Today's agenda

- Recursion
- How to write Recursive code.

why recursion →

- Tree
- Backtracking
- DP → Amazon
- Graph

Google

Recursion:

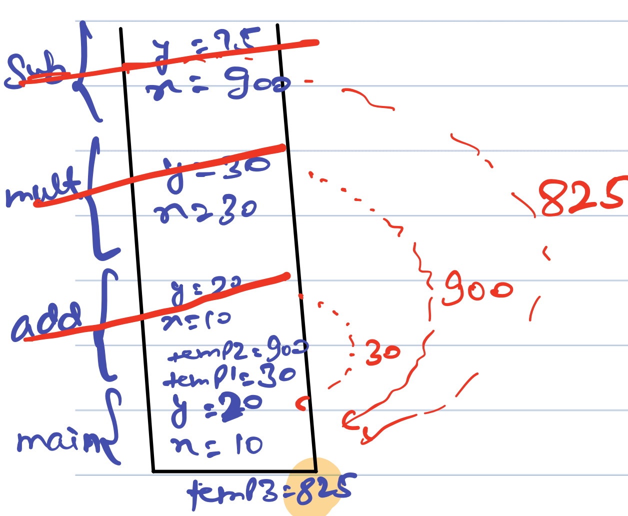
↳ function calling itself.

* function call

```
main ( ) {
    int x = 10;
    int y = 20;
    int temp1 = add(x, y);
    int temp2 = mult(temp1, 30);
    int temp3 = sub(temp2, 75);

        s.o.p(temp3);

}
```

10 20 above x, y; 30 above temp1; 900 above temp2

```
int add(int x, int y){
    return x+y;
}

int mult(int x, int y){
    return x*y;
}

int sub(int x, int y){
    return x-y;
}
```



// Thought Process

$N = 5$

$$Sum(N) = 1 + 2 + 3 + \dots + N-1 + N$$

$$\overset{15}{Sum(5)} = \overset{10}{Sum(4)} + 5$$

$$\overset{10}{Sum(4)} = \overset{6}{Sum(3)} + 4$$

$$\overset{6}{Sum(3)} = \overset{3}{Sum(2)} + 3$$

$$\overset{3}{Sum(2)} = \overset{1}{Sum(1)} + 2$$

$$\overset{1}{Sum(1)} = \overset{0}{Sum(0)} + 1$$

Q) Given N, find sum of no-s from (1.....N), using recursion

Three magical steps of recursion.

Faith: define what your function should do and have faith that it works.

Main logic: Solve your Problem with SubProblem
↳ smaller instance of same Problem.

Base case: Solution to Smallest SubProblem.


```
int Sum (int N){
    if(N==1){return 1;}

    int temp= Sum(N-1);

    return temp+N;
}
```

Faith: Given N, Calculate & return sum of first N natural no-s.

Main logic:

Sum (N) ← temp+N

↓

Sum (N-1) → temp

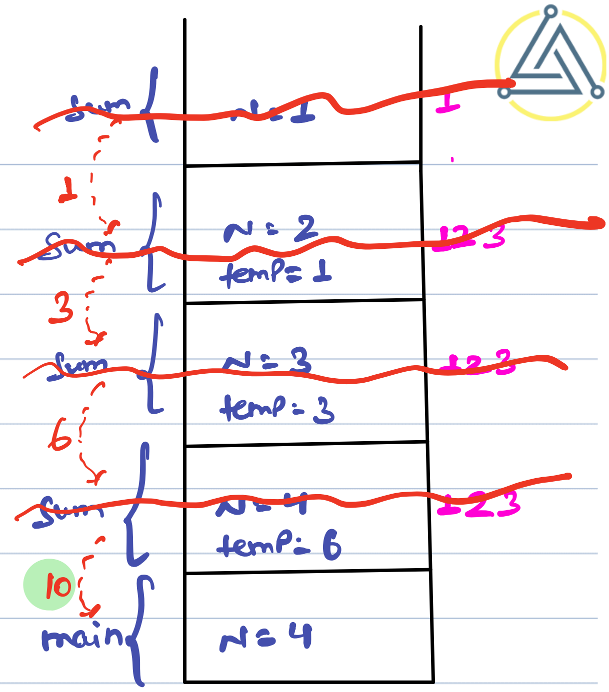
base case: N==1 → 1

↳ overall T.C: O(1) * N = O(N)

↳ overall S.C: O(1) * N = O(N)

Tracing → dry run

```
int  Sum ( int N) {
1  if (N == 1) { return 1; }

2  int temp = Sum (N-1);

3  return  temp + N;
}
```



Sum (N) ✓✓ ↑ ans

↓

Sum (N-1) ✓✓

↓

Sum (N-2) ✓✓

↓

Sum (N-3) ✓✓

⋮ ✓✓

⋮ ✓✓

Sum (1) ✓✓

Sum (4) 6+4 = 10

↓ ↑ 6

Sum (3) 3+3

↓ ↑ 3

Sum (2) 1+2

↓ ↑ 1

Sum (1)

Q) find factorial of N.

