CSE 214

Practice Midterm 1 (Problems)

[This practice test does not imply that the original midterm question will be of the same format]

(1) Write true or false for the following statements:
   a. $15n + 20n^2 - 10000 = O(n^2)$ [true]
   b. $2nlogn + 4n + logn = O(n)$ [false]
   c. $2^n + 100^n + logn = O(100^n)$ [true]

(2) You are given a binary tree with n nodes. You are asked to delete two nodes from the given tree. After deletion, how many edges are left in the updated tree?
   a. n
   b. n - 1
   c. n - 2
   d. n - 3

(3) If you are given a list of numbers and asked to make a binary search tree using them, which of the following traversal on that BST will give you the sorted sequence of those numbers?
   a. preorder
   b. inorder
   c. postorder
   d. level order

(4) Consider the following code fragment

```
int sum = 0;
for(int i = 0; i <= n; i++)
{
      for(int j = 0; j <= n; j++)
            sum = sum + j;
      for(int k = 1; k <= n; k*=2)
            sum = sum * k;
}
```

What is the total number of operations that occur in terms of input size n in closed form (consider each for() as a single operation). What is the time complexity in big O-notation?

Answer:

Line 1: 1

Line 2: n + 2

Line 4: (n+1) * (n+2)

Line 5: (n+1) * (n+1)

Line 6: (n+1) * ($\log_2$(n)+2) [as the 2nd inner for loop runs for 1, 2, 4, 8, and so on]

Line 7: (n+1) * ($\log_2$(n)+1)

Total time: 1 + (n+2) + (n+1)*(n+2) + (n+1)*(n+1) + (n+1)*($\log_2$(n)+2) + (n+1)*($\log_2$(n)+1)

= $2n^2 + 2n\log_2(n) + 2\log_2(n) + 9n + 9$

Big O-notation: $O(n^2)$

(5) You are given a linked list class as follows, write a method middleElement() which takes the head of the linked list as parameter and returns the middle element (just the data) without deleting the element with a **single pass** through the list. If there are even number of elements, then there are two middle elements, return the second middle element. What is the time complexity of your middleElement method in big O-notation?

```
class LinkedList
{
      Node head;  // head of list

      /* Linked list Node*/
      private class Node
      {
            String data;
            Node next;
            Node(String d)   { data = d; next = null; }
      }
}
```

Answer:

```
public String middleElement(Node head)
  {
      Node slow_ptr = head;
      Node fast_ptr = head;
      if (head != null)
      {
          while (fast_ptr != null && fast_ptr.next != null)
          {
              fast_ptr = fast_ptr.next.next;
              slow_ptr = slow_ptr.next;
          }
          return slow_ptr.data;
      }
      else
        return null;
  }
```

(6) Complete the following method that reverses the elements of an integer queue recursively. By reversing the queue, we mean that the first element becomes the last, the 2nd element becomes the 2nd-to-last element, … and the last element becomes the first element. You may assume the IntQueue class has the standard methods `isEmpty`, `enqueue` and `dequeue`.

```
public static void reverse(IntQueue Q) {

      int temp;
```

```
    if ____________________________________   // stopping case
        return;
    else {   // recursive case: must be exactly 3 statements
        ____________________________________;
        ____________________________________;
        ____________________________________;
    }
}
```

Answer:

```
public static void reverse(IntQueue Q) {
    int temp;
    if ______(Q.isEmpty())___   // stopping case
        return;
    else {     // recursive case: must be exactly 3
statements
        ____temp = Q.dequeue()__;
        ____reverse(Q)__________;
        ____Q.enqueue(temp)_____;
    }
}
```

(7) Consider the following recursive method on an integer array data with n integers:

```
public static int mystery(int[] data, int n) {
    int sum;
    if (n <= 0) return 0;
    else {
        if (data[n-1] % 2 == 0)
            sum = mystery(data, n-1) + 1;
        else
            sum = mystery(data, n-1);
        return sum;
    }
}
```

a. What is the final return value of this method if it is called with the following parameters initially: data = {2, 2, 3, 3, 3, 4, 4, 4, 4}, n = 9?

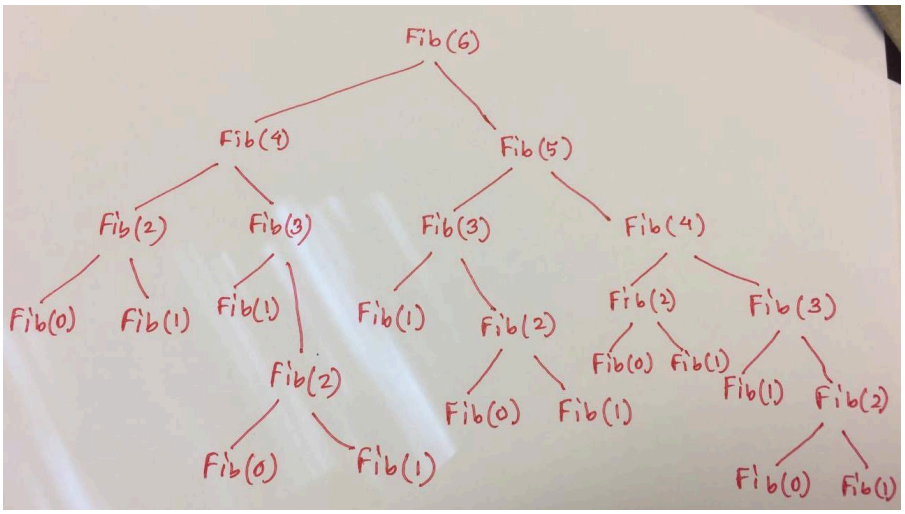
Answer: 6

(8) Draw a graph illustrating the recursive call tree structure for computing Fib(6) (the 6th fibonacci number) for:

