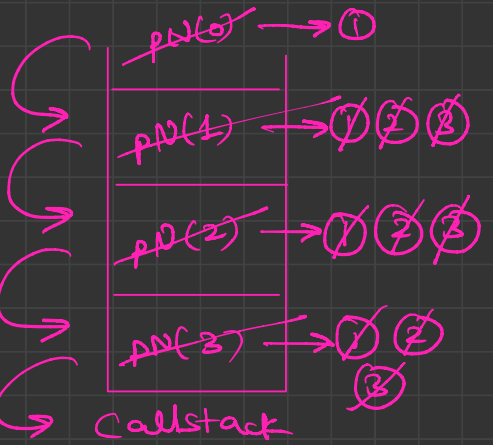
Recursion

faith: It prints "AccioJobs" N times;

```
void printNTimes(int N)
{
    (1) if (N == 0) return;          ---> Base Case
    (2) System.out.println("AccioJobs");   ---> My Work
    (3) printNTimes(N-1);            ---> recursive call
}
```



o/p

```
AccioJobs
AccioJobs
AccioJobs
```

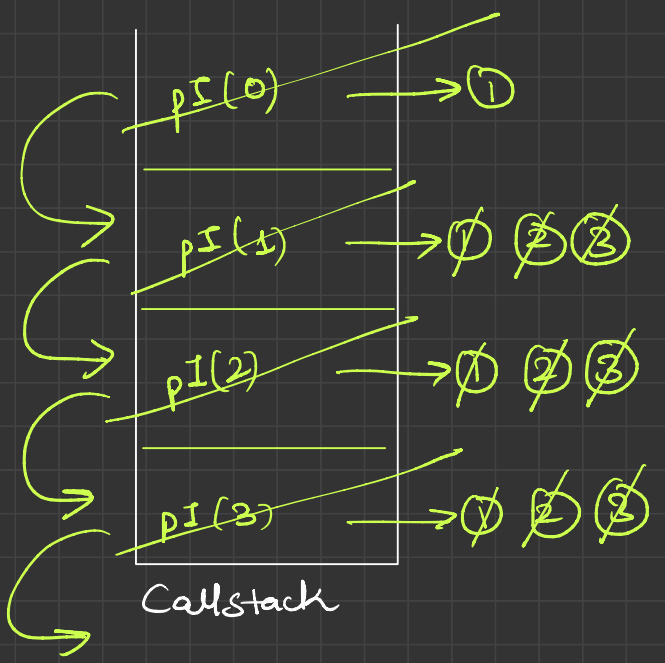
Q print numbers in inc. fashion from 1 ..... N.

faith: print Numbers from 1 ..... N.

```
void printInc(int N)
{
① if(N==0) return;

② printInc(N-1);
③ System.out.print(N+" ");
}
```

o/p

1 2 3



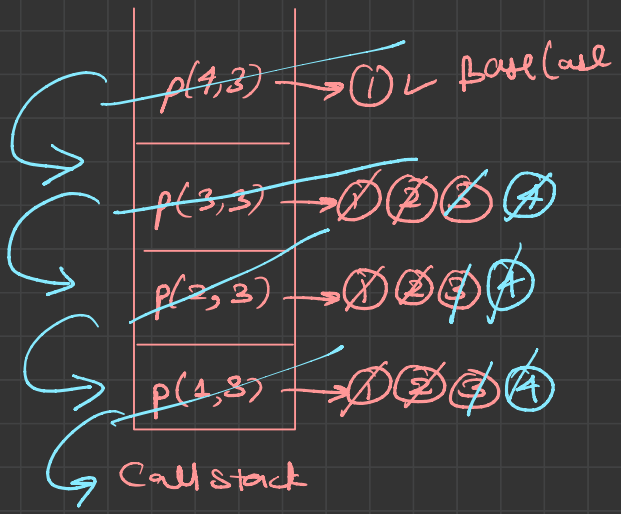
Q. print Numbers as

1 2 3 4 5 5 4 3 2 1 1 2 3 3 2 1

faith: It print from i → N , then N → i

```
void printInDec (int i, int N)
{
  (1) if (i > N) return;
  (2) System.out.print(i + " ");
  (3) printInDec (i+1, N);
  (4) System.out.print(i + " ");
}
```

