Todays Content

1. TC in Recursion
2. SC in Recursion

New Student:

Watch Rec1 & Rec2

Basics:

$T(n) = 3n + 10$

$T(5) = 3*5 + 10 = 25$

$T(7) = 3*7 + 10 = 31$

Recursive Relation:

$T(N) = T(N-1) + N$

$T(5) = T(4) + 5$

$T(3) = T(2) + 3$

$T(N-1) = T(N-2) + N-1$

$T(N) = 2T(N/2) + 1$

$T(N/2) = 2*T(N/4) + 1$

$T(N/4) = 2T(N/8) + 1$

$T(N) = T(N/2) + N$

$T(N/2) = T(N/4) + N/2$

Steps to Calulate TC:

1. Identify Recurence Relation & Base Case
2. Solve recursive Relation ⇒ Generalise the relation for k.
3. Get TC for Code.

```
int sum(int N){
    if(N==1){return 1;}
    return sum(N-1) + N;
}
```

Assume Time Taken of Sum(N) = T(N)

Sum(N-1) = T(N-1)

Recursive Relation: $T(N) = T(N-1) + 1$

Base Condition: $T(1) = O(1)$

Solve Recursive Relation.

$T(N) = T(N-1) + 1, \; T(1) = 1$

$T(N-1) = T(N-2) + 1$

$T(N) = T(N-2) + 1 + 1$

$= T(N-2) + 2$

$T(N-2) = T(N-3) + 1$

$T(N) = T(N-3) + 1 + 2$

$= T(N-3) + 3$

$T(N-3) = T(N-4) + 1$

$T(N) = T(N-4) + 1 + 3$

$= T(N-4) + 4$

Generalized Expression: After k substitutions, expression

$T(N) = T(N-k) + k \quad T(1) = 1$ // unknown term to known value

// $T(N-k) = T(1)$, $N-k=1$, $k = N-1$

$T(N) = T(N-(N-1)) + N-1$

$= T(N-N+1) + N-1$

$T(N) = T(1) + N-1 = 1 + N-1 = O(N)$

Trick TC in Recursion:

TC = #(no: of Function call) * (Time taken for each Function call)

TC = (N) x 1 ⇒ O(N)

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

Assume Time Taken for fact(N) = T(N)

↳ fact(N-1) = T(N-1)

Relation: $T(N) = T(N-1) + 1$ | Base: $T(0) = 1$ | ans. TODO: $T(N) =$

```
long Pow(int a, int n){
    if(n==0){return 1}
    return Pow(a, n-1) * a
}
```

Time Taken for Pow(a, n) = T(N)

↳ Pow(a, n-1) = T(N-1)

Relation: $T(N) = T(N-1) + 1$ | Base: $T(0) = 1$ | ans. TODO: $T(N) =$

// Fast Exponentiation

```
long Pow(int a, int n){
    if(n==0){return 1}
    long t = Pow(a, n/2);
    if(n%2==0){return t*t}
    else{return t*t*a}
}
```

Time taken for pow(a, n) = T(N)

↳ pow(a, n/2) = T(N/2)

Relation: $T(N) = T(N/2) + 1$ | Base1: $T(0) = 1$ | Base2: $T(1) = 1$ | ans.: $T(N) = \log N$

```
long Pow(int a, int n){
    if(n==0){return 1}
    if(n%2==0){
        return Pow(a, n/2) * Pow(a, n/2)
    }
    else{
        return Pow2(a, n/2) * Pow2(a, n/2) * a
    }
}
```

Time taken for pow(a, n) = T(N)

↳ pow(a, n/2) = T(N/2)

Relation: $T(N) = 2T(N/2) + 1$ | Base1: $T(0) = 1$ | Base2: $T(1) = 1$

$T(N) = O(N)$

Solve Recursive Relation.

