Today's Content:

- Recursion Basics → function calling itself : Solving a problem using smaller instance of same problem (sub problem) called as recursion
- Fact() / power() / Fib()
- Recursive Relations
- Gray Code

Recursion Steps:

1) Assumption : Decide what your function does, assume it does

2) Main logic : Solving Assumption with Sub Problems

3) Base Condition : When to terminate

Factorial :

fact(3) = 3 * 2 * 1 = 6

fact(5) = 5 * 4 * 3 * 2 * 1 = 120

1! = 1 0! = 1

As per assumption

fact(4) : → 24

fact(6) : → 720

fact(N-1) : → (N-1)!

Ass: Given N, calculate & return N!

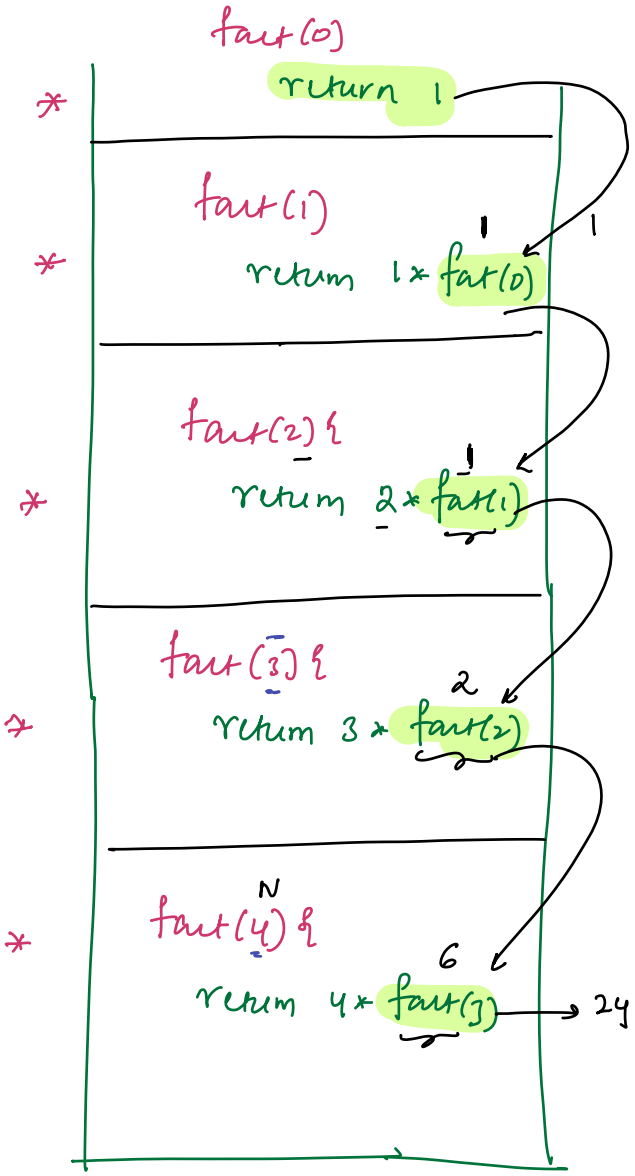
Note: If function returns n complete execution it comes out of stack.

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

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

Some function of N

T(0) = 1



Space Complexity:

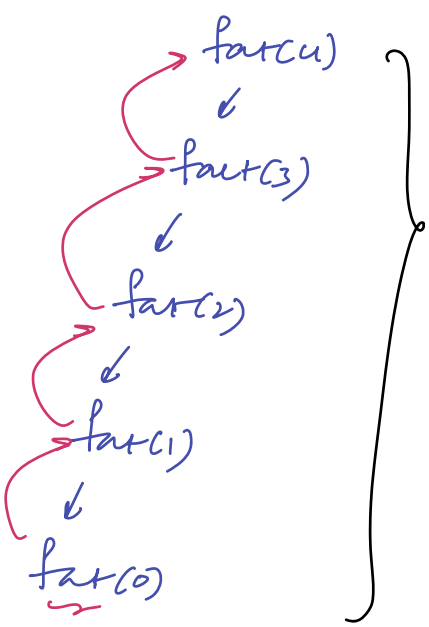
Stack memory:

Recursion Tree

Worst Case:

What your stack size = (N+1) = O(N)

