NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
EXAMINATION FOR CS1020
Semester 2: AY2011/12

CS1020 – Data Structures and Algorithms I

April 2012 Time allowed: 2 hours

Matriculation number: | | | | | | | | | |

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **SIXTEEN (16)** questions and comprises **EIGHTEEN (18)** 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 Q9), 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 (Q10 to Q16), 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.

EXAMINER'S USE ONLY			
Section / Question	**Possible**	**Marks**	**Check**
A. MCQs 1-9	**27**		
B. Q 10	**8**		
B. Q 11	**5**		
B. Q 12	**13**		
B. Q 13	**15**		
B. Q 14	**12**		
B. Q 15	**12**		
B. Q 16	**8**		
Total	**100**		

SECTION B (7 Short Questions: 73 Marks)

10. **[8 marks]**

(a) What is the time complexity (in big-O notation) of the following recursive code? [4 marks]

```
public static int xyz(int n) {
   if (n <= 2) return n;
   int sum = 0;
   for (int j=1; j<n; j *= 2)
      sum += j;
   for (int k=n; k>1; k /= 2)
      sum += k;
   return xyz(n - 1) + sum;
}
```

10. (continued...)

(b) What is the time complexity (in big-O notation) of the following recursive code when **proc(A, 0, n-1)** is called with n > 0? [4 marks]

```
public static void proc(int[] A, int start, int end) {
   if (start < end) {
      int mid1 = (start*2 + end)/3;
      int mid2 = (start + end*2)/3;

      proc(A, start, mid1);
      proc(A, mid1 + 1, mid2);
      proc(A, mid2 + 1, end);

      for (int i=start; i<end; i++)
         A[i]++;

      int sum=0;
      for (int j=1; j<10000; j *= 2)
         sum += j;

      for (int k=10000; k>0; k /= 4)
         sum -= k;
   }
}
```

11. **[5 marks]**
Consider the infix-to-postfix algorithm using a single stack. Suppose at a certain point in the algorithm, the states of the operator stack, postfix string output, and the remaining input are as follows:

Operator stack

Postfix output so far: "**a b @ c d e**"

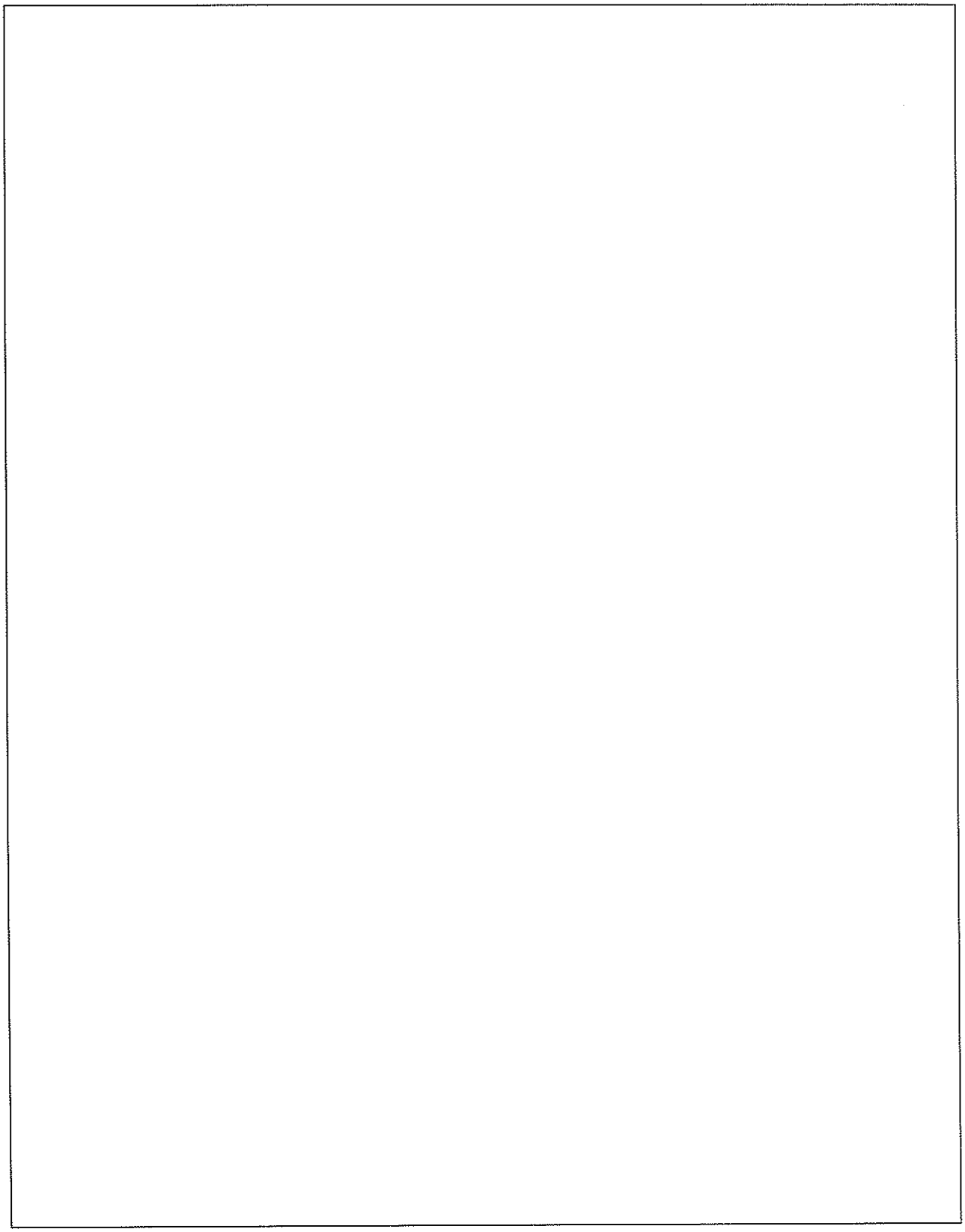
Remaining infix input: "**+ f) / g**"

