CS2040S TUTORIAL 3

Lucia Tirta Gunawan
luciatirtag@u.nus.edu

WELCOME TO THE THIRD TUTORIAL

THIS WEEK’S TOPIC IS…

LIST

STACK

QUEUE

Q1A. TRUE OR FALSE

Deletion in any Linked List can always be done in O(1) time.

Answer:

False. Deletion only O(1) at head or tail (for doubly linked list). O(n) otherwise.

Q1B. TRUE OR FALSE

A search operation in a Doubly Linked List will only take O(log n) time.

Answer:

False. Search is always O(n).

What if the elements in the doubly link list is sorted?

Still O(n) !!!

Even if it is sorted, we cannot search in O(log n) time using binary search since you cannot directly access a node at a particular index in O(1) time unlike the case of an array.

Q1C. TRUE OR FALSE

All operations in a stack are O(1) time when implemented using an array.

Answer:

True. On average, insertion is O(1) time, and the worst case individual insertions can be O(n) time due to resizing of the array.

Note that if we consider amortization, we can prove that a queue implemented with an array has worst case amortized O(1) time complexity for insertion.

For this course we will use the amortized time complexity rather than the worst case time complexity for array-based implementation of list/stack/queue

Q1D. TRUE OR FALSE

A stack can be implemented with a Singly Linked List with no tail reference with O(1) time for all operations.

Answer:

True. Insertion and deletion only required to be done at the head of the linked list.

Q1E. TRUE OR FALSE

All operations in a queue are O(1) time when implemented using a Doubly Linked List with no modification.

Answer:

True. Doubly linked list by default have tail reference.

Tail reference required to do insertion to the back in O(1) time.

Q1F. TRUE OR FALSE

Three items A, B, C are inserted (in this order) into an unknown data structure X. If the first element removed from X is B, X can be a queue.

Answer:

False. First element removed should be A if X is a queue.

Queue is FIFO (First In First Out)

Q2. CIRCULAR LINKED LIST

Implement a method swap(int index) in the CircularLinkedList class below to **swap the node at the given index with the next node.** The ListNode class (as given in the lectures) contains an integer value.

```
class CircularLinkedList {

    public int size;
    public ListNode head;
    public ListNode tail;

    public void addFirst(int element) {
        size++;
        head = new ListNode(element, head);
        if (tail == null)
            tail = head;
        tail.setNext(head);
    }

    public void swap(int index) { ... }
}
```

Constraints:

- Index ≥ 0
- If the index is larger than the size of the list, then the index wraps around.
- CANNOT create new nodes
- CANNOT modify element in any node

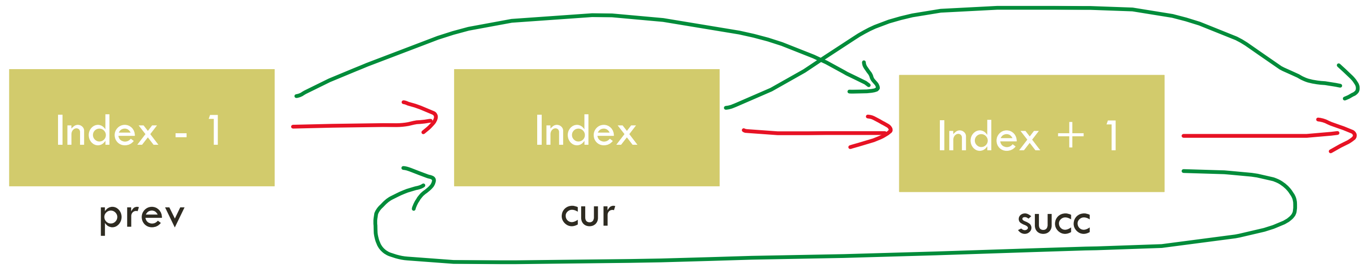
Objective: modify the pointers

Q2. CIRCULAR LINKED LIST

Case 1:
Size < 2
do nothing

Case 2:
size = 2
Swap head and tail

Case 3:
size > 2
use modulo operation to reduce the index (we don't have to iterate through the entire linked list multiple times).



