Data Structures and Algorithms (CSE-102)
Mid-Sem Exam

Time: 90 min. **Date: 16th February, 2020**
Marks: 50

Instructions: A) Do not spend too much time on one question. B) Do not repeat the question while writing your answer. C) Make appropriate assumptions, only if necessary, that do not oversimplify the problem. Clearly state your assumptions. D) Write clear and concise answers.

Q1. **[20 marks] Topic- Recursion**

a. **[5 marks]** Predict the output and explain the functionality of the following function:

```
int fun(int count) {
   printf("%d\n", count);
   if(count < 3) {
     fun(fun(fun(++count)));
   }
   return count;
}
int main()
{
   fun(1);
   return 0;
}
```

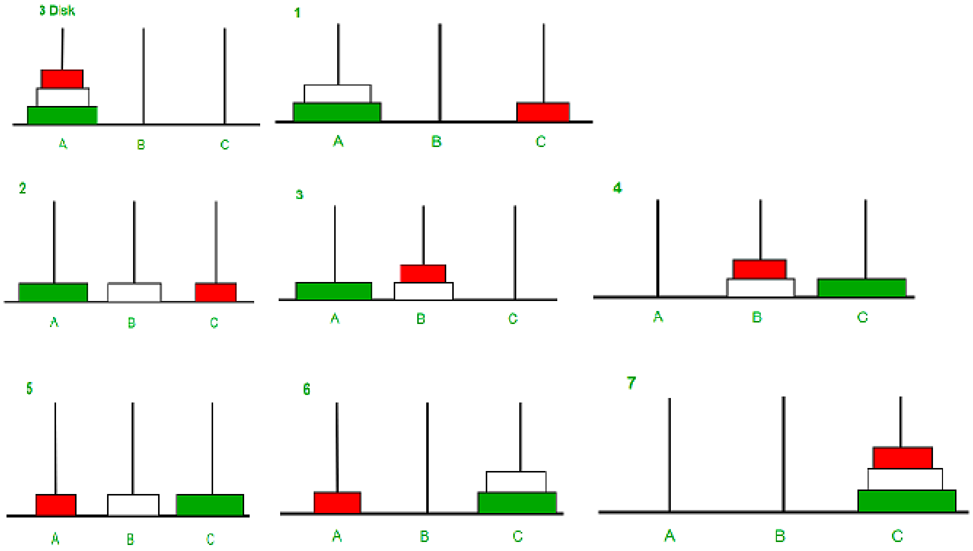
//2 marks for correct output, 3 marks for for steps (internal binary marking)

```
Output: 1
  2
  3
  3
  3
  3
  3
```

The main() function calls fun(1). fun(1) prints “1” and calls fun(fun(fun(2))). fun(2) prints “2” and calls fun(fun(fun(3))). So the function call sequence becomes fun(fun(fun(fun(fun(3))))). fun(3) prints “3” and returns 3 (note that count is not incremented and no more functions are called as the if condition is not true for count 3). So the function call sequence reduces to fun(fun(fun(fun(3)))). fun(3) again prints “3” and returns 3. So the function call again reduces to fun(fun(fun(3))) which again prints “3” and reduces to fun(fun(3)). This continues and we get “3” printed 5 times on the screen.

b. **[15 marks]** Let us have some fun with the **Tower of Hanoi**. Tower of Hanoi is a mathematical puzzle where we have **three** pegs (rods) and **n** disks. The objective of the puzzle is to move the entire stack of disks to another peg, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

For example, let the three pegs be A, B and C and the no. of disks be n. In order to shift n disks from A to C, following steps will be followed:
Step1- Shift 'n-1' disks from 'A' to 'B' using C.
Step2- Shift last disk from 'A' to 'C'.
Step3- Shift 'n-1' disks from 'B' to 'C' using A.