O/P 1 2 3 3 2 1



Q. fibonacci.

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	.....
0	1	1	2	3	5	8	13	21	.....

$$fibo(n) = fibo(n-1) + fibo(n-2) \quad n \neq 1, 2$$

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	. . . . .
0	1	1	2	3	5	8	13	21	. . . . .

→ faith: returns, n^{th} fibonacci number

```
int fibo(int N)
{
    if(N==1) return 0;
    if(N==2) return 1;          } Base Case

    int prev = fibo(N-1);       ~~> (N-1)th fibo.
    int second_prev = fibo(N-2); ~~> (N-2)th fibo   } Recursive call

    return prev + second_prev;  } Your work.
}
```

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	. . . . .
0	1	1	2	3	5	8	13	21	. . . . .

```
int fibo(int N)
{
① if (N == 1) return 0;
② if (N == 2) return 1;
③ int prev = fibo(N-1);
④ int second_prev = fibo(N-2);

⑤ return prev + second_prev;
}
```

N = 6

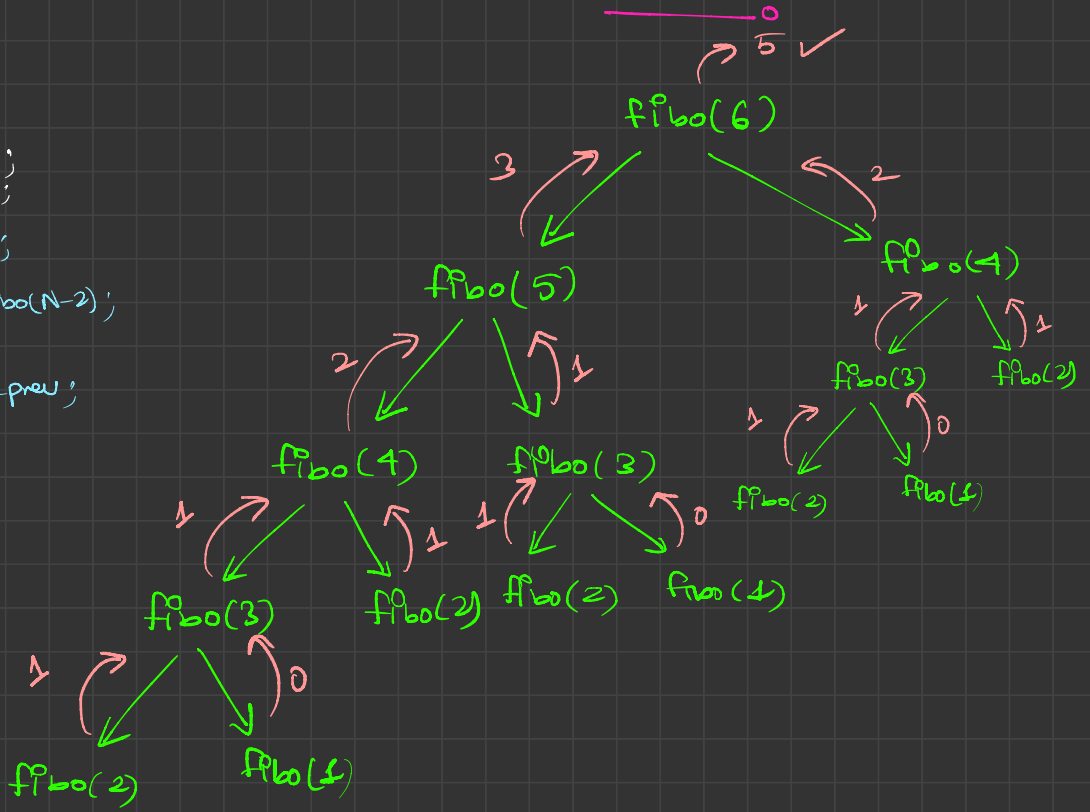
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	. . . . .
0	1	1	2	3	5	8	13	21	. . . . .

```
int fibo(int N)        // 6
{
    if (N == 1) return 0;
    if (N == 2) return 1;

    int prev = fibo(N-1);

    int second_prev = fibo(N-2);

    return prev + second_prev;
}
```

