# Fibonacci Series

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 |

## Recursion

- Base case condition
- Recursive function call

$T(n)$

```
fib(n) {
    if (n == 1) {          // ① Base case condition
        return n;
    }
    else {                 // ② Recursive function call
        return fib(n-1) + fib(n-2);
    }
}
```

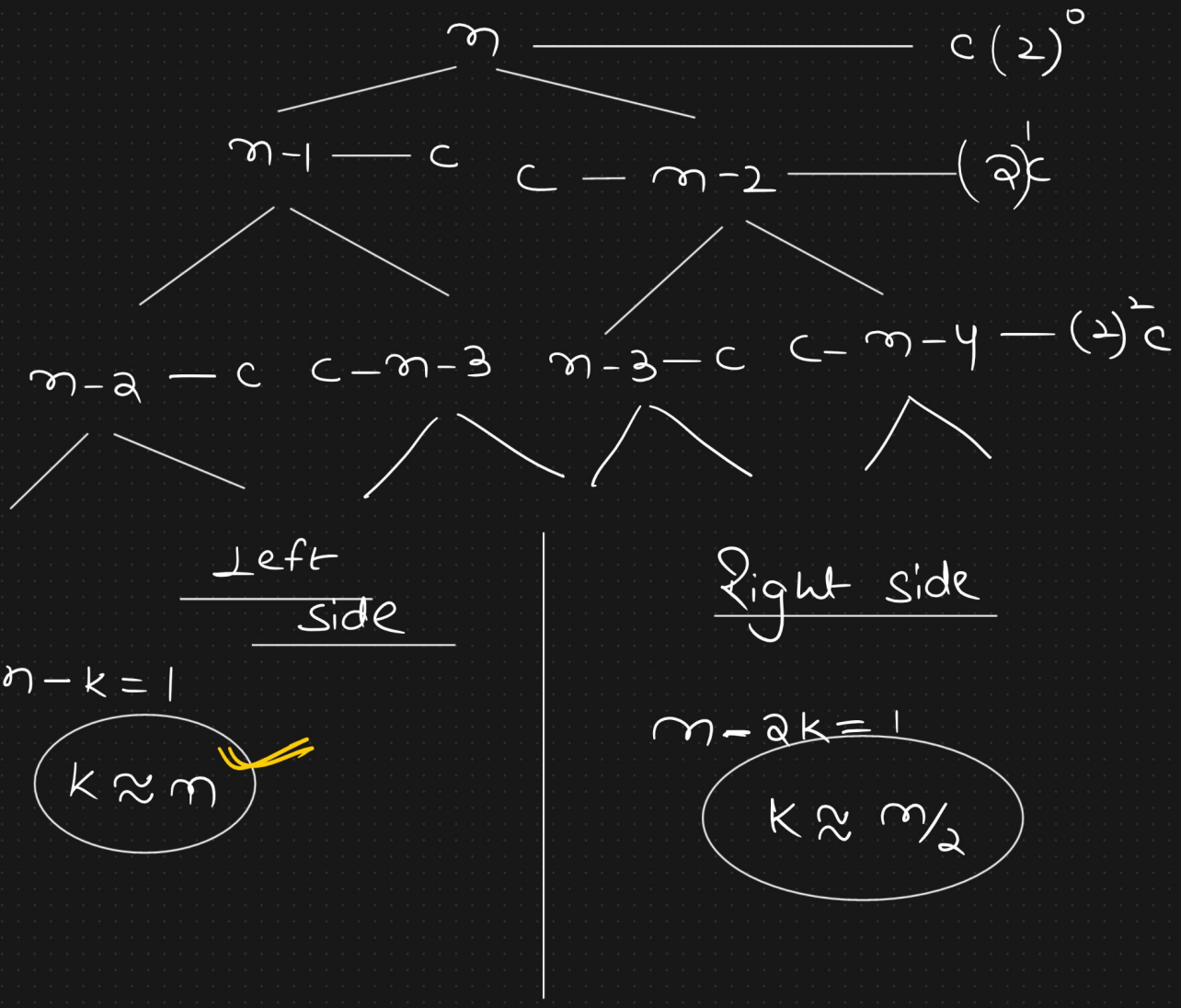
C

$T(n-1) + T(n-2)$