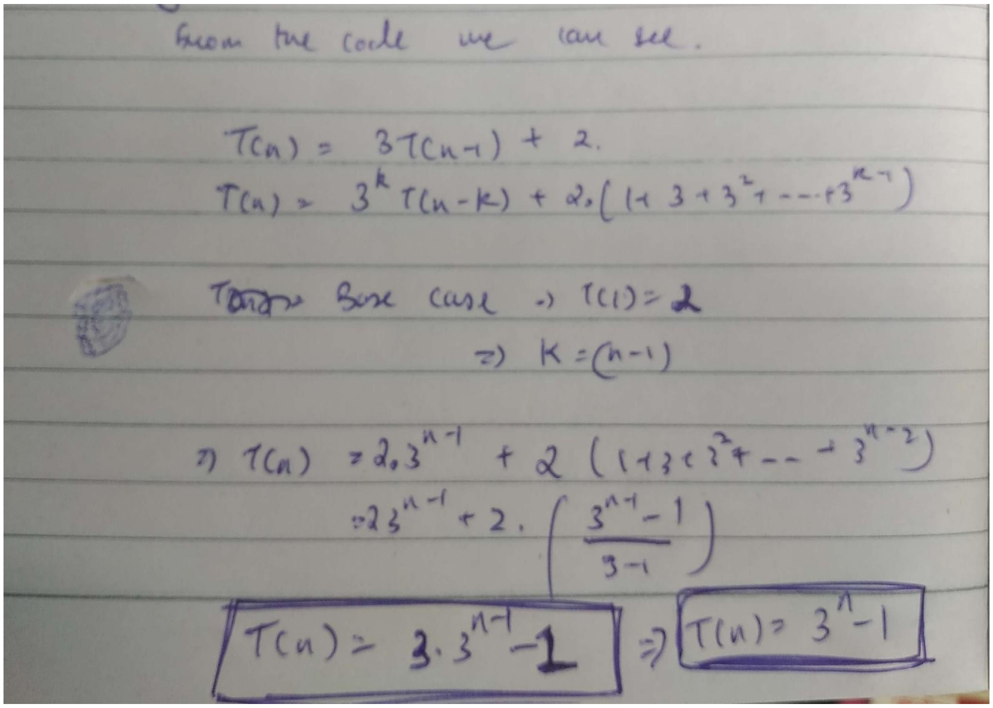


Now, suppose that the pegs are numbered **0, 1, 2** where **1** is the source, **2** is destination and **0** is the interim peg.
Write the **java type pseudo code** for solving the **Tower of Hanoi** problem with a restriction that all the moves taken to solve the problem **must involve peg 0** i.e no move is possible from peg 1 to peg 2.
Also write the recurrence relation for the algorithm you have used and derive its complexity.
// 10 marks for java pseudo code, (partial marking at TA discretion)

//2 for correct recurrence, 2 for deriving the complexity, 1 for correct complexity (internal binary marking)

// no move is possible from peg 1 to peg 2, but move from peg 2 to peg 1 is allowed also it can include peg 0. Complexity may vary accordingly.

```
public modifiedTowerOfHanoi(int n, int from, int aux, int to)
{
        if(n==1)
        {
                System.out.println(“Moving disc 1” + “from ” + str(from) + “to” +
str(aux));
                System.out.println(“Moving disc 1” + “from ” + str(aux) + “to” +
str(to));
        }
        modifiedTowerOfHanoi(n-1, from, aux, to);
        System.out.println(“Moving disc ” +str(n)+ “from ” + str(from) + “to ” +
str(aux) );
        modifiedTowerOfHanoi(n-1, to, aux, from);
        System.out.println(“Moving disc ” +str(n)+ “from ” + str(aux) + “to ” +
str(to) );
        modifiedTowerOfHanoi(n-1, from, aux, to);
}
public static void main()
{
        modifiedTowerOfHanoi(3, 1, 2, 0);
}
```



from the code we can see.

$$T(n) = 3T(n-1) + 2.$$

$$T(n) = 3^k T(n-k) + 2.(1+3+3^2+---+3^{k-1})$$

Base case $\rightarrow T(1) = 2$

$$\Rightarrow K = (n-1)$$

$$\Rightarrow T(n) = 2.3^{n-1} + 2(1+3+3^2+---+3^{n-2})$$

$$= 2 3^{n-1} + 2.\left(\frac{3^{n-1}-1}{3-1}\right)$$

$$\boxed{T(n) = 3.3^{n-1} - 1} \Rightarrow \boxed{T(n) = 3^n - 1}$$

The steps required to solve this problem are:

1. Recursively Move top n-1 disks from peg 1 to peg 2, with every move involving peg 0
2. Move disk n from peg 1 to peg 0
3. Recursively Move n-1 disks from peg 2 to peg 1
4. Move disk n from **peg 0 to peg 2**
5. Recursively Move n-1 disks from peg 1 to peg 2, with every move involving peg 0