Ex: N=3 → 3*2*1 = 6

N=4 → 4*3*2*1 = 24

```
int fact(int N) {
    if(N==0){ return 1; }

    int temp = fact(N-1);

    return temp*N;

}
```

Faith: Given N, Calculate & return factorial of N.

main logic:

fact(N) ← temp*N

↓

fact(N-1) → temp

overall T.C: O(1) * N = O(N)

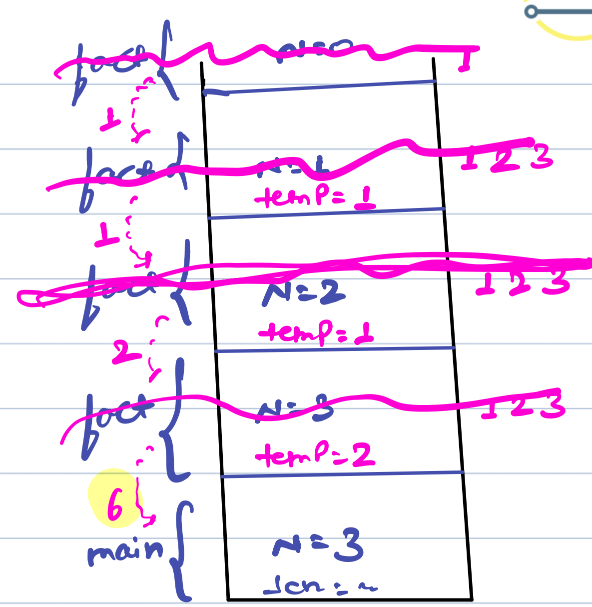
overall S.C: O(1) * N = O(N)

base case:

fact(0) = 1

Dry run

```
int fact(int N) {
1  if (N==0) { return 1; }

2  int temp = fact(N-1);

3  return temp * N;
}
```



fact(3)

↓

fact(2)

↓

fact(1)

↓

fact(0)

Break till 9:40 pm

Q) Print N^{th} fibonacci number, with recursion.

Ex:

0	1	2	3	4	5	6	7	8	9	10
0	1	1	2	3	5	8	13	21	34	55

$fib(N) = fib(N-1) + fib(N-2)$.

```
int fib(int N) {
    if(N==0){ return 0; }
    if(N==1) { return 1; }

    int tempP1 = fib(N-1);
    int tempP2 = fib(N-2);

    return tempP1 + tempP2;
}
```

Faith: Given N, calculate & return Nth fibonacci number.

Main logic:

fib(N) → tempP1 + tempP2

fib(N-1) → tempP1 fib(N-2) → tempP2

Base Case:

↳ fib(0) = 0

fib(1) = 1

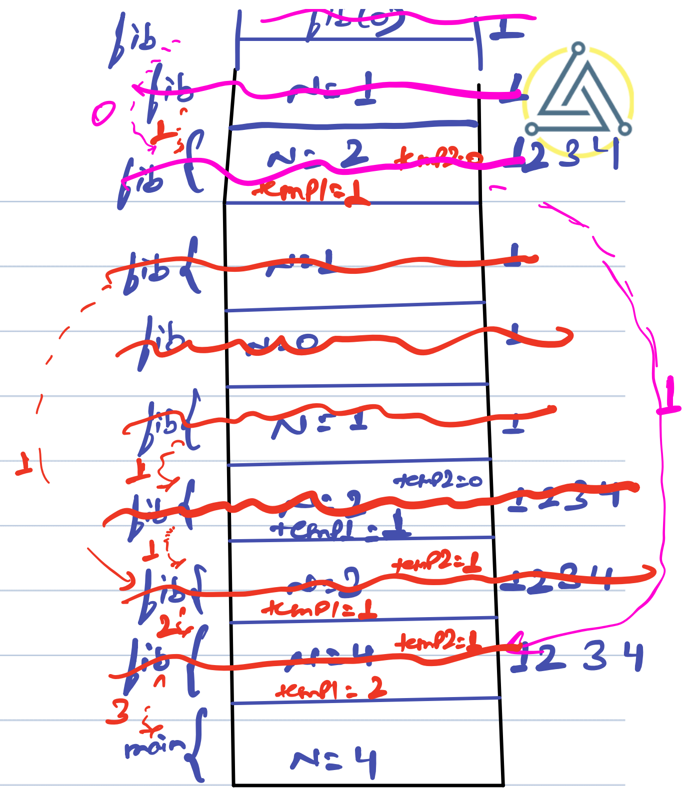
T.C of 1 function: O(1)