$T(N) = T(N/2) + 1 \qquad T(0) = 1$

$\uparrow$ ——— $T(N/2) = T(N/4) + 1$

$T(N) = T(N/4) + 1 + 1$

$= T(N/4) + 2 \Rightarrow T(N/2^2) + 2$

$\uparrow$ ——— $T(N/4) = T(N/8) + 1$

$T(N) = T(N/8) + 1 + 2$

$= T(N/8) + 3 \Rightarrow T(N/2^3) + 3$

$\uparrow$ ——— $T(N/8) = T(N/16) + 1$

$T(N) = T(N/16) + 1 + 3$

$= T(N/16) + 4 \Rightarrow T(N/2^4) + 4$

After k Substituting

$T(N) = T(N/2^k) + k \qquad T(0) = 1$

$N/2^k = 0$, can we get k? *

$T(N) = T(N/2^k) + k \qquad T(0) = 1 \qquad T(1) = 1$

$N/2^k = 1, \quad N = 2^k, \quad k = \log_2 N$

// After $\log_2 N$ substitutions $T(N/2^k) = T(1)$

$T(N) = T(1) + \log_2 N$

$T(N) = O(\log_2 N)$

Recursive Relation

$T(N) = 2T(N/2) + 1 \qquad T(0) = 1$

$T(N/2) = 2T(N/4) + 1$ ← $2T(N/2) + 1$

$T(N) = 2[2T(N/4) + 1] + 1$ $\qquad 2^2 T(N/2^2) + 2^2 - 1$

$= 4T(N/4) + 3 \Rightarrow 2^2 T(N/2^2) + 2^2 - 1$ $\qquad 2^3 T(N/2^3) + 2^3 - 1$

$T(N/4) = 2T(N/8) + 1$ ← $2^4 T(N/2^4) + 2^4 - 1$

$T(N) = 4[2T(N/8) + 1] + 3$

$= 8T(N/8) + 7 \Rightarrow 2^3 T(N/2^3) + 2^3 - 1$

$T(N/8) = 2T(N/16) + 1$ ←

$T(N) = 8[2T(N/16) + 1] + 7$

$= 16T(N/16) + 15 \Rightarrow 2^4 T(N/2^4) + 2^4 - 1$

After k Substitutions:

$T(N) = 2^k T(N/2^k) + 2^k - 1 \qquad T(0) = 1$

$N/2^k = 0$ *

$T(N) = 2^k T(N/2^k) + 2^k - 1 \qquad T(0) = 1 \qquad T(1) = 1$

$N/2^k = 1, \quad N = 2^k, \quad k = \log_2 N$

$T(N) = N * T(1) + N - 1$

$= N * 1 + N - 1$

$= 2N - 1$

$T(N) = O(N)$

Fibanacci TC & SC


```
int Fib(int N){
    if(N==0 || N==1){
        return N
    }
    return Fib(N-1) + Fib(N-2)
}
```

Time taken for Fib(N) = T(N)

Fib(N-1) = T(N-1) Fib(N-2) = T(N-2)

Recursive Relation:

$$T(N) = T(N-1) + T(N-2) + 1$$

Base

$T(0) = 1$ $T(1) = 1$

TC:

Solve Recursive Relation:

$$T(N) = T(N-1) + T(N-2) + 1$$

$T(N-2) = T(N-3) + T(N-4) + 1$

$T(N-1) = T(N-2) + T(N-3) + 1$

$$T(N) = T(N-2) + T(N-3) + 1 + T(N-3) + T(N-4) + 1 + 1$$

$$= T(N-2) + 2T(N-3) + T(N-4) + 3$$

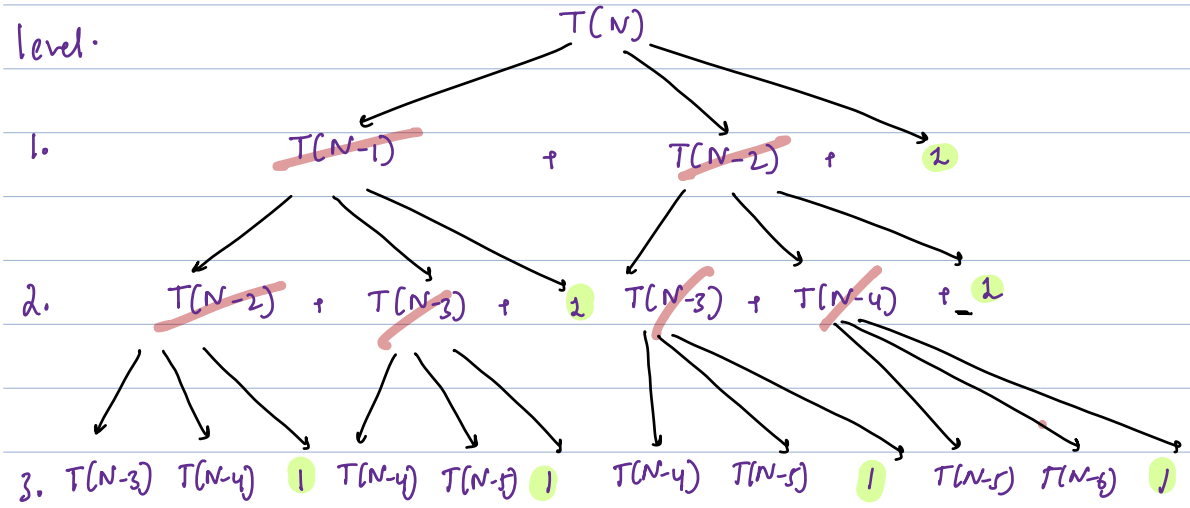
$T(N-4) =$

$T(N-3) =$

$T(N-2) =$

Note: We get more & more terms ☹

Recurence Tree



obs: level	man T( )	T( )	Total O(1)
1	$T(N-1)$	$2 = 2^1$	$1 = 2^1-1$
2	$T(N-2)$	$4 = 2^2$	$3 = 2^2-1$
3	$T(N-3)$	$8 = 2^3$	$7 = 2^3-1$
4	$T(N-4)$	$16 = 2^4$	$15 = 2^4-1$
5	$T(N-5)$	$32 = 2^5$	$31 = 2^5-1$
k ⟶	$T(N-k)$	2^k	2^k-1 $T(0)=1$ $T(1)=1$

$N-k=0,\ k=N$

After N Substitutions

N ⟶ $T(0)$ 2^N + 2^N-1

Recursive Relation Base

$T(N) = 2T(N/2) + N, \quad T(1) = 1$

$T(N/2) = 2T(N/4) + N/2$

$= 2[2T(N/4) + N/2] + N$

$= 4T(N/4) + 2N \Rightarrow 2^2 T(N/2^2) + 2N$

$T(N/4) = 2T(N/8) + N/4$

$= 4[2T(N/8) + N/4] + 2N$

$= 8T(N/8) + 3N = 2^3 T(N/2^3) + 3N$

$T(N/8) = 2T(N/16) + N/8$

$= 8[2T(N/16) + N/8] + 3N$

$= 16T(N/16) + 4N = 2^4 T(N/2^4) + 4N$

$2T(N/2) + N$

$2^2 T(N/2^2) + 2N$

$2^3 T(N/2^3) + 3N$

$2^4 T(N/2^4) + 4N$

After k Substitutions:

$T(N) = 2^k T(N/2^k) + kN \qquad T(1) = 1$

$N/2^k = 1, \quad N = 2^k \quad k = \log_2 N$

$= N * T[1] + \log_2 N \cdot N$

$= N + N \log_2 N$

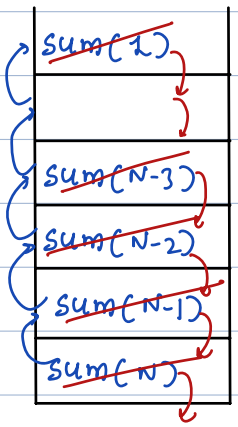
$T(N) = O(N \log_2 N)$

Space Complexity in Recursion

↳ Function calls are stored in stack, which we consider as extra space.

SC : Max Stack Size

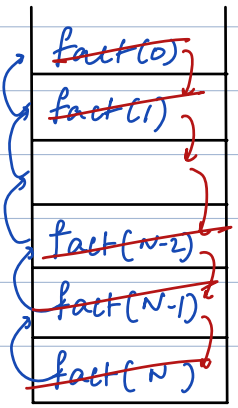
```
int sum(int N){
    if(N==1) {return 1;}
    return sum(N-1) + N;
}
```



MaxStackSize = N

0 → 1 → 2 → .. N → N-1 → N-2 → .. 0

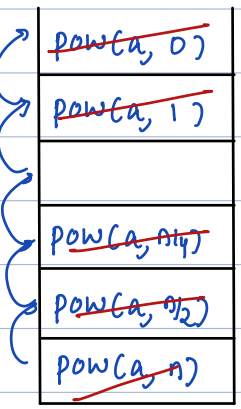
```
int fact(int N){
    if(N==0) {return 1;}
    return fact(N-1) * N
}
```



MaxStackSize = N+1

SC : O(N)

```
long Pow(int a, int n){
    if(n==0) {return 1}
    long t = Pow(a, n/2);
    if(n%2==0) {return t*t}
    else {return t*t*a}
}
```