TC: $O(2^N)$

SC: $O(N)$

Recursive Tree



$Pow(x, n) \rightsquigarrow x^n$

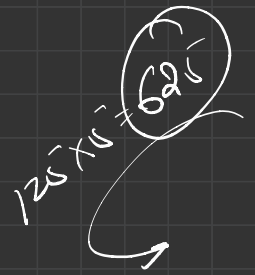
$x^0 = 1$ (Base case)

$Pow(x, n) = x * Pow(x, n-1)$

faith: returns $x, n-1$

```
int Pow (int x, int n)
{ ① if (n == 0) return 1;
  ② int a = Pow (x, n-1);
  ③ return x * a;
}
```

TC: O(N)
SC: O(N)



Call stack

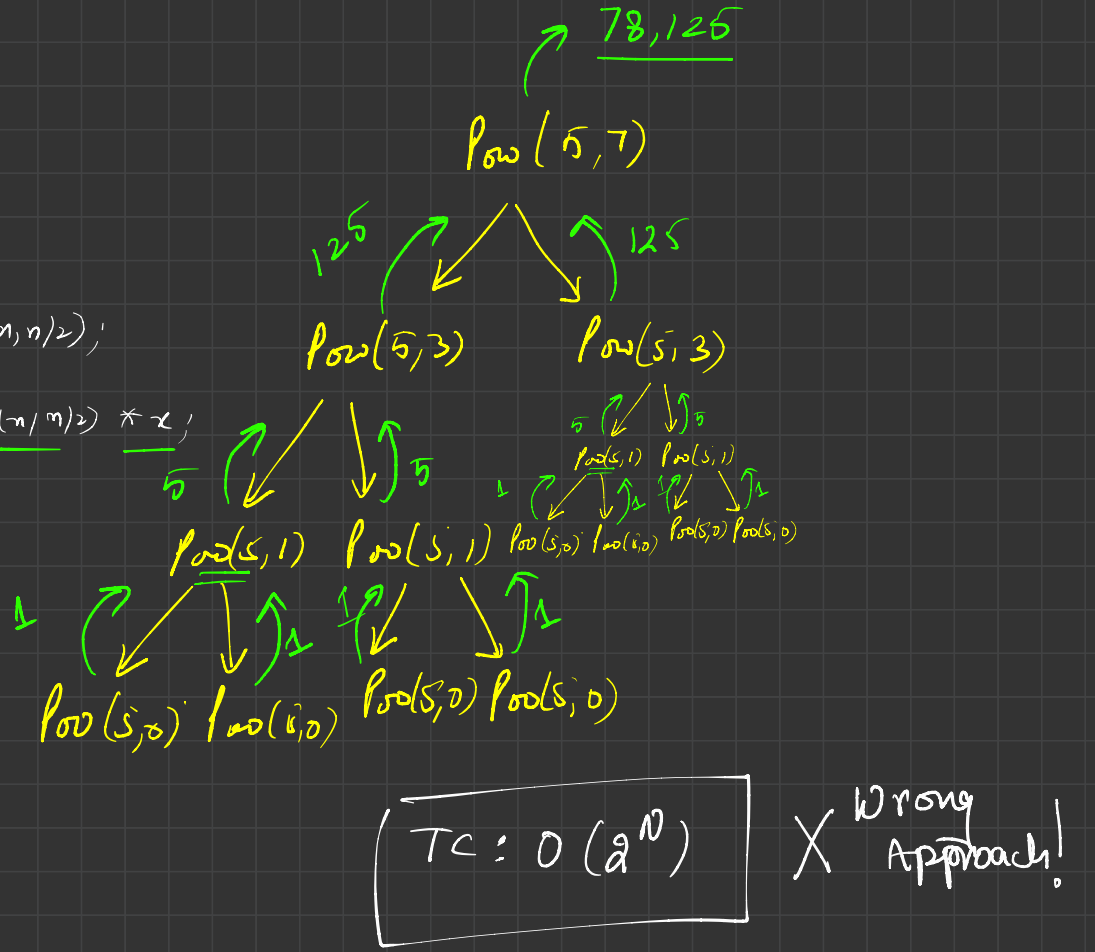
$$5^4 = 5^2 \times 5^2$$

$$5^7 = 5^3 \times 5^3 \times 5$$

$$pow(x,n) = \begin{cases} pow(x, n/2) * pow(x, n/2) & , n \in \text{even} \\ pow(x, n/2) * pow(x, n/2) * x & , n \in \text{odd} \end{cases}$$

```
int Pow(int x, int n)
