National University of Computer and Emerging Sciences

Assignment No. 3

Design and Analysis of Algorithms

Time Complexity

CS2009

Fall 2024

Deadline: September 18, 2024

Submission Instructions:

- All problems must be solved and submitted on Google Classroom.
- Use A4 size papers.
- Submit the scanned Assignment in PDF form on Google Classroom.
- This is an individual assignment.
- **Plagiarism is strictly prohibited.**
- **Please do not use any AI tool and do not copy from others.**
- **Just analyze the problem and then brainstorm the solution.**

Question 01 (Master Theorem): (80 Marks)

Apply Master Theorem on below equations.

1. T (n) = 4T (n/2) + n ^2
2. T (n) = 2nT (n/2) + n ^n
3. T (n) = 2T (n/2) + n log n
4. T (n) = 3T (n/3) + √ n
5. T (n) = √ 2T (n/2) + log n
6. T (n) = 5T (n/3) + n ^3
7. T (n) = 7T (n/4) + n log n
8. T (n) = 8T (n/4) + n √n

Question 02 (Recurrence Relation): (70 Marks)

1. This procedure is a solution to the **classic Towers of Hanoi problem.** Find the recurrence relation and Time complexity using Tree.

```
void move(int from, int to, int aux, int numDisks)
{
    // pre: numDisks on peg from,
    // post: numDisks moved to peg to

    if (numDisks == 1)
    {
        System.out.println(from + " to " + to);
    }
    else
    {
        move(from, aux, to, numDisks - 1);
        move(from, to, aux, 1);
        move(aux, to, from, numDisks - 1);
    }
}
```

2. This procedure is the solution of check BST or not Function. Find the recurrence relation and

Time complexity using Tree.

```
boolean isBST(Tree t)
{
    // postcondition: returns true if t represents a binary search
    // tree containing no duplicate values;
    // otherwise, returns false.



    if (t == null) return true;  // empty tree is a search tree

    return valsLess(t.left, t.info) &&
           valsGreater(t.right, t.info) &&
           isBST(t.left) &&
           isBST(t.right);
}
```

3. Consider the following recurrence relation: **T(n) = T(n/3) + O(n), T(1) = 1**. What is the time complexity of this algorithm?

4. Consider the following recurrence relation: **T(n) = 2T(n/2) + O(n log n), T(1) = 1**. What is the time complexity of this algorithm?

5. Consider the following recurrence relation: **T(n) = T(n/2) + T(n/4) + O(n), T(1)= 1**. What is the time complexity of this algorithm?

6. Consider the following recurrence relation: **T(n) = T(n/3)+T(2n/3)+O(n), T(1)= 1**. What is the time complexity of this algorithm?

7. Consider the following recurrence relation: **T(n) = 4T(n/3)+O(log n), T(1)= 1**. What is the time complexity of this algorithm?