# Recurrence Relation

→ Recursive Tree Method

$T(n) = T(n-1) + T(n-2) + c$




Left side

$n - k = 1$

$k \approx n$

Right side

$n - 2k = 1$

$k \approx n/2$

GP series

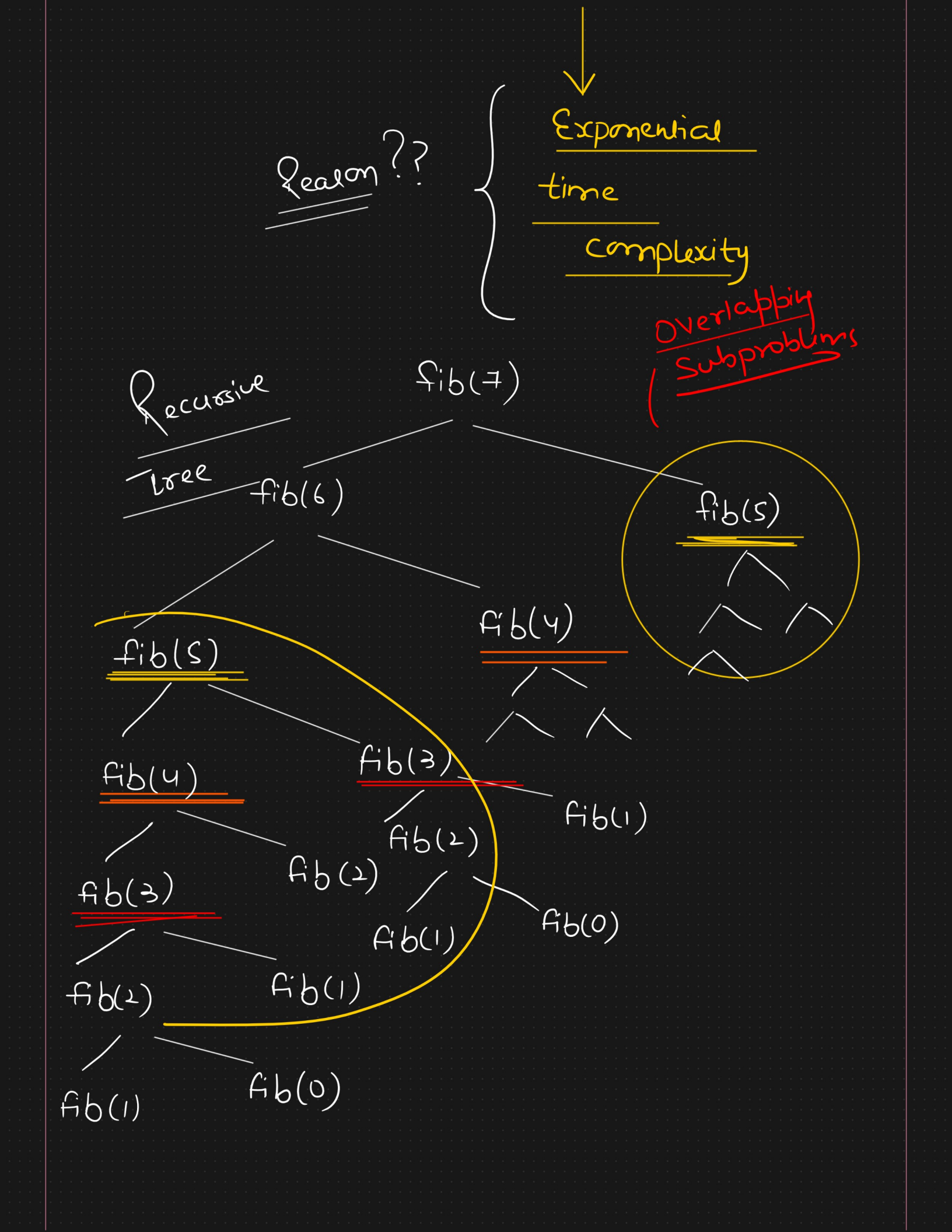
$r = 2$

$\text{Sum} = \frac{a(r^n - 1)}{r - 1}$

$c(2^0 + 2^1 + 2^2 + \dots + 2^k)$

$c(2^0 + 2^1 + 2^2 + \dots + 2^n)$

$c\left(\frac{2^n - 1}{1}\right) \approx O(2^n)$


Reason ??