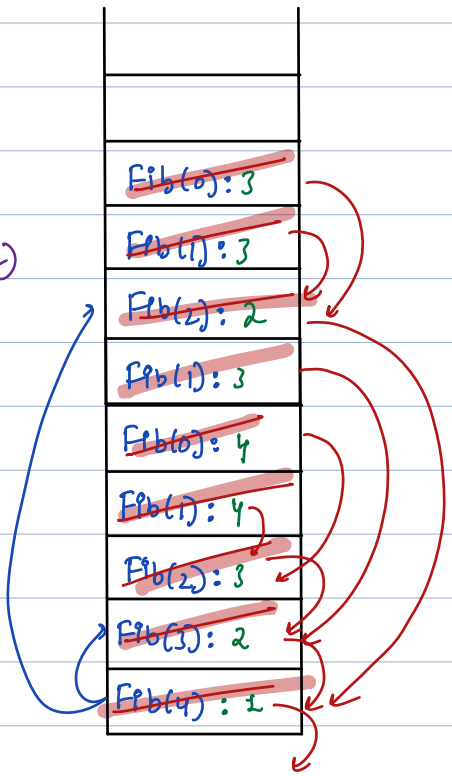
N	N/2	N/4	N/8	..	2	0
+1	+1	1	1		1	1

MaxStackSize = $\log_2^N + 1$

SC : $O(\log_2^N)$

Space Complexity:

```
int Fib(int N){   SC: O(N)
    if(N==0 || N==1){
        return N
    }
    return Fib(N-1) + Fib(N-2)
}
```

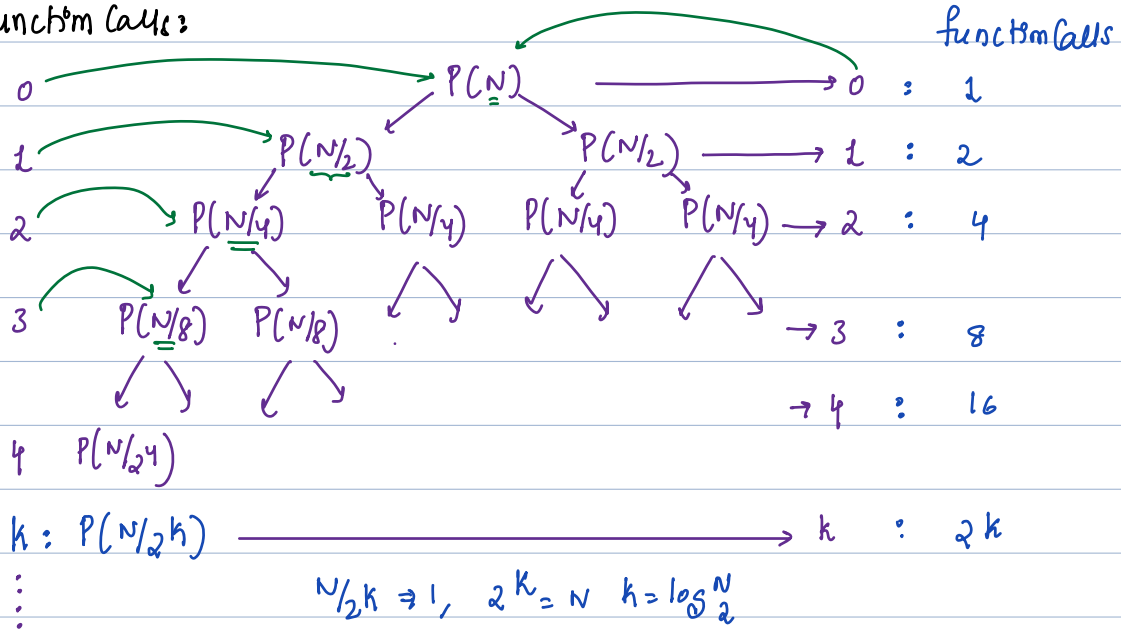


Trick TC in Recursion: Approximate Estimate

TC = #(no: of Function call) * (Time taken for each Function call)

```
long Pow(int a, int n){
    if(n==0){return 1}
    if(n%2==0){
    |   return Pow(a, n/2) * Pow(a, n/2)
    }
    else{
    |   return Pow2(a, n/2) * Pow2(a, n/2) * a
    }
```

Function Calls:



Total Function Calls:

$= 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots 2^{\log_2 N} =$

fr: $2^0 + 2^1 + 2^2 + \cdots 2^n = 2^{n+1} - 1$ $\quad a^{m+n} = a^m * a^n$ $\quad a^{\log_a N} = N$

$= 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots 2^{\log_2 N} = 2^{\log_2 N + 1} - 1 = 2^{\log_2 N} * 2^1 - 1$

$= 2 * 2^{\log_2 N} - 1 = 2N - 1$

TC : (2N-1) * (1) ⇒ O(N)