SC: Max Stack size



Tree: Skewed Tree

SC: N+1 → O(N)

Time Complexity

Recurrence Relations: →

Used for TC

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

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

$= T(N-2) + 2$

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

$= T(N-3) + 3$

$= T(N-4) + 4$

$= T(N-k) + k$

$N-k = 0$

$k = N$

// know $T(0) = 1$

$= T(0) + N$

$= 1 + N$

$= N+1$

$= O(N)$

Fibonacci Series:

number	0	1	2	3	4	5	6	7
	0	1	1	2	3	5	8	13

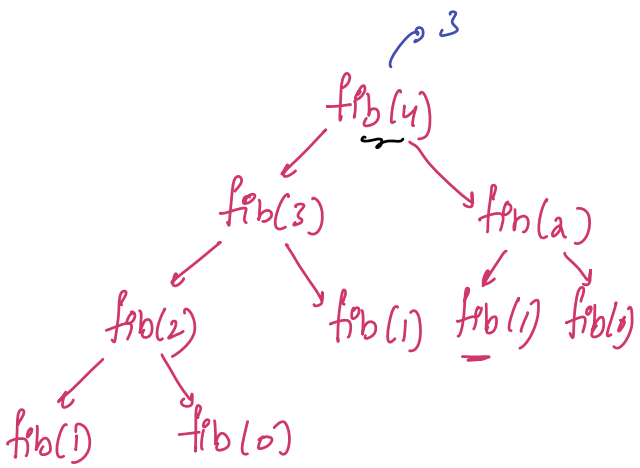
fib(0) = 0
fib(1) = 1

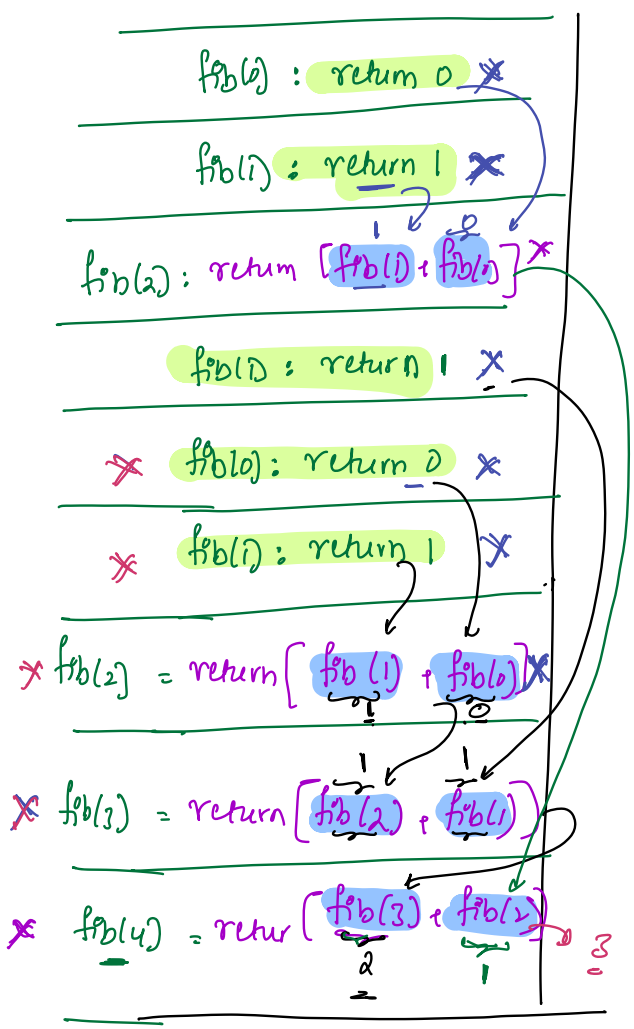
N^{th} fib = $N-1^{st}$ fib + $N-2^{nd}$ fib

Ass: Given N, it will return N^{th} fib number

```
int fact(N){
    if(N==0){return 0}
    if(N==1){return 1}
    return {fib(N-1) + fib(N-2)}
}
```