Q2. **[10 marks] Topic- Sorting/ Complexity Analysis**

a. **[2 marks]** Show the asymptotic runtime analysis of given function and find its complexity:

```
for(i=1; i<=n; i++){
 for(i=1; i<=n^2; i++){
  for(i=1; i<=n^3; i=5*i)
 {x = y + z}
}}
```

//1 mark for explanation/ analysis, 1 for correct complexity

O(logn)

b. **[3 marks]** A machine needs a minimum of 200 sec to sort 1000 elements by Quick sort. Considering the worst-case, find the minimum approximate time needed to sort 200 elements.

Explanation:

The Quick sort requires n^2 comparisons in worst case, where n is size of input array. So, 1000 * 1000 ≈ 10^6 comparisons are required to sort 1000 elements, which takes 200 sec. **//1 mark**

To sort 200 elements minimum of 200 * 200 ≈ 40000 comparisons are required. **//1 mark**

This will take 200 * 40000 / 1000000 ≈ 8 sec. **//1 mark**

c. **[3 marks]** In a modified merge sort, the input array is splitted at a position one-third of the length(N) of the array. Find the recurrence relation for this modified Merge Sort. What would be the tightest upper bound on time complexity of this algorithm? **//1 for recurrence, 1 for analysis/ explanation, 1 for complexity**

The time complexity is given by: T(N) = T(N/3) + T(2N/3) + N
Solving the above recurrence relation gives, T(N) = N(logN base 3/2)

d. **[2 marks]** What is the best possible way for choosing the pivot element? What's the worst case of Quick Sort in that case? Justify your answer.

//1 mark for correct best possible way, 0.5 for Quicksort worst case, 0.5 for justification.
Choosing pivot randomly from the elements of the sequence is the safest possible way of choosing the pivot element.
Moreover, The quadratic worst case for (almost) sorted sequences can also be eliminated by taking the median of the first, last, and middle elements as the pivot. This is because, the median of the 1st, middle and last element of the sorted array is the middle element for the entire array which ensures a partition with exactly ½ of the array elements.
Thus making the time complexity = 2*T(n/2) + O(n) = … = 2^k T(n/2^k) + kO(n) = n+nlog(n) = nlog(n).
However, the worst case time complexity is still O(n^2) since there is still a chance that the pivot is the smallest element in the sequence.

Q3. **[10 marks] Topic- Stack/ Queue**

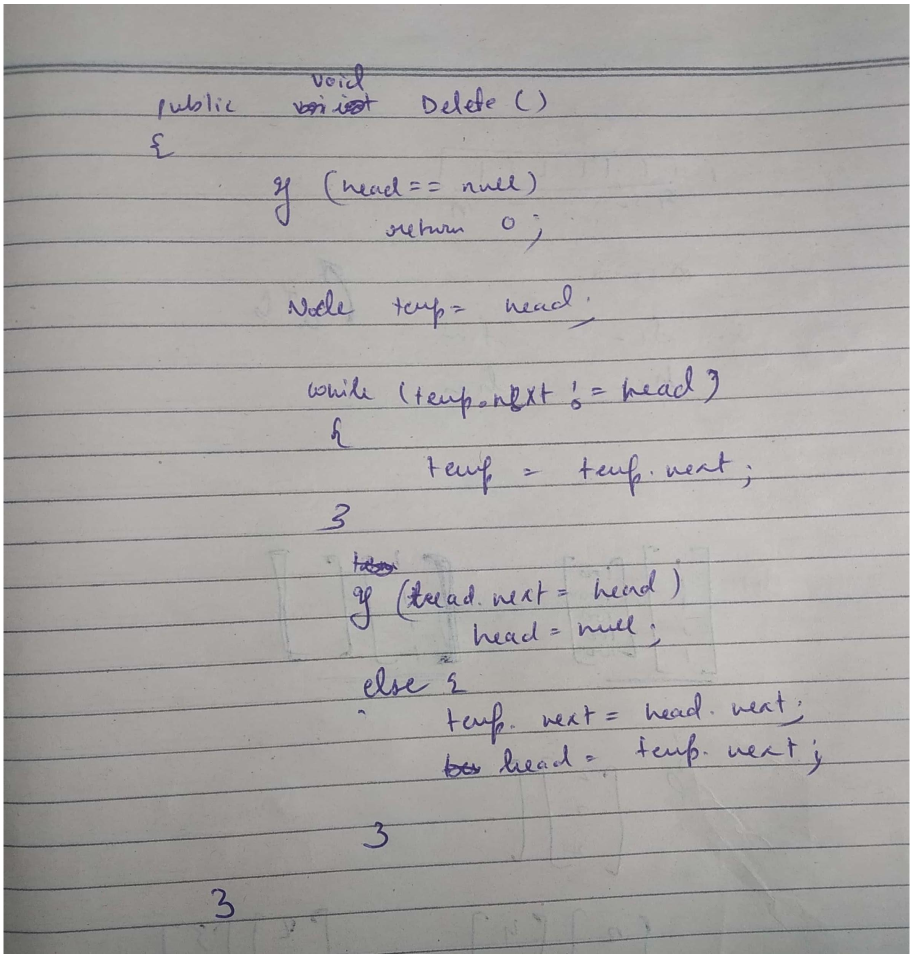
a. **[5 marks]** Write the implementation of ~~circular~~ queue using **circular** linked list.

