Todays Content

1. Time Complexity based on Recursive Relation
2. Time Complexity based on function calls
3. Space complexity : Max stack size

Basics:

$T(N) = 3N + 10$

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

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

Recursive Relation: Is a mathematical expression that define a sequence in terms of it's previous terms

Ex: $T(N) = T(N-1) + N$ $\quad T(N) = 2T(N/2) + 1$

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

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

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

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

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

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

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

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

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

Steps to Calculate TC:

1. Create Recursive Relation & Base Case
2. Solve recursive Relation

   a. Substitution method b. Recursive Tree method

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

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

Recursive Relation

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

Base Condition

$T(1) = 1$

Solve Recursive Relation:

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

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

Generalized Expression:

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

$T(1) = 1$

$N - K = 1, \quad K = N-1$

Substitute K = N-1

$$T(N) = N-1 + T(1)$$
$$= N-1+1$$
$$\boxed{T(N) = N}$$

```
long pow(long a, long N){
    if(N==0){return 1;}
    return pow(a,N-1) * a;
}
```

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

Recursive Relation

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

Base Condition

$T(0) = 1$

Solve Recursive Relation:

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

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

Generalized Expression:

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

$T(0) = 1$

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

Substitute k = N

$T(N) = N + T(0)$
$= N + 1$

Big O

$T(N) = O(N)$

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

Assume Time Taken for $pow(N) = T(N)$

Recursive Relation

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

Base Condition

$T(0) = 1$

$T(1) = 1$

Solve Recursive Relation:

$T(N) = 1 + 2T(N/2) \longrightarrow 1^{st}$ Sub $\longrightarrow 2^1 - 1 + 2T(N/2)$

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

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

$= 3 + 4T(N/4) \longrightarrow 2^{nd}$ Sub $\longrightarrow 2^2 - 1 + 2^2 T(N/2^2)$

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

$= 3 + 4 + 8T(N/8)$

$= 7 + 8T(N/8) \longrightarrow 3^{rd}$ Sub $\longrightarrow 2^3 - 1 + 2^3 T(N/2^3)$

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

$= 7 + 8 + 16T(N/16)$

$= 15 + 16T(N/16) \rightarrow 4^{th}$ Sub $\longrightarrow 2^4 - 1 + 2^4 T(N/2^4)$

Generalized Expression:

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

$N/2^k = 0$ * Cannot calculate k here, In these cases get value of $T(1)$

$T(N) = \underline{\underline{2}}^k - 1 + 2^k T(N/2^k)$ $\quad T(1) = 1$

$N/2^k = 1$ $\quad \underline{\underline{N = 2^k}}$

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

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

$= N - 1 + N * 1$

$= N - 1 + N$

$= 2N - 1$

Big O

$T(N) = O(N)$

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

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

Recursive Relation

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

Base Condition

$T(0) = 1 \quad T(1) = 1$

Solve Recursive Relation:

$T(N) = 1 + T(N/2) \longrightarrow 1 + T(N/2^1)$

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

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

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

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

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

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

Generalized Expression:

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

$N/2^k = 0$ # we cannot calculate k

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

$N/2^k = 1 \Rightarrow N = 2^k \Rightarrow k = \log_2 N$

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

$= \log_2 N + T(1)$

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

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

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

Recursive Relation

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

Base Condition

$T(0) = 1 \quad T(1) = 1$

Solve Recursive Relation:

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

$$\begin{cases} T(N-1) = 1 + T(N-2) + T(N-3) \\ T(N-2) = 1 + T(N-3) + T(N-4) \end{cases}$$

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

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

$$\begin{cases} T(N-2) \\ T(N-3) \\ T(N-4) \end{cases}$$

Note: When we have 2 or more different T() terms, substitution method is not suitable to solve recursive relation.