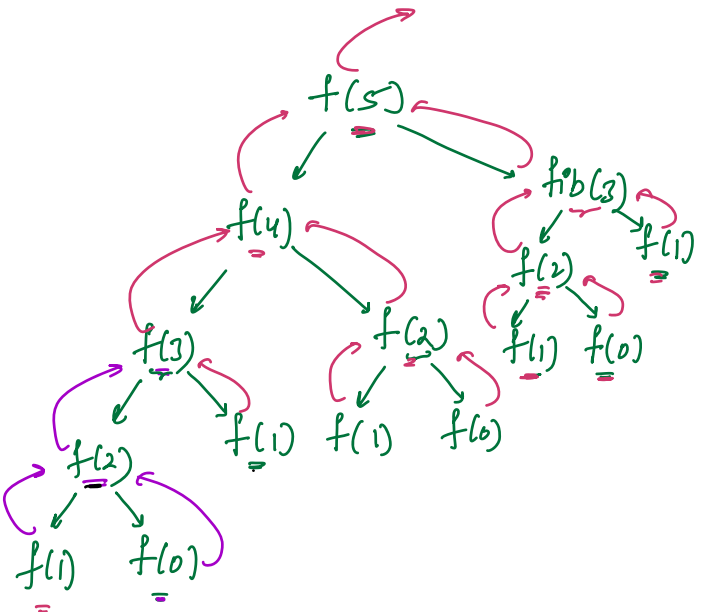
$T(N) = T(N-1) + T(N-2) + 1$





SC:

Recursive Tree:



TC:

Recursive Relation:

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

$T(N) = 2^N$ → doubt Session

Call function: stack size increase +1

If returns: stack size decrease -1

Max: 5

SC: Height of Tree

↳ SC: O(N)

Power function:

// Given a & N (+ve)
: Calculate a^N

$a=2,\ N=5 \rightarrow 2^5 \} \rightarrow \{32\}$

$a=3,\ N=4 \rightarrow 3^4 \} \rightarrow \{81\}$

// Given a, N → Calculate & Return a^N

Main logic

$a^{10} = a^9 * a$

$a^{10} = a^5 * a^5$

$a^{14} = a^7 * a^7$

$a^{15} = a^7 * a^7 * a$

$a^{21} = a^{10} * a^{10} * a$

$a^n \Rightarrow$ { if (n % 2 == 0) $a^{n/2} * a^{n/2}$; else $a^{n/2} * a^{n/2} * a$ }

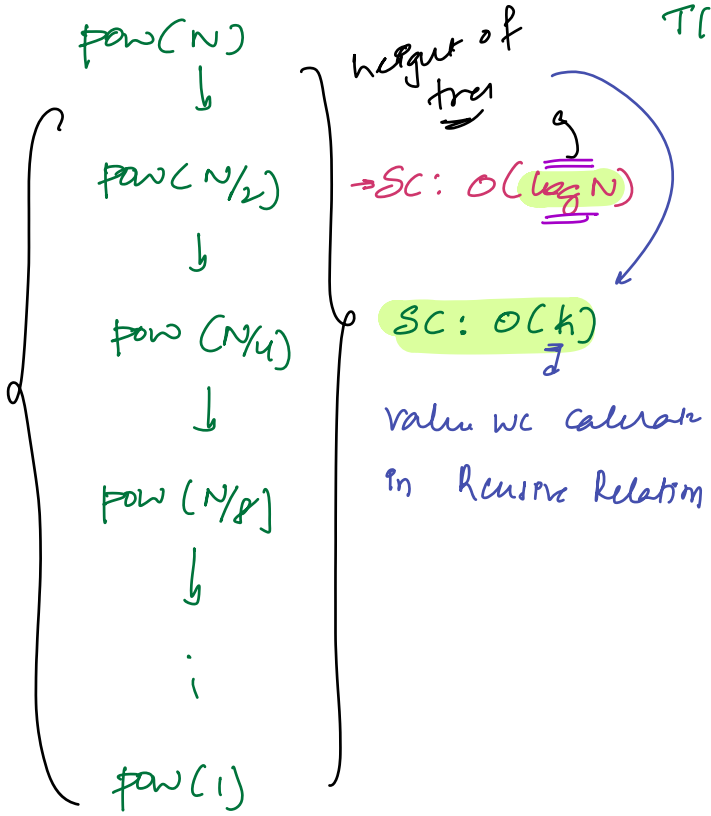
T(N) → function of N

Main logic

$a^n \Rightarrow$ { $p = a^{n/2}$; if (n % 2 == 0) $p * p$; else $p * p * a$ }

```
int power (a, n) {
    if (a==0) {return 0}
    if (n==0) {return 1}
    p = power (a, n/2) {a^(n/2)}
    if (n%2 == 0) {
        return p*p
    }
    else {
        return p*p*a
    }
}
```