Exponential time complexity
Overlapping Subproblems
Recursive tree
fib(7)
fib(6)
fib(5)
fib(5)
fib(4)
fib(4)
fib(3)
fib(1)
fib(2)
fib(2)
fib(3)
fib(0)
fib(1)
fib(1)
fib(2)
fib(0)
fib(1)

$n \longrightarrow$ Very very large

$\geq 35$

↳ Time Limit Exceeded

↓

Re-computation of same function calls internally

① choices, Overlapping subproblem

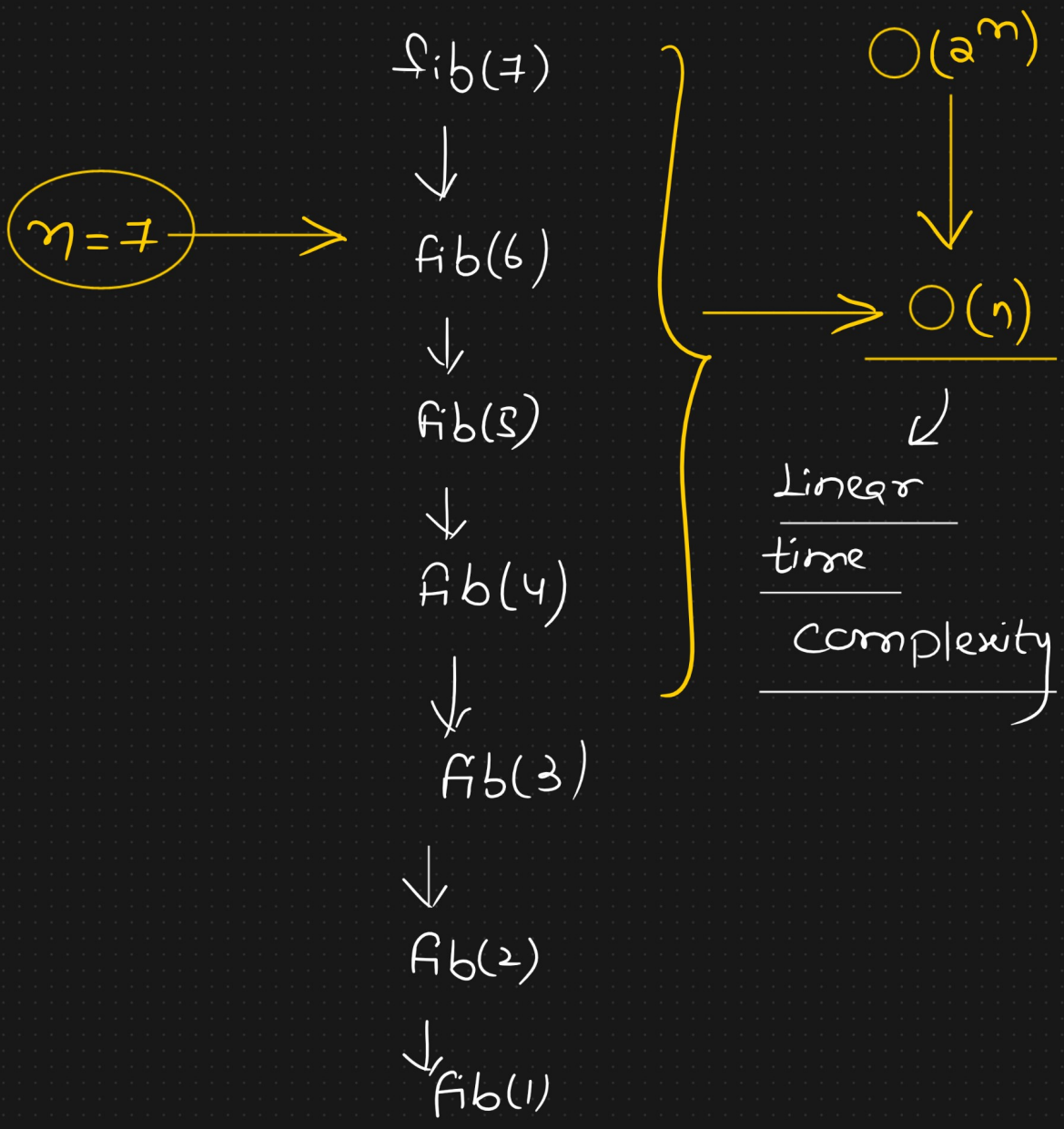
② Optimization (Maxima/Minima)