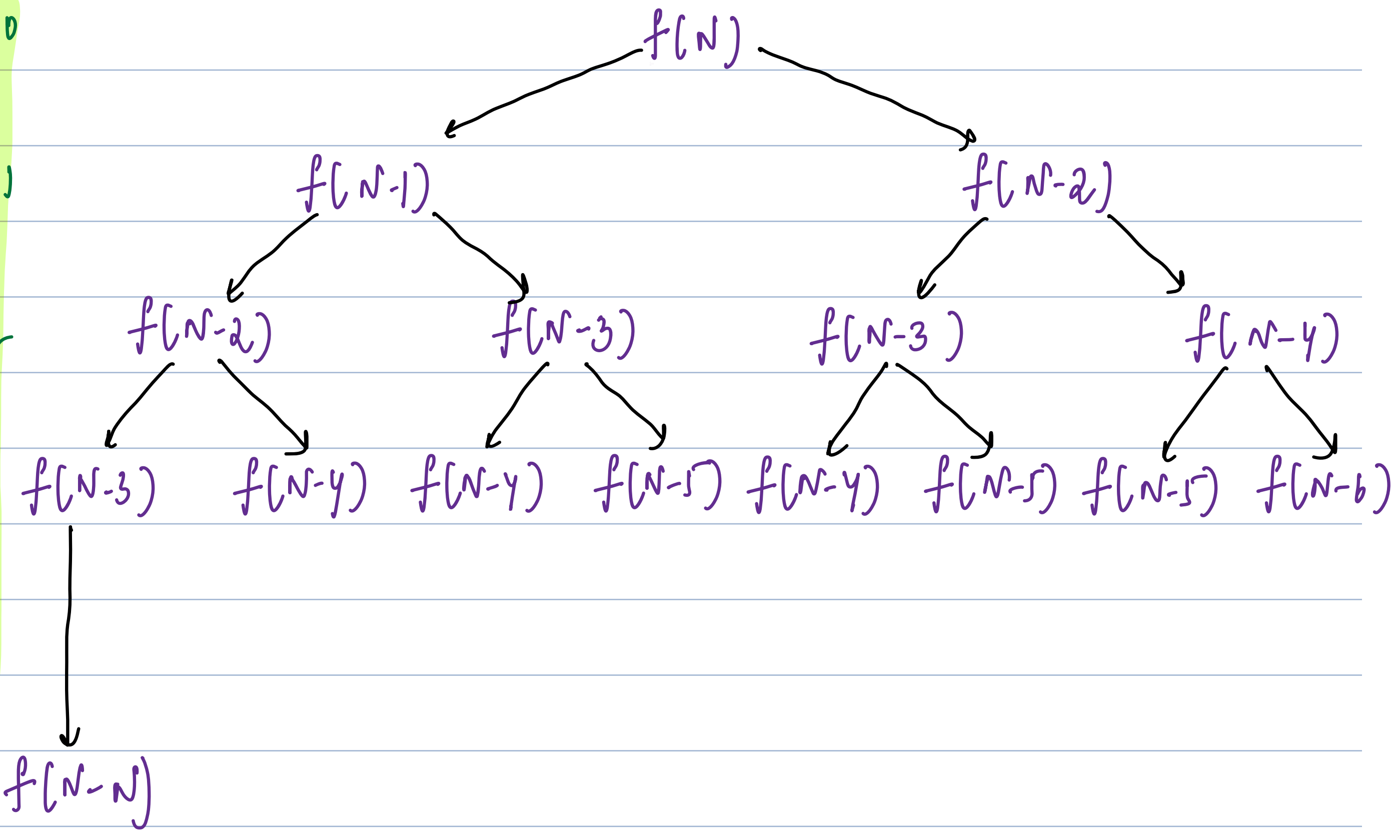
Generalized Expression: *

$\boxed{T(N) = O(\ )}$

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

2^{nd} Approach: # No: of function calls * Time taken for each function call

level	function calls
0	$1 = 2^0$
1	$2 = 2^1$
2	$4 = 2^2$
3	$8 = 2^3$
N	2^N



Total function calls $= 2^0 + 2^1 + 2^2 + \cdots 2^N = 2^{N+1} - 1 = 2 * 2^N - 1$

Time taken for each calls $= O(1)$

Final TC $= (2 * 2^N - 1) * O(1)$

$= 2 * 2^N - 1$

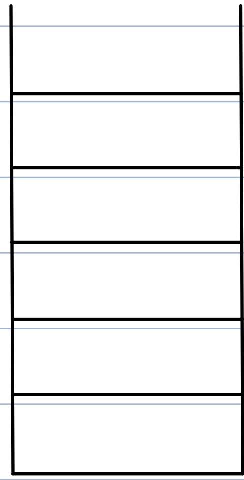
TC $= O(2^N)$

Space Complexity in Recursion:

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

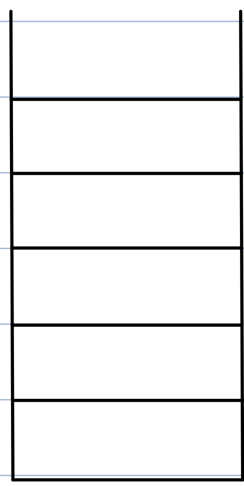
SC: Max stack size = Space used by recursion.

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



```
long pow(long a, long N){
    if(N==0){return 1;}
    return pow(a,N-1)*a;
}
```

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



Space Complexity:

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