{ if(n==0) return 1;

  if(n%2==0)
      return Pow(x,n/2)*Pow(x,n/2);
  else
      return Pow(x,n/2)*Pow(x,n/2)*x;
}
```

```
int Pow(int x, int n)
{ if (n == 0) return 1;

  if (n%2 == 0)
      return Pow(x,n/2) * Pow(x,n/2);
  else
      return Pow(x,n/2) * Pow(x,n/2) * x;
}
```

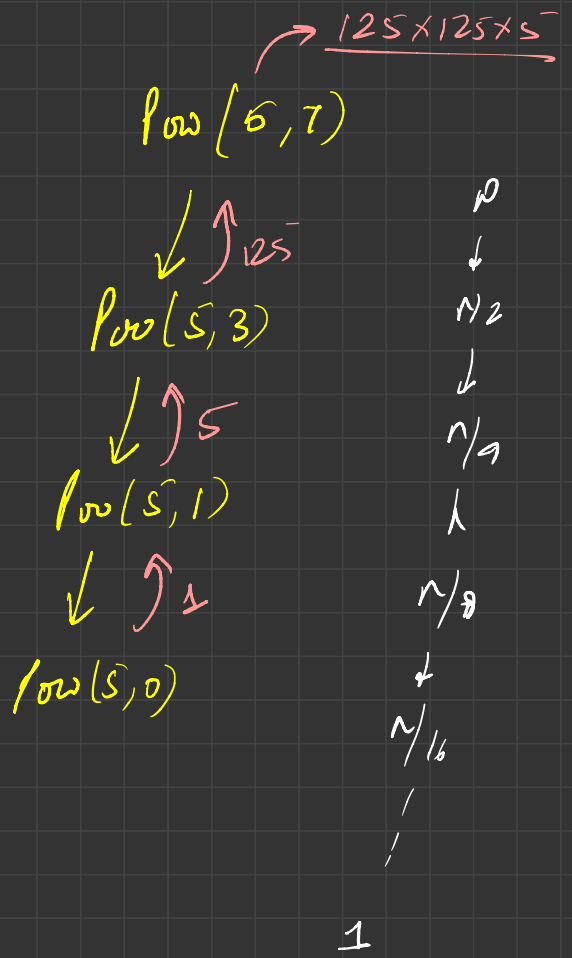


$TC: O(a^N)$ ✗ Wrong Approach!

```
int Pow (int x, int n)
{
    if (n == 0) return 1;

    int a = Pow (x, n/2);

    if (n%2 == 0)
        return a*a;
    else
        return a*a*x;
}
```

$TC : O(\log_2 N)$

$SC : O(\log_2 N)$



$$O(1) < O(\log_2 N) < N < N^2 < N^3 \ldots < 2^n < e^n$$