Dynamic Programming

① Memoization → Recursion

↳ Store the result of every unique function call

Exponential time Complexity




fib(n) — value

Hashtable

| key | value |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 5 |
| 6 | 8 |

SC = O(n) (array)

## Tabulation

→ No Recursion

fib

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 |

fibonacci(n) → O(n) time complexity

fib(0) = 0

fib(1) = 1

No Recursion

```
for(i = 2 ; i <= n; i++)
{
    fib(i) = fib(i-1) + fib(i-2);
}
return fib(n);
}
```

(Stack space)

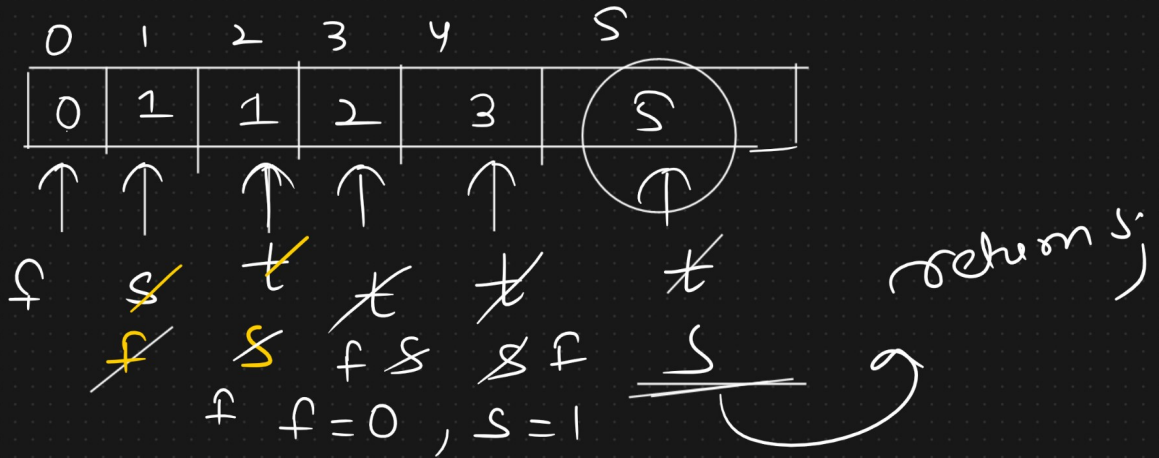
| Memoization | Tabulation |
|---|---|
| → Recursion | No Recursion |
| → Slower | faster as comparable to Memo. |

① Recursion → ② Memoization → ③ Tabulation

④ More optimized approach

time complexity - O(n)

Space complexity - O(1)




```
for(i=2; i<=n; i++)
{
    t = f+s;
    f = s;
    s = t;
}
```

return S;

}

- time complexity - O(n)
- Space complexity - O(1)

No use of any external data structure