Note that the order of changing the references matters as you may lose nodes if the references are not changed properly

```
curr.setNext(succ.getNext());
succ.setNext(curr);
prev.setNext(succ);
```

Q2. CIRCULAR LINKED LIST

```java
public void swap(int index) {
    if (size < 2)
        return; // if 0 or 1 nodes, don't bother

    if (size == 2) {
        ListNode newTail = head;
        head = tail;
        tail = newTail;
        return;
    } // if 2 nodes, only need to swap head and tail references

    // At this point, the list has >2 nodes
    index %= _size; // ensure that index < size of list first

    // get the 3 desired nodes
    ListNode prev = _tail;
    for (int loopIdx = 0; loopIdx < index; loopIdx++)
        prev = prev.getNext();
    ListNode curr = prev.getNext(); // curr now at indexed node
    ListNode succ = curr.getNext();

    // swap the 2 nodes: Note the order!
    curr.setNext(succ.getNext());
    succ.setNext(curr);
    prev.setNext(succ);

    if (index == 0) {
        head = succ; // head incorrect
    } else if (index == size - 2) {
        tail = curr; // swap(tail-1), tail incorrect
    } else if (index == size - 1) {
        head = curr; // swap(tail), both head & tail swapped
        tail = succ;
    }
}
```

Q3. WAITING QUEUE

Abridged problem description:

We have a queue that is implemented using array. Implement a function leave(String personName) that allows a person with name personName to leave the queue at any time.

Give at least 2 ways and state the time complexity

Q3. WAITING QUEUE

For all solutions, we need to maintain 2 variables front and back that denotes the start and end of the queue.

Solution 1:

Iterate from front to back.

If personName is found at index i, shift all elements after index i to the left by 1.

Time complexity: O(n)

Q3. WAITING QUEUE

Solution 2: Lazy deletion

Each element in the queue has a Boolean flag to indicate whether a person has left the queue or not. (we could also make a Person class with flag as one of its attributes)

The time complexity of the leave operation still O(n), since we need to iterate from front to back to find the person and flag it

However, the dequeue operation now could run in O(n) when everyone left the queue already but we need to dequeue the entire queue.

Q3. WAITING QUEUE

Solution 3:

Store the names of the people who want to leave the queue in a separate data structure (hash table).

When a person is served, the collection is searched to find a matching person.

The efficiency of leave() is improved to O(1), but the method requires more space. dequeue also deteriorates to O(n), as we may have to remove multiple elements until we find someone who has not already left the queue.

Q4. EXPRESSION EVALUATION

Abridged problem description:

There are only 4 operators +, -, *, / and the following rules apply

- (+ a b c) returns the sum of all the operands, and (+) returns 0.
- (- a b c) returns a - b - c - … and (- a) returns 0 - a. The minus operator must have at least one operand.
- (* a b c) returns the product of all the operands, and (*) returns 1.
- (/ a b c) returns a / b / c / … and (/ a) returns 1 / a, using double division. The divide operator must have at least one operand.

These are called Lisp expressions

Q4. EXPRESSION EVALUATION

Design and implement an algorithm that uses stacks to evaluate a legal Lisp expression with n tokens, each token separated by a space, all inputs are valid, no division by 0. Output the result, which will be one double value.

Q4. EXPRESSION EVALUATION

The algorithm require 2 stacks.

1. We start by pushing the tokens one by one into the first stack until we see the first ")"
2. We then pop tokens in the first stack and push them into the second stack one by one until we pop the element "(".
3. Now in the second stack, the operator is the first tokens to be removed followed by the tokens to be operated on, and we can remove the tokens inside one by one and evaluate the expression in the same order as they were given in the input. The result of the expression is pushed back into the first stack.
4. We repeat the above steps until all tokens are processed, and the final answer will be the one remaining value inside the first stack
5. The time complexity of the algorithm is O(n), because each token will only be added or removed from a stack not more than 4 times.

