Recursion - I

Sum till n

Objective: To get the sum of the first n number using recursion.

Base Case : If n == 0:

return 0;

We add the current number and recurse for Sum(n-1)

Time Complexity: O(N)

Space Complexity: O(N)

```
int Sum(int n) {
    if (n == 0) {
        return 0;
    }
    int prevSum = Sum(n - 1);
    return n + prevSum;
}
```

Print the numbers in the sequence:

Objectives:

1. Print the number in increasing order up to n.
2. Print the number in decreasing order from n to 1.

Approach:

For increasing order, we first print the remaining numbers using recursion and then print the current number.

For decreasing order, we first print the current number and then print the remaining number using recursion.

Time Complexity: O(N)

Space Complexity: O(N)

```
void dec(int n) {
    if (n == 1) {
        cout << "1" << endl;
        return;
    }
    cout << n << endl;
    dec(n - 1);
}

void inc(int n) {
    if (n == 1) {
        cout << "1" << endl;
        return;
    }
    inc(n - 1);
    cout << n << endl;
}
```

Fibonacci

Objective: To get the Fibonacci of n

Recurrence Relation: F(n) = F(n-1) + F(n-2)

Base Case : If (n == 0 or n == 1): //Since F(n) depends on prev two states

return n; //so base case will have two cases

We calculate fib(n - 1), fib(n - 2) using recursion, and return the sum of the two

Time Complexity: O(N)

Space Complexity: O(N)

```
int fib(int n) {

    if (n == 0 || n == 1) { //base case
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}
```

Factorial

Objective: To get the factorial of the n

Base Case : If n == 0:

return 1;

We multiply the current number and recurse for factorial(n-1)

Time Complexity: O(N)

Space Complexity: O(N)

```
int factorial(int n) {

    if (n == 0) {
        return 1;
    }
    //int prevfact= factorial(n-1);
    return n * factorial(n - 1);

}
```

To calculate n^p using Recursion

Objective: To calculate pth power of n i.e. $\mathbf{n^p}$.

Base Case : If p == 0:

return 1;

We multiply the current number and recurse for power(n, p - 1)

Time Complexity: O(N)

Space Complexity: O(N)

```
int power(int n, int p) {

    if (p == 0) {
        return 1;
    }

    int prevPower = power(n, p - 1);
    return n * prevPower;
}
```