where @ is the unary minus operator. What are the input infix expression and the final output postfix expression?

Infix expression:

Final output postfix expression:

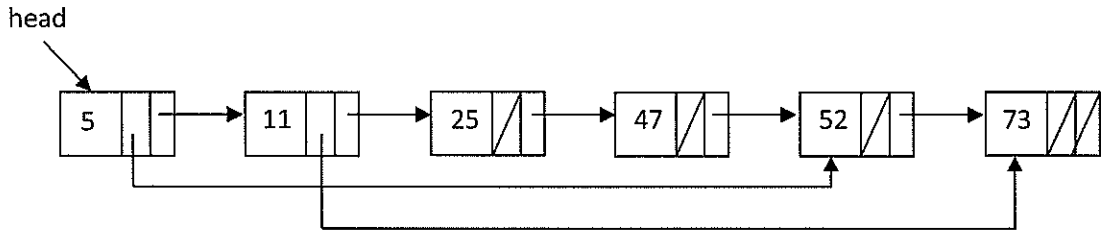
12. **[13 marks]**
Design an efficient algorithm to support methods in **SecondMaxStack** that include the ordinary stack operations i.e. push, pop, peek, and isEmpty operations, and an additional operation called **SecondMax** that will return the second largest of all the items in **SecondMaxStack**. All operations are to be performed in constant time. You may use up to two stacks to implement **SecondMaxStack**. You can express your algorithm by a pseudo-code or a Java program.



13. **[15 marks]**
Assume the following KListNode declaration:

