into a hash table of length m = 13 with the following hash function:

    h(key) = key mod m

    (i) Show the content of the hash table after inserting all the keys using **linear probing** for collision resolutions. [1 mark]

0	1	2	3	4	5	6	7	8	9	10	11	12

    (ii) Show the content of the hash table after inserting all the keys using **quadratic probing** for collision resolutions. [2 marks]

0	1	2	3	4	5	6	7	8	9	10	11	12

    (iii) Which collision resolution technique works better for this input sequence and why? [1 mark]

This page is left blank intentionally.

14. **[16 marks]**
    Study the program below.

```
import java.util.*;

class Matrix1 {
   public static void main(String[] args) {
      int[][] mtx = {{1,2,3,4},{2,4,6,8},{3,6,9,12},{4,8,12,16}};
      int[] arr = aMethod(mtx, mtx.length);
      System.out.println(Arrays.toString(arr));
   }

   public static int[] aMethod(int[][] mtx, int n) {
      int[] mystery = new int[2*n-1];
      for (int k = 0; k < 2*n-1; k++) {
         int sum = 0;
         for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                 if (i + j == k)
                     sum += mtx[i][j];
            }
         }
         mystery[k] = sum;
      }
      return mystery;
   }
}
```

(a) Circle the elements in the two-dimensional array **mtx** that are used to compute the corresponding entry in the **mystery** array. For example, for mystery[0], the only element in **mtx** that is used to compute it is mtx[0][0] and it has been circled for you. [3 marks]

$$\begin{bmatrix} \textcircled{1} & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[0]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[1]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[2]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[3]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[4]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[5]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$
mystery[6]

(b) What is the output of the program? [3 marks]

This page is left blank intentionally.

14. (continued…)

(c) A careful observation shows that it is possible to obtain the results stored in the **mystery** array by shifting the rows of the matrix before adding them up. We will write another program to do this. The basic idea is to read in <u>one row of the matrix at a time</u>, perform the necessary shifting operation, then add the row to the accumulated sum thus far. Complete the code below that uses two linked lists, **row** and **sum**, to represent the row read in and the accumulated sum thus far, respectively. You may create additional linked list if necessary. [10 marks]

```
import java.util.*;

class Matrix2 {
  public static void main (String[] args) {
    Scanner sc = new Scanner(System.in);
    LinkedList<Integer> row = new LinkedList<Integer>();
    LinkedList<Integer> sum = new LinkedList<Integer>();

    System.out.println("Input number of rows ");
    int numRow = sc.nextInt();
    for (int i = 0; i < numRow; i++) {



    }
    System.out.println(sum);
  }
}
```

This page is left blank intentionally.

15. **[9 marks]**
A word-ladder contains connections from one word to another word by changing only one letter at a time. Each of the words in the word-ladder should be a word from a dictionary of acceptable words. For example, from COLD to WARM, a few possible word-ladders are shown below. The letter that is changed is shown bold and underlined.

COLD → CO**<u>R</u>**D → C**<u>A</u>**RD → **<u>W</u>**ARD → WAR**<u>M</u>**
COLD → CO**<u>R</u>**D → COR**<u>M</u>** → **<u>W</u>**ORM → W**<u>A</u>**RM
COLD → **<u>W</u>**OLD → WO**<u>R</u>**D → W**<u>A</u>**RD → WAR**<u>M</u>**

Describe an algorithm for finding a word-ladder given a starting word, an ending word, and a dictionary of acceptable words. Your algorithm should make use of one or more data structures covered in this module. The first two steps of the algorithm are given. [9 marks]

Step 1: Store all of the words from the dictionary in a linked list L.

Step 2: Get the starting word, and search through L to find all words that are one letter different from the starting word.

=== END OF PAPER ===