Q4. EXPRESSION EVALUATION

An example with the expression (+ (- 6) (* 2 3 4)). We denote the first stack as A and the second stack as B. In the diagrams below, the top of the stack is on the right.

1. The main stack pushes the tokens one by one until it reads “)”.

A:	(	+	(	-	6.0				

B:									

2. The main stack transfers its tokens to the temporary stack for evaluation.

A:	(	+							

B:	6.0	-							

Q4. EXPRESSION EVALUATION

3. The temporary stack pushes back the result after performing subtraction.

A:

(	+	-6.0						

B:

4. Main stack continues to push tokens until it reads ")".

A:

(	+	-6.0	(	*	2.0	3.0	4.0	

B:

5. Main stack transfers tokens to temporary stack one by one.

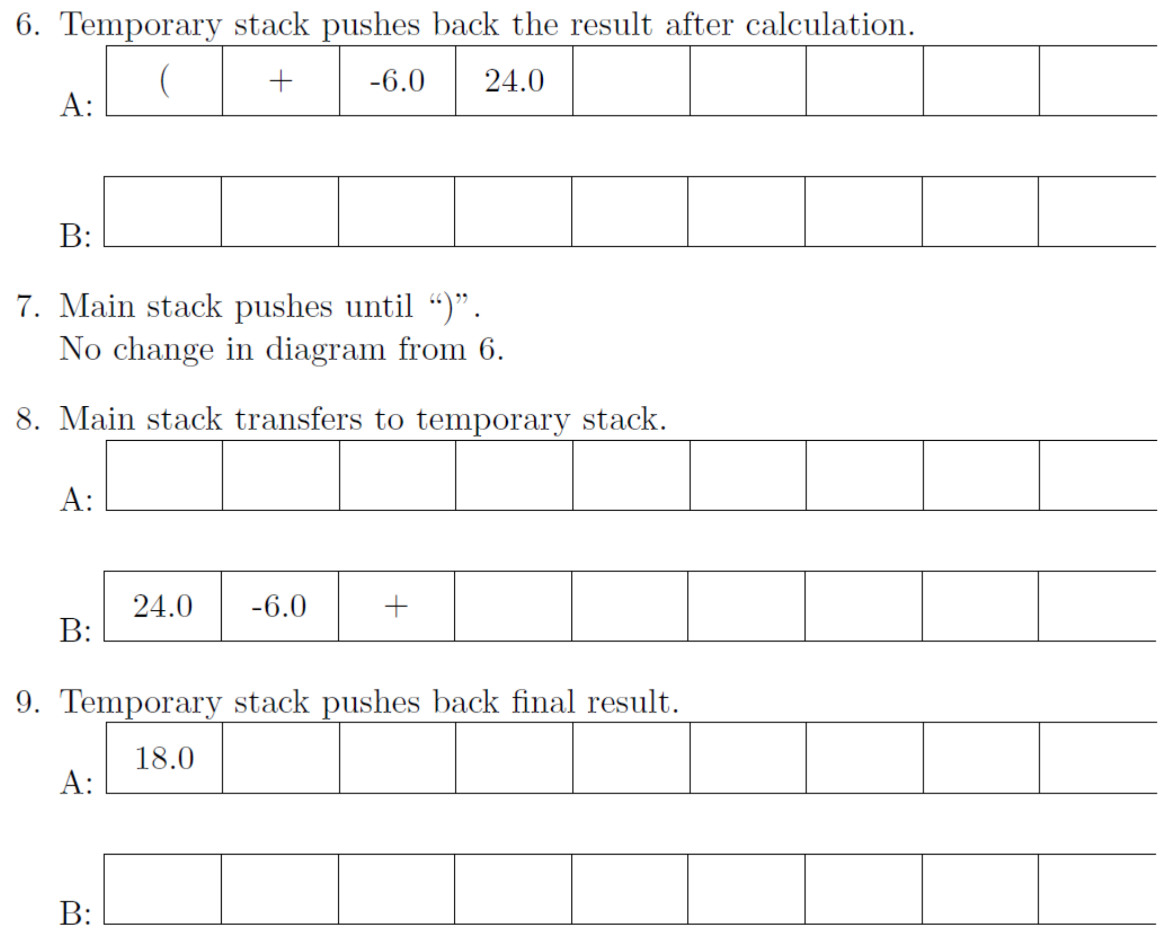
A:

