CSE12 Midterm Review

Pooja Bhat

Feedback Please! :)

https://docs.google.com/a/eng.ucsd.edu/forms/d/1rfe6mnYrINIfkWlmUnGAGY2RYTPPMSdK0YZ8Q_FMrjk/viewform?usp=send_form

Tips for midterm

- Time Management is crucial
  - Estimate and divide your time among different questions
  - Do not waste your time if you get stuck on a particular question. Revisit it later
  - Make sure you time the Practice Midterm to see what your current pace is and how much you can improve.
- No cheat sheets
- HW2 - very VERY important
- Practice writing code with pen and paper (VERY IMPORTANT)
- Go through all the review material available - True/False questions from HW1, HW3 (solutions on TEd), Extra credit problems from HW2, Interview questions from discussion slides.
- True/False, Multiple choice, Short answers and Coding
- MajorTopics

Consider a singly linked list with a head and tail pointer. Given that representation, which of the following operations could be implemented in O(1) time?

I. Insert item at the front of the list
II. Insert item at the rear of the list
III. Delete front item from list
IV. Delete rear item from list
V. Search for an item in the list

A. I and II
B. I and III
C. I, II, and III
D. I, II, and IV
E. all of them

Answer: C

Consider an implementation of a singly linked list with a head pointer only. Given the representation, which of the following operation can be implemented in O(1) time?

i) Insertion at the front of the linked list
ii) Insertion at the end of the linked list
iii) Deletion of the front node of the linked list
iv) Deletion of the last node of the linked list

a) I and II
b) I and III
c) I,II and III
d) I,II and IV

Answer: B

Consider an implementation of Circular Doubly Linked list with a head reference only. Given the representation, which of the following operation can be implemented in O(1) time?

I) Insertion at the front of the linked list
II) insertion at the end of the linked list
III) Deletion of the front node of the linked list
IV) Deletion of the end node of the linked list

A) I and II
B) I and III
C) I, II and III
D) I, II, III and IV

- Read the question carefully to see what data is given to you: a head, a tail or both.
- Know the complexity of Linked List operations for each type of Linked List

Constant time O(1) vs Linear time O(n) operations: Array

1. Add to the end (assume array is not full)
2. Add to the beginning (assume array is not empty)
3. Search for an item
4. Remove from the beginning (assume array has >1 item)
5. Remove from the end (assume non-empty array)

A. 1, 2, 5
B. 1, 5
C. 1, 2. 4, 5
D. 1, 2, 3, 4, 5

B. 1, 5 are constant time operations.

True or False

I decided to implement a stack using an array, where the top of the stack is always at array index 0. This will guarantee pop and push operations to run in O(1). (Do not worry about resizing).

A. True
B. False

Answer: False

Neither push nor pop is a constant-time operation (Why?)

Big-Theta complexity for the following code?

```
for (int j = 1 ; j < n ; j *= 2) {
        for (int i = 1; i < n; i++) {
                some_statement;
        }
}
```

A. O(n2)
B. O(n)
C. O(logn)
D. O(nlogn)

First for loop has complexity of $BT(\log_2 n)$

Second for loop has complexity of BT(n)

Thus, it has complexity of (nlog(n))

What is the content of myList?

MyLinkedList<String> myList = new MyLinkedList<String>();

myList.add(0, “blue”)
myList.add(1, “greener”)
myList.add(0, “I’m”)
myList.add(“than”)
myList.add(1, “so”)
myList.add(“purple”)
myList.add(3, “I’m”)

Write a method isIdentical that checks whether two LinkedLists are identical. Two lists are identical if they have the same contents. Assume that you have access to the head nodes of both lists, as well as a getter for the element in every node.

public boolean isIdentical(DoublyLinkedList12 list1, DoublyLinkedList12 list2) {

}

1) Does it work for empty lists?
2) Does it work if only one list is empty?
3) What if lists are not of equal length?
   Example:
   List1: 1->2->3->null
   List2: 1->2->3->4->null

True or False

Binary search is as efficient on linked lists as on arrays, provided the list is doubly linked.

A. True
B. False

B. False

Balanced Parentheses

Given an expression string, write a program to examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in expression.

For example, the program should print true for exp = “[()]{}{[()()]()}” and false for exp = “[(])”.

What data structure do you think will be most useful?

Sort a given LinkedList

Given a Singly linked list with each node containing values either 0, 1 or 2. Write a method to sort the list efficiently. Assume you have a reference to head. Each node has an integer data and a next reference.

Input : 1 -> 1 -> 2 -> 0 -> 2 -> 0 -> 1 -> 0

Output : 0 -> 0 -> 0 -> 1 -> 1 -> 1 -> 2 -> 2

public void sortAList(SinglyLinkedList myList)

The concatenation of two lists must be performed in O(1) time. Which of the following variation of a list can be used?

Assume head is given.

A. Singly linked list
B. Doubly linked list
C. Circular Singly Linked List
D. Circular doubly linked list
E. Array implementation of list

Answer: D

What if head and tail were given?