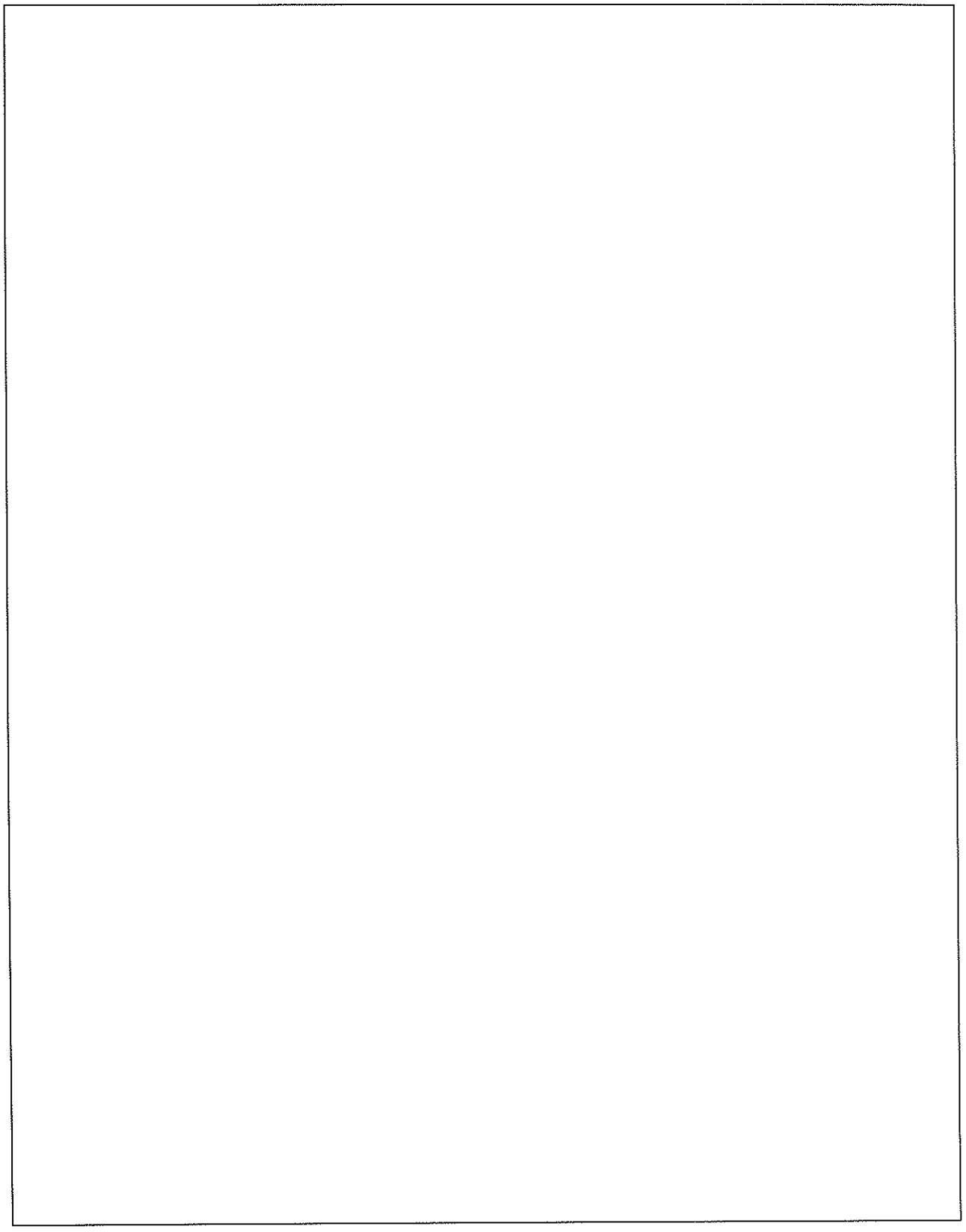
```
class KListNode {
   private int _item;          // value in the node
   private KListNode _kNext;   // points to a node K steps ...
                               //    down the list
   private KListNode _next;    // the usual next pointer

   public int getItem() {return _item;}
   public KListNode getKNext() {return _kNext;}
   public KListNode getNext() {return _next;}

   // constructors and other accessors and mutators not listed here
}
```

Each node in a LinkedList created from KListNode class has two pointers: one points to the next node and the other points to a node that is k steps down the list. In the following example, k = 4.



(a) Searching on a linked list is normally a O(n) operation as we need to perform linear search even if the values are sorted. With the _kNext pointer, the efficiency for searching on a sorted KLinkedList can be improved.

Write a static method to search for a value given a head pointer pointing to the first node of the list and an integer value. The method should return the reference to the node containing the value or return null if the value is not found in the list. [9 marks]



13. (continued...)

