Student ID: ____________________ **Student Name:** ____________________

CS203 Data Structure and Algorithm Analysis **Quiz 1**

Note 1: Write all your solutions in the question paper directly. You can ask additional answer paper if necessary

Note 2: If a question asks you to design an algorithm, full marks will be given if your algorithm runs with optimal time complexity

Note 3: If a question asks you to design an algorithm, you should **first** describe your ideas in general words, **then** write the pseudocode, and **end** with time complexity analysis.

I. (5 marks) The time complexity of inserting an element to the head of a linked list is O(1).

II. (10 marks) Proof $n^3 + 25n^2 = O(n^3)$.

We need to find constants c1, c2 such that for all $n \geq c2$, it holds

$$n^3 + 25n^2 \leq c1 \cdot n^3$$

Towards this purpose, inspect the inequality:

$$\Leftrightarrow 25\, n^2 \leq (c1 - 1)\; n^3$$

$$\Leftrightarrow 25 \leq (c1 - 1)\, n.$$

By setting c1 = ___26___ and c2 = 1, the above holds. This concludes the proof.

III. (5 marks) Which of the following functions is $O(n \log \sqrt{n})$ (A)

A. $358 \cdot n\log_2 n$ B. $n^{1.2}/\log^5 n$ C. $(1.01)^n$ D. $n \cdot (\log_2 n)^{1.0003}$

IV. (5 marks) Stack s is initialized to empty, use s to convert the infix expression 6/2 +(8*9-3*5)/3 to equivalent postfix expression. In the process, when the 5 is scanned, what are the elements in the stack in order from the bottom of the stack to the top of the stack?

A. +(*- b. +(-* C. /+(*-* D. /+-*

V (5 marks) Suppose that we use an array to simulate a ring queue with size n, rear and front are the index value of the rear of queue and the front of queue, respectively. (1) If **rear == front**, the queue is empty. If **(rear+1)%n = front**, the queue is full.

VI [30 marks]

- Please write down the postfix of the expression: **5 * ((9 + 3) * (4*2) + 7)**
- **Ans: 5 9 3 + 4 2 * * 7 + * (5 marks)**
- Design an algorithm to evaluate the above postfix expression (You can omit time complexity analysis in this problem). Then, show the running steps of your algorithms for the following expression:

- **Idea: (5 marks)**

```
Read the tokens in one at a time
    If it is an operand, push it on the stack
    If it is a binary operator:
        pop top two elements from the stack,
        apply the operator,
        and push the result back on the stack
return the top of stack
```

Pseudocode: (10 marks)
Algorithm EvaluatePostfix(string postfix)

```
1.  n ← len(postfix),
2.  stack oprnds ← empty
3.  for i ← 0 to n-1
4.      token ← postfix [i]
5.      if (token is operand) then
6.              oprnds.push(token)
7.      else
8.              leftop ← oprnds.pop()
9.              rightop ← oprnds.pop()
10.             tmp ← leftop  token rightop
11.             oprnds.push(tmp)
12. return oprnds.top()
```

Running steps: (10 marks)

Stack operations	Stack elements
push(5)	5
push(9)	5 9
push(3)	5 9 3
push(pop() + pop())	5 12
push(4)	5 12 4
push(2)	5 12 4 2
push(pop() * pop())	5 12 8
push(pop() * pop())	5 96
push(7)	5 96 7
push(pop() + pop())	5 103
push(pop() * pop())	515

VII. (20 marks) Give you a size-n sorted array A (by ascending order), design an algorithm to count the number of ordered tuples (i, j, k) such that A[k]- A[i] <= m.

Idea: (8 marks)
For each element A[i], we need find A[k] in array A such that (1) A[k] – A[i] <= m, (2) there is A[j] between A[i] and A[k], it implies k – i >1.
Suppose the index x is the first element such that A[x]-A[i] > m, it means all elements from index i+1 to x-1 are satisfying (1). Thus, the number of ordered tuples is C_{x-i-1}^{2} for each (i, x) pair, i.e., choose arbitrary 2 elements, from A[i+1] to A[x-1], the small one is A[j] and the large one is A[k].
Hence, the total number of ordered tuples is the summation of C_{x-i-1}^{2} for each (i, x) pair.