// 5 marks for java pseudo code, (partial marking at TA discretion)

Qn.3

```
class Node
{
    int data;
    Node next;
    public Node( int data )
    {
        this.data = data;
        ~~this.data = data~~;
    }
}
class Queue
{
    Node head = ~~new Node()~~ null;

    public void insert ( int data)
    {
            if( head == null)
            {
                 head= new Node(data);
            }
            else
            {
                 Node newN = new Node (data);
                 Note temp = head;
                 while( temp.next != head)
                 {
                       temp = temp.next;
                 }
                 newN.next= head;
                 temp. next= newN;
    }
```



```
public   void
         ~~voi int~~  Delete ()
{
        if (head == null)
              return 0;

        Node temp = head;

        while (temp.next != head)
        {
              temp = temp.next;
        }
          ~~temp~~
          if (~~h~~head.next = head)
                 head = null;
          else {
                 temp. next = head. next;
              ~~be~~ head = temp. next;

          }

}
```

b. **[5marks]** Here is an INCORRECT pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while ( more input is available)
 {
          read a character
          if ( the character is a '(' )
                 push it on the stack
          else if ( the character is a ')' and the stack is not empty )
                 pop a character off the stack
          else
                 print "unbalanced" and exit
 }
print "balanced"
```

Give two examples of unbalanced sequences which the above code thinks is balanced. **2.5 marks for each example, internal binary marking**

Ans. ((()) //any example in which one parenthesis is missing

Q4. [10 marks] Topic- LinkedList

a. **[5 marks]** Write the recursive solution **(Java type pseudo code)** for reversing doubly linked list in-place. (without using extra space).

// 1 mark for base condition, 4 marks at TA discretion

Ans. https://www.geeksforgeeks.org/reverse-doubly-linked-list-using-recursion/

Algorithm

1) If list is empty, return

2) Reverse head by swapping head->prev and head->next

3) If prev = NULL it means that list is fully reversed. Else reverse(head->prev)

b. **[2 marks]** Suppose there are two single linked lists both of which intersect at some point and become a single linked list. The head or start pointers of both the lists are known, but the intersecting node and lengths of lists are not known. What is the worst case time complexity of optimal algorithm to find intersecting node from two intersecting linked lists, where m, n are lengths of given lists?

//1 mark for complexity and 1 for explanation

This takes Θ(m+n) time in worst case, where m and n are the total lengths of the linked lists.

1. Traverse the two linked list to find m and n.
2. Get back to the heads, then traverse |m − n| nodes on the longer list.
3. Now walk in lock step and compare the nodes until you found the common ones.

c. **[3 marks]** Given only a pointer/reference to a node to be deleted in a singly linked list, how do you delete it, provided that we don't have pointer to head node? Write the algorithm in steps.

//1 mark for approach and 2 for algorithms (internal binary)

Copy the data from the next node to the node to be deleted and delete the next node. Something like following:

```
struct Node *temp  = node_ptr->next;

 node_ptr->data  = temp->data;

 node_ptr->next  = temp->next;

 free(temp);
```