Recursion Tree



TC:

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

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

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

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

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

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

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

// generalized

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

$N/2^k = 1$

$2^k = N$

$k = \log_2 N$

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

$= T(1) + \log_2 N$

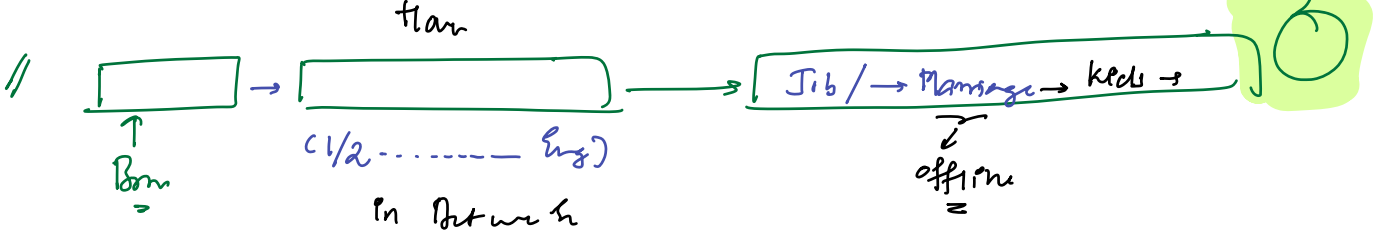
TC: $O(\log_2 N)$

10:43 PM

Zomat → Swiggy → Zepto

Swiggy Snow

Zomat mart



Gray Code:

Given N, generate all N bit numbers → 2^N → $[0, 2^N - 1]$ → {gray Code}

Note: Numbers in sequence should differ by Exactly 1 bit.

// N = 2

0 0 — 1 bit
0 1 — 2 bit
1 0 — 1 bit
1 1 — 1 bit

// Not gray Sequence

// N = 2

0 0 — 1 bit
0 1 — 1 bit
1 1 — 1 bit
1 0 — 1 bit

[0 1 3 2]

N = 2

1 0 — 1 bit
1 1 — 1 bit
0 1 — 1 bit
0 0 — 1 bit

[2 3 1 0]

N = 2

0 0 — 1 bit
1 0 — 1 bit
1 1 — 1 bit
0 1 — 1 bit

[0 2 3 1]