(b) While the _kNext pointers improve the efficiency for searching on a sorted linked list, it would require more codes for inserting node to and removing node from the list. In particular, more nodes need to be modified when either operation is executed.

(i) What are the minimum and maximum numbers of nodes affected when a node is inserted into such a linked list? That is, how many _next pointers and _kNext pointers need to be changed? [3 marks]

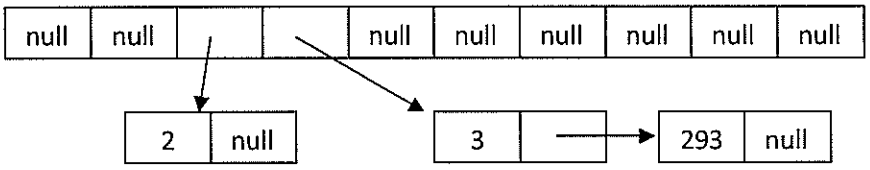
(ii) What are the minimum and maximum numbers of nodes affected when a node is removed from such a linked list? That is, how many _next pointers and _kNext pointers need to be changed? [3 marks]

14. **[12 marks]**

(a) Suppose a badly designed hash table HT has 10 entries. Keys are non-negative integers and the hash function is as follows:

```
public int hash(int key) {
   return key % 10;
}
```

Collisions are handled with the separate chaining collision resolution technique (with new values added at the head of the chain). Suppose that the state of the hash table is currently represented as follows:



(i) Draw (using the same drawing conventions) the state of the hash table after the following operations: [3 marks]

HT.insert(17);
HT.insert(207);
HT.insert(17);
HT.insert(208);
HT.insert(33);

14. (continued...)

(ii) Give a sequence of 100 keys to insert that would result in the worst-case performance when accessing data. You may use "..." if your pattern is clear.[3 marks]

(iii) Suggest some techniques that would improve the design of the hash table. [3 marks]

(b) Consider a (better-designed) hash table HT2, which is being used in the following code (where the integer method "size" returns the number of items stored in the hash table and the boolean method "containsKey" returns true or false depending on whether the key is in the hash table, and "random" returns a random number between 0 and size * 10^6):

```
int counter = 0;
for (int i = 0; i < HT2.size(); i++) {
   if (HT2.containsKey(random()))
      counter++;
}
```

What is the best-case, average-case, and worst-case big-O running time? You may assume that the number of keys in HT2 is n. [3 marks]

15. **[12 marks]**

(a) The code for Insertion Sort is given below.

```
public static void insertionSort(int[] a) {
   for (int i=1; i<a.length; i++) {
      int next = a[i];
      int j;
      for (j=i-1; j>=0 && a[j]>next; j--)
         a[j+1] = a[j];
      a[j+1] = next;
   }
}
```

From the above code, the position to insert the element 'next' is found by scanning sequentially backwards. Could we improve the running time of Insertion Sort by using Binary Search instead to locate the position to insert 'next', since the part of the array to the left of 'next' is already sorted? Explain your answer. [4 marks]

15. (continued...)

(b) Suppose you are given a linked list with **head** pointing to its first node, and a **split()** method that splits the linked list into halves with the first node of the second half of the original list now pointed to by **mid**. The next pointer of the last node in the first half of the linked list is set to null.

The algorithm for **MergeSort** on this linked list is shown below.

```
MergeSort(Node head) {
        Node mid = split(head);
        Merge(MergeSort(head), MergeSort(mid));
}
```

Write an algorithm for the **Merge()** method. To get full credit, you must not create any additional linked list. [8 marks]

16. **[8 marks]**

Suppose you are given a very large integer array A with 1,000,000,000 elements, with elements sorted in descending order. However, only the first n elements contain data which are positive integers, but the value of n is unknown. The rest of the array elements contain zeros.

Write a method **search(A, k)** to search for a key k in array A. It returns the index of the array where k is found, or -1 otherwise. Your algorithm may use binarySearch(A, k, left, right) that searches for k in A[left] through A[right], assuming that the array is sorted in descending order. However, you do not need to write this Binary Search method.

You may also write other helper method(s) if necessary.

To obtain full credit, your **search(A, k)** method must run in $O(\log_2 n)$.

=== END OF PAPER ===