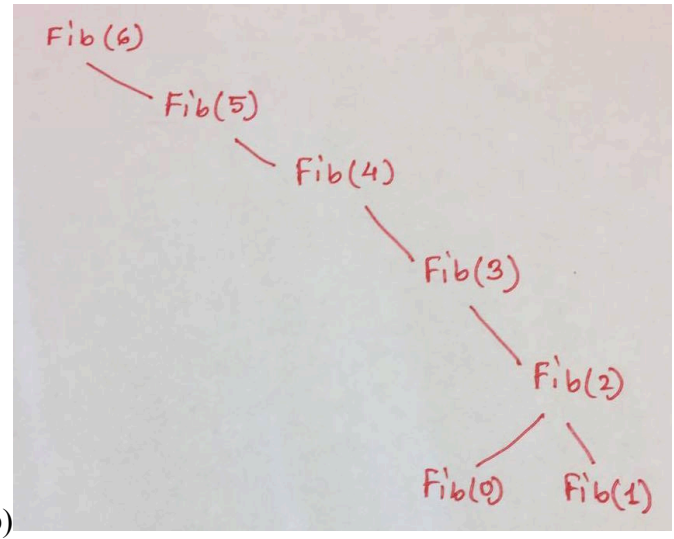
a. A purely recursive approach
b. Recursion using dynamic programming with top-down memorization
c. What is the difference between top-down and bottom-up dynamic programming? Write a code for Fib(int n) with bottom-up dynamic programming.

Answer:

(a)



(b)



(c) Top Down dynamic programming starts by looking at the given argument and computing smaller results as needed. Bottom up dynamic programming starts by evaluating the base cases and then building up all larger values until required value is computed.

```
public int fibonacci(int n) {
```

```
    int[] d = new int[n+1];

    d[0] = 0;

    d[1] = 1;

    for(int i = 2; i <= n; i++)

         d[i] = d[i-1] + d[i-2];

    return d[n];

}
```

(9) Use the algorithm discussed in class to convert the following infix expression into postfix notation by showing the state of the stack at each step in the conversion.
(G * (((A * (B / (C + D))) – E) + F))

Answer:

Stack (sideways)	Postfix string
empty	G
*	G
*	GA
* *	GA
* *	GAB
* * /	GAB
* * /	GABC
* * / +	GABC
* * / +	GABCD
* * /	GABCD+
* *	GABCD+/
*	GABCD+/*
* -	GABCD+/*
* -	GABCD+/*E
*	GABCD+/*E-
* +	GABCD+/*E-
* +	GABCD+/*E-F
*	GABCD+/*E-F+
empty	GABCD+/*E-F+*

(10) You are given a string having parenthesis like below
"(((X)) (((Y))))"
You have to find the maximum depth of balanced parenthesis, like 4 for the above

example. Since 'Y' is surrounded by 4 balanced parentheses. If parentheses are unbalanced then return -1.

a. Which data structure is most suitable to solve this problem linked list or stack?
b. Write code (or pseudocode) to solve this problem based on your answer to the previous question. You do not need to write the linked list or stack class. Consider those are given to you.

Answer:

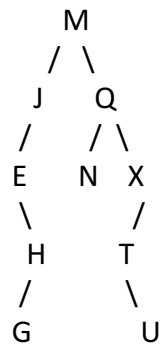
(a) Stack

(b)
```
public int maxDepth(String str){
        charStack S = new charStack();
        maxdepth = 0;
        for(int i = 0; i  < str.length(); i++){
                if(str.charAt(i) == '('){
                        S.push(str.charAt(i));
                        maxdepth = Math.max(maxdepth, S.size());
                }
                else if(str.charAt(i) == ')')
                        S.pop();
        }
        if(S.isEmpty())
                return maxdepth;

        else
                return -1;

}
```

(11) USE THE FOLLOWING BINARY SEARCH TREE FOR QUESTIONS I AND II:



(I)

(a) Write the inorder traversal of the binary tree above.
(b) Write the preorder traversal of the binary tree above.
(c) Write the postorder traversal of the binary tree above.

(d) For any binary search tree with *n* letter nodes, what is the maximum number of nodes that have to be searched to find any letter?

(II)

(a) What is the depth of the binary tree above?

(b) What is the maximum number of nodes that can be inserted to the binary tree above without increasing its depth and without changing the binary tree property?

(c) Draw the binary search tree after the node containing M is removed, consider a letter e.g., 'A' is smaller than another letter 'B' if it appears before 'B' in the alphabet.

Answer:

(I)

(a) EGHJMNQTUX

(b) MJEHGQNXTU

(c) GHEJNUTXQM

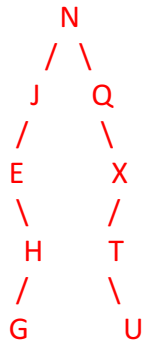
(d) n

(II)

(a) 4

(b) 14 + 6 + 1 = 21

(c)



(12) Let the class BTNode represent a node of a binary tree that stores an integer, defined as follows:

```
public class BTNode {

    private int data;

    private BTNode left, right;

    // BTNode methods

}
```

Write Java code for the following recursive BTNode methods,

(a) `public void inorder()`
Prints the contents of the binary tree rooted at this node using an inorder traversal.

(b) `public int count()`
Returns the number of leaves in the binary tree rooted at this node.

Answer:

(a)
```
{
        if (left != null) left.inorder();
        System.out.println(data);
        if (right != null) right.inorder();
}
```

(b)
```
{
  if ((left == null) && (right == null)) return 1;
  int sum = 0;
  if (left != null) sum = sum + left.count();
  if (right != null) sum = sum + right.count();
  return sum;
}
```