Pseudocode: (7 marks)
Algorithm IXpair(Array A, n, m)

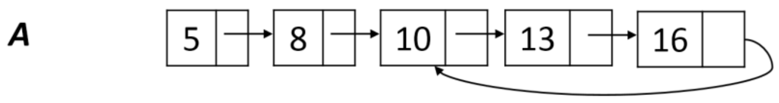
```
1.  Queue Q ← empty set
2.  Q.enqueue(A[1])
3.  Q.enqueue(A[2])
4.  sum ← 0
5.  for i ← 3 to n
6.      a ← A[i]
7.      while (a-Q.front() > m)
8.          s ← Q.size()-1
9.          p ← (s >= 2) ? $C_s^2$ : 0
10.         sum ← sum + p
11.         Q.dequque()
12.     Q.enqueue(a)
13. while(Q is not empty)
14.     s ← Q.size()-1
15.     p ← (s >= 2) ? $C_s^2$ : 0
16.     sum ← sum + p
17.     Q.dequque()
18. return sum
```

Time complexity analysis: (5 marks)
each element a in Array A enqueue once, it is O(n) cost,
meanwhile each element dequeue once, it is O(n) cost,
thus, the total cost is O(n)+O(n) = O(n).

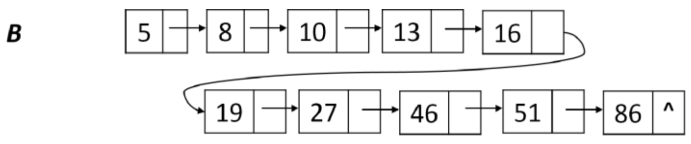
Please note binary search also can be used to find A[x] for each A[i].
But only 50% percentage marks will give to this kind of solutions.

VIII. (20 marks) Design an algorithm to check if a linked list is a circular linked list. For example:

Linked list **A** is a circular linked list, return Yes.



Linked list **B** is not a circular linked list, return No.



Idea: (5 marks)

We want to detect linked lists whether it is a circular linked list. How would we do that? We can do like this way: we use a slow runner (go one step per iteration) and a fast runner (go two steps per iteration). If the fast runner reaches null pointer (i.e., the end of this linked list), it returns No, as it is not circular linked list. If the fast runner collides the slow runner, it returns Yes, the linked list is a circular linked list.

If there is a circular in given linked list, can fast runner might "hop over" slow runner completely without ever colliding? The answer is NO. Suppose that fast runner did hop over slow runner, such that slow runner is a spot i and fast runner is at spot i+1. In previous step, slow runner is at i-1 and fast runner would at spot (i+1)-2, i.e., i-1 spot. That is, they would have collided.

Pseudocode: (7 marks)

Algorithm isCircular(linkedlistnode head)

```
1.  linkedlistnode slow ← head
2.  linkedlistnode fast ← head
3.  while(fast!=null && fast.next!=null)
4.  {
5.        slow ← slow.next;
6.        fast ← fast.next.next;
7.        if(slow==fast)
8.              return YES
9.  }
10. If(fast!=null && fast.next!=null)
11.       Return No
```

Time complexity analysis: (8 marks)

(1) If it is not a circular linked list, it costs O(n), where n is the size of linked list.

(2) If it is a circular linked list, what is the time complexity? It depends on when do fast and slow runner collide.

The analysis as follows:
Suppose the linked list has a "non-looped" part of size k, the circular portion has size LOOP_SIZE, the total linked list size is k + LOOP_SIZE = n.

We know every p steps of slow runner takes, fast runner has taken 2p steps. Therefore, when slow runner enters the circular portion after k steps, fast runner has taken 2k steps. In addition, fast runner must be 2k-k=k steps in the circular portion. Since k might be much larger than the LOOP_SIZE, we should actually write this as mod(k,LOOP_SIZE) steps, which we denote as K.

So now we know the following facts:

1. Slow runner is 0 steps into the circular
2. Fast runner is K steps into the circular
3. Slow runner is K steps behind fast runner
4. Fast runner is LOOP_SIZE – K steps behind slow runner
5. Fast runner catches up to slower at a rate of 1 step per unit of time.

So when do they meet? It is LOOP_SIZE-K steps by the facts 4 and 5 above.
Thus, the total cost is k + LOOP_SIZE-K = k + LOOP_SIZE – (k – M*LOOP_SIZE) = O(k+LOOP_SIZE) = O(n) as (k-M*LOOP_SIZE) > 0 .

Thus, the time complexity is O(n).