No. of functions = 2^N

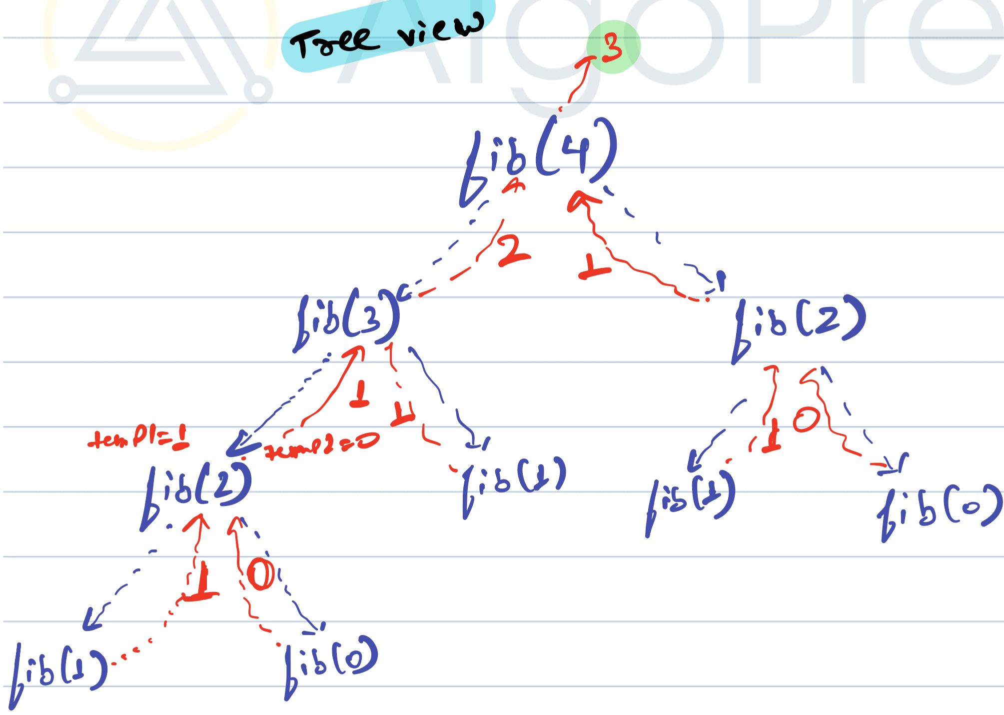
overall S.C: O(1) * N

= O(N)

```
1 { if (N==0){ return 0; }
    if (N==1) { return 1; }
int fib(int N) {

2   int temp1 = fib(N-1);
3   int temp2 = fib(N-2);

4   return temp1 + temp2;
}
```



Tree view



Q) Print increasing

↳ Given N, Print all the numbers from 1→N, using recursion.

```
void Printincreasing (int N){
    if (N==1){ s.o.p(1);
               return; }

    Printincreasing (N-1);
    s.o.p (N);
    return;
}
```

Faith: Given N, Print no.s from 1 to N.

Main logic:

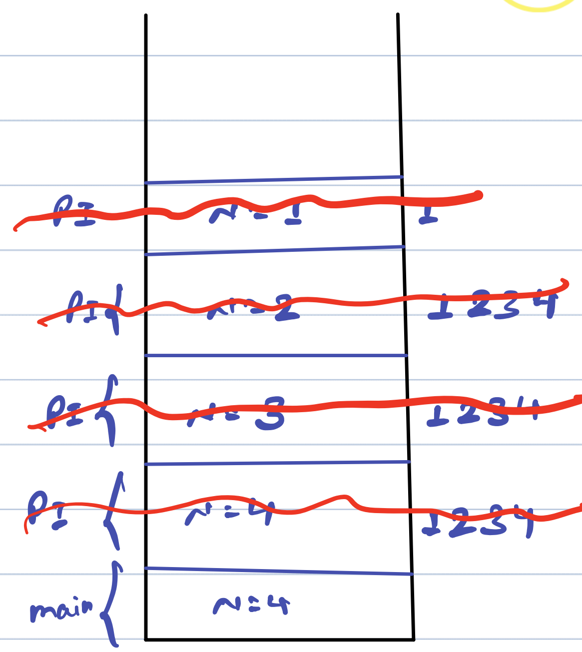
{{1 2 3 . . N-1} N}

Base Case:

if (N==1){ s.o.p(1); return; }

Tracing

```
void PrintIncreasing(int N){
  1  if(N==1){ s.o.p(1);
                return;}

  2  PrintIncreasing(N-1);
  3    s.o.p(N);
  4    return;
}
```



1
2
3
4

PI(4) 4 ✓

↓

PI(3) 3 ✓

↓

PI(2) 2 ✓

↓

PI(1) ✓

```
int fib(int n) {
1 { if (n==0){ return 0; }
    if (n==1) { return 1; }

2 int temp1 = fib(n-1);
3 int temp2 = fib(n-2);

4 return temp1 + temp2;
}
```