10. Assume that you are given a Stack ADT with push, and pop operations. Answer the questions referring to the code given below.

```
Stack s1 = new Stack(8);
s1.push("A");
s1.push("B");
s1.push("C");
s1.push("D");
s.push(“E”)
Stack s2 = new Stack(8);
for (int i=0; i<3; i++) {
        s2.push(s1.pop());
}
Stack s3 = new Stack(8);
while (s2.size() > 0) {
        s3.push(s2.pop());
 }
while (s3.size() > 0) {
        s1.push(s3.pop());
}
```

a) What is the size of the stack s1 after the code executes?
b) What is the size of s2 after code executes?
c) What is the size of s3 after code executes?
d) What is the top of the stack s1 after the code executes?
e) What is the top of the stack s2 after the code executes?

Answers:

a) 5
b) 0
c) 0
d) C
e) Stack is empty

```
public boolean isIdentical(DoublyLinkedList12 list1, DoublyLinkedList12 list2) {

        Node  ptr1 = list1.head;
        Node ptr2 = list2.head;
        while(ptr1!=null && ptr2!=null) {
                if(ptr1.getElement() != ptr2.getElement())
                        return false;
                ptr1 = ptr1.next;
                ptr2 = ptr2.next
        }
        if(ptr1 != ptr2)
                return false;
        return true;
}
```

Intersection of lists

Given two SORTED singly-linked lists with head, write an algorithm that outputs the intersection of the two lists. The output list should also be sorted.

E.g.

List 1: 2 -> 5 -> 6 -> 9 -> 10
List 2: 1-> 2 -> 7 -> 10 -> 11 -> 11
Output: 2 -> 10

Possible test cases: What if one of the list is empty? What if there are no common elements? What if the two lists are identical?

Note: An $O(n^2)$ solution is correct, but not efficient. For full credit, you need to come up with the most efficient solution, O(n) in this case

Consider the following method in Doubly Linked List class. If the input list is 1-2-3-4-5, what is the output list after this method executes? Assume list is not empty

```
public void magic() {
    Node temp = head;
    while(temp != tail) {
            E val = temp.data;
            temp.data = temp.next.data;
            temp.next.data = val;
            temp = temp.next;
    }
}
```

Find out if a given linked list is a singly-linked list or has a cycle in it? (In a list with cycle, the last node may point to ANY other node in the list)

- Pretty useful technique - often called as the ‘hare’ and ‘tortoise’ algorithm for cycle-detection
- Use 2 pointers: a fast pointer and a slow pointer. In each iteration, move the slow pointer by a node and the fast pointer by 2 nodes. If there is no cycle, the fast pointer will eventually reach the end (will be either null or point to null). If there is a cycle, the fast pointer and the slow pointer will eventually meet.

```
boolean hasLoop(Node head) {
        if(head== null) // list does not exist..so no loop either.
                return false;
        Node slow, fast; // create two references.
       slow = fast = head; // make both refer to the start of the list.
       while(true) {
                slow = slow.next;          // 1 hop.
       if(fast.next != null)
          fast = fast.next.next; // 2 hops.
      else
          return false;          // next node null => no loop.
       if(slow == null || fast == null) // if either hits null..no loop.
          return false;
       if(slow == fast) // if the two ever meet...we must have a loop.
          return true;
   }
}
```

Write a method called FindMid that returns the value of the node at the middle of a doubly linked list without using any form of counter. Assume that you are given the Node and DLList structures (no iterators) and their associated methods.

Also, assume that the total number of nodes in the list is odd.

HINT: Can you use your hare and tortoise algorithm here?

Union of lists

Given two SORTED singly-linked lists with a head, write a method that generates the union of two lists (without duplicates). The output list should also be sorted.

List 1: 2 -> 5 -> 6 -> 9 -> 10
List 2: 1-> 2 -> 7 -> 10 -> 11 -> 11
Output: 1 -> 2 -> 5 -> 6 -> 7 -> 9 -> 10 -> 11

public void findUnion(SinglyLinkedList list1, SinglyLinkedList list2, SinglyLinkedList union)
Possible test cases: What if one of the list is empty? What if the two lists are not of equal lengths? What if they are?
Note: An $O(n^2)$ solution is correct, but not efficient. For full credit, you need to come up with the most efficient solution, O(n) in this case

Generate Binary Numbers from 1 to n

Given a number n, write a method that generates and prints all binary numbers with decimal values from 1 to n.

Examples:

Input: n = 2
Output: 1, 10

Input: n = 5
Output: 1, 10, 11, 100, 101

Generate Binary Numbers from 1 to n

1) Create an empty queue of strings
2) Enqueue the first binary number “1” to queue.
3) Now run a loop for generating and printing n binary numbers.
    a) Dequeue and Print the front of queue.
    b) Append “0” at the end of front item and enqueue it.
    c) Append “1” at the end of front item and enqueue it.