DSA-Lab # 5 - Recursion

1. **Write a recursive function for computing the following Series. Drive their time complexity.**
   I. **1+2+3+4+...+N**
   II. **1+3+5+7+....+N** (where N must be odd)
   III. **$1+2+4+8+16+...+2^N$** (N is an integer) **N can be upto 63. Specifically check for N>32.**
   IV. **$1+3+9+27+81+...+3^N$ N can be upto 55. Specifically check for N>32.**
   V. **1+3+9+27+81+... + N/9 + N/3+N**
   VI. **1+2+4+8+16+...+ N/2+N**

2. **Write the recursive code for.**
   I. **Decimal to Binary Convertor (just display it)**
   II. **Itoa convertor (integer to string convertor)**
   III. **Write the recursive code for GCD(A, B)**

3. **Write the Recursive**
   I. **Write the recursive code for SearchFirstEntry**
   II. **Write the recursive code for SearchLastEntry**
   III. **BinarySearch and test it on a huge Data.**

4. **Write a Recursive function to compute POWER(X, Y, M) compute X^Y % M (X^Y modulo M).**
   I. **The algorithm must take O(Y)**
   II. **The algorithm must take O(log Y) times additions/subtractions.**

5. Write a Program which wants to do multiplication of AxB imagine all A and B are n bit strings and there is module available **ADD(X,Y)** and you want to write this **MULT(X,Y)** using **ADD(X,Y)** How you will going to write this module. Write the module such that It takes a minimum number of steps, obviously Calling ADD Y times is a very bad idea and unacceptable.
   I. **The algorithm must take O(Y) times additions/subtractions.**
   II. **The algorithm must take $O(\log^2 Y)$ times additions/subtractions.**
   III. **(BONUS) Using Memoization/Bottom Up approach - Write The algorithm must take O(log Y) times additions/subtractions.**

6. **Write a Program which you compute A/B and A%B and the only operations allowed are subtraction and addition.**
   I. **The algorithm must take O(B) times additions/subtractions.**
   II. **The algorithm must take $O(\log^2 B)$ times additions/subtractions.**
   III. **Using Memoization/Bottom Up approach - Write The algorithm must take O(log B) times additions/subtractions.**

7. **Write the recursive and iterative code for Fibonacci Number computation.**
   I. Analyze why Iteration is working so fast as compared to recursive implementation of Fibonacci Numbers.
   II. Use **Memorization Technique** to make the recursive algorithm fast.
      - **Test on which depth it fails?**
   III. **Do BottomUp approach of iterative version of Fibonacci Numbers.**

8. **Write the recursive and iterative code for Computing the TriSum sequence:** 1, 2, 3, 6, 11, 20, 37, ...
   I. Write the recursive mathematical formulation.
   II. Write the recursive code for the Sequence generator
   III. Analyze what will be its time complexity (the approximate number of times the recursive call will be called.
   IV. Give the **Memorization Technique** solution to avoid recalculation of the same TriSum number again and again.
      - **Test on which depth it fails?**
   V. **Do BottomUp approach an iterative version of TriSum.**

CHALLENGE 1

Given an array, generate all the possible subarrays of the given array using recursion.

Examples: **Input :** [1, 2, 3] **Output :** [], [1], [1, 2], [2], [1, 2, 3], [2, 3], [3] **Input :** [1, 2] **Output :** [], [1], [1, 2], [2]

CHALLENGE 2

Given a stack, sort it using recursion. Use of any loop constructs like while, for..etc is not allowed. We can only use the following ADT functions on Stack S:

is_empty(S)	: Tests whether stack is empty or not.
push(S)	: Adds a new element to the stack.
pop(S)	: Removes top element from the stack.
top(S)	: Returns value of the top element. Note that this function does not remove elements from the stack.

Example:

Input:
-3 <--- Top 14 18 -5 30

Output:
30 <--- Top 18 14 -3 -5

Note: you can add utility functions(without using loop).