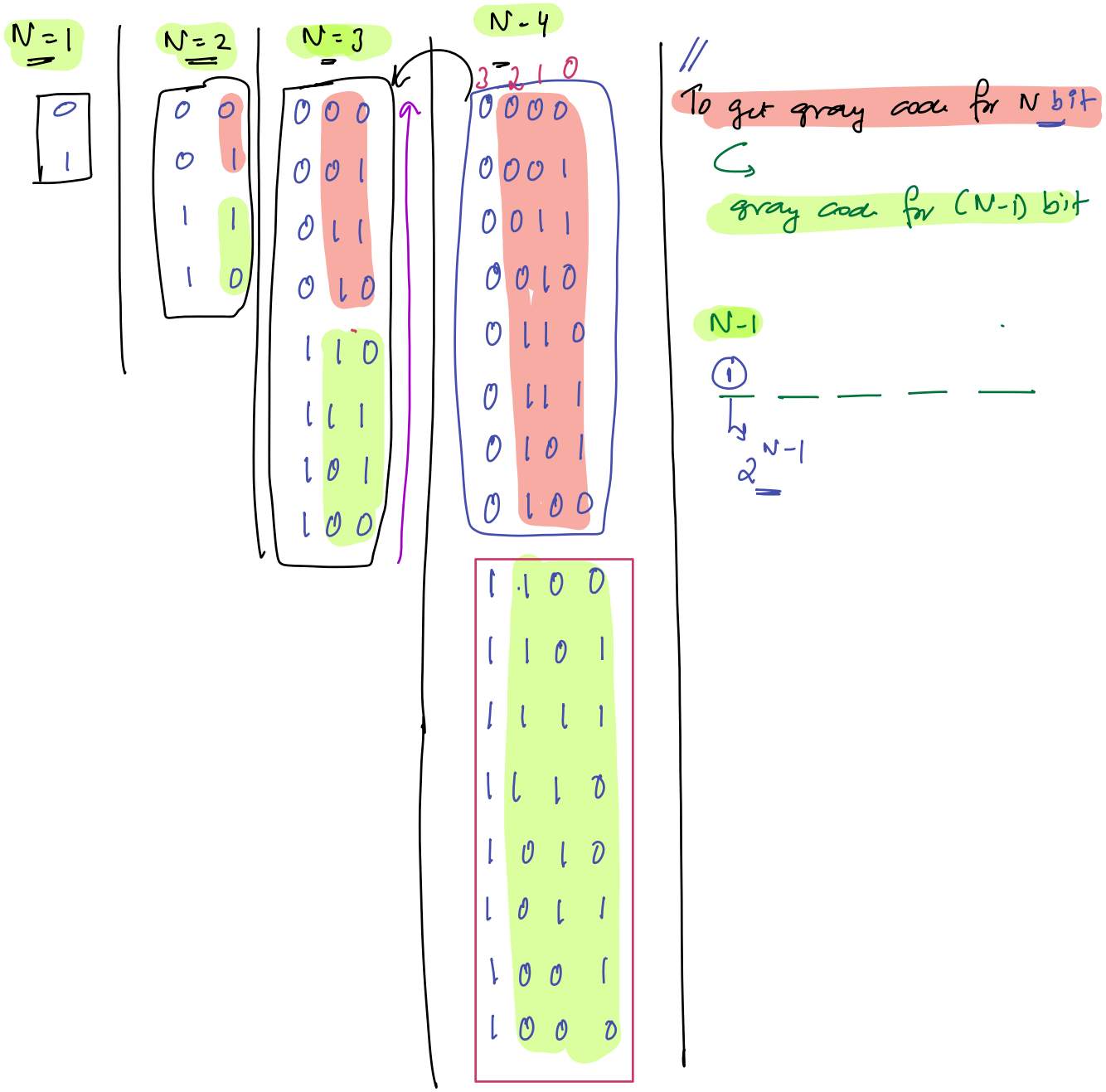
lexicographically → {print}

N = 3 → [0 7]

0 0 0 — 1 bit
0 0 1 — 1 bit
0 1 1 — 1 bit
0 1 0 — 1 bit
1 1 0 — 1 bit
1 0 0 — 1 bit
1 0 1 — 1 bit
1 1 1 — 1 bit

N = 4 → [0 15]

Q): Any gray code →



(We need to return gray code in decimal format)

// Ass: given N → { return gray code sequence of N bit numbers }

```
list<int> grayCode(int N)
{
    if(N==1) { list d; d.insert(0) d.insert(1) }
             return d

    list p = grayCode(N-1)
    int n = p.size(); → { n = 2^(N-1) Elements }
    list ans;
    for(i=0; i<n; i=i+1){
        ans.insert(p[i])
    }                                   → n
    for(i=n-1; i>=0; i--){
        ans.insert(p[i] + 2^(N-1))
    }                                   → n
    return ans;
}
```

$n = 2^{N-1}$

$2n = 2 \times 2^{N-1}$

2^N elements

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

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

$T(N-1) = T(N-2) + 2^{N-1}$

$= T(N-2) + 2^{N-1} + 2^N$

$T(N-2) = T(N-3) + 2^{N-2}$

$= T(N-3) + 2^{N-2} + 2^{N-1} + 2^N$

$T(N-3) = T(N-4) + 2^{N-3}$

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

C

C

C → keep Substing :

C

$2^0 + 2^1 + 2^2 + 2^3 - \cdots - 2^N$

$a = 1$

$r = 2$

$t = N+1$

Sum of gp : $\frac{a * (r^t - 1)}{r - 1}$

$= \frac{1 * (2^{N+1} - 1)}{2 - 1}$ → $(2^{N+1} - 1)$

→ $2 + 2^N - 1$

→ TC : → $O(2^N)$

// SC:

F(N) → F(N-1) → F(N-2) - - — F(0)

Recursion Stack : O(N)

//

Recursion: Towers of Hanoi → Romin's

↳ Sorting Based:

→ { Doubts }

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