(	+	-6.0						

B:

4.0	3.0	2.0	*					

Q4. EXPRESSION EVALUATION



6. Temporary stack pushes back the result after calculation.

A:

(	+	-6.0	24.0					

B:

7. Main stack pushes until “)”.
No change in diagram from 6.

8. Main stack transfers to temporary stack.

A:

B:

24.0	-6.0	+						

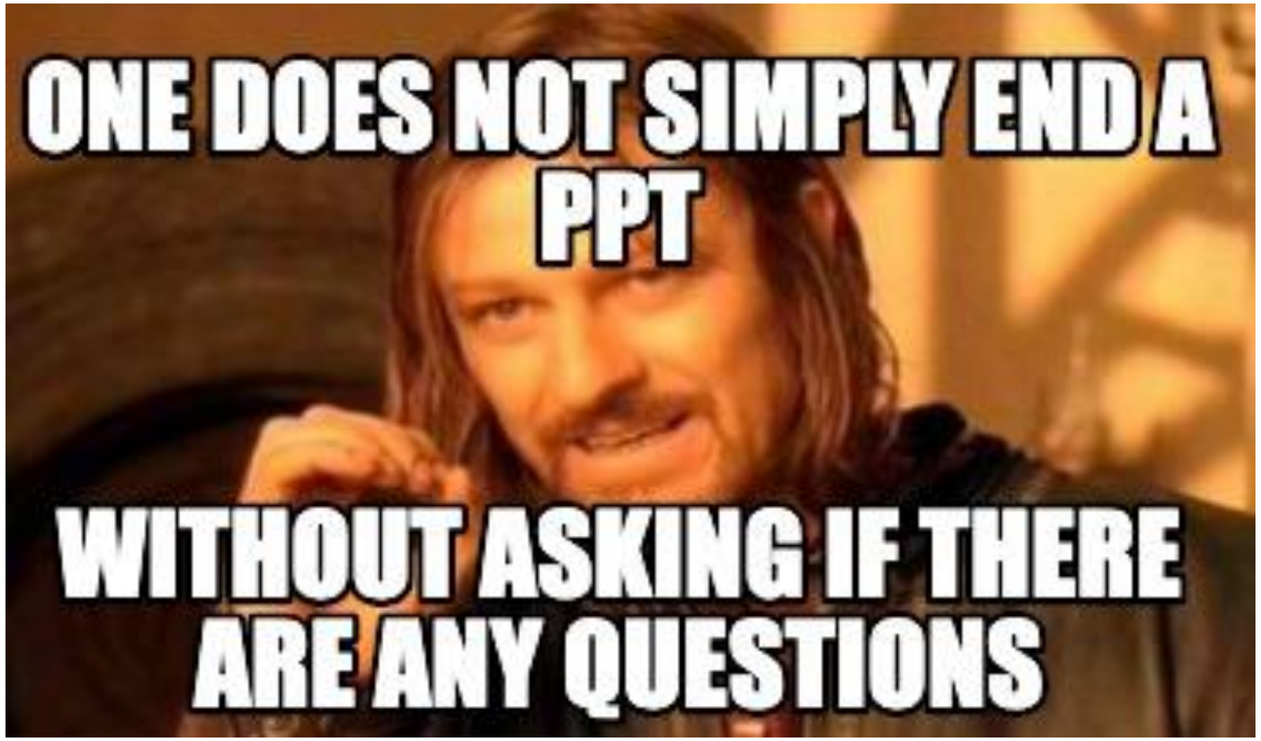
9. Temporary stack pushes back final result.

A:

18.0								

B:

ANY QUESTION?


ONE DOES NOT SIMPLY END A PPT
WITHOUT ASKING IF THERE ARE ANY QUESTIONS

SEE YOU IN THE NEXT TUTORIAL!