google interview

Algorithm paradigm.

↳ 1 f2f → 2dp

3 f2f → 2dp

- Brute force
- Divide n Conquer
- Greedy
- Dynamic Programming (DP)

dunno

↳ a dsa

Q: You're given a number n, figure out the n^{th} fibonacci

↳ Recursive Sol^n → any i^{th} term is the sum of prev 2 terms.

1st term

0, 1, 1, 2, 3, 5, 8, 13, 21.....

0th term

$$f(n) = f(n-1) + f(n-2)$$

This func^n returns the n^{th} fib

Base Case → $n == 0 \rightarrow 0$

$n == 1 \rightarrow 1$

State of dp

Repeating Subproblems

Lot of code which is exec again

if we remember whatever subproblem we have already solved

f(5) → 5

f(4) → 3

f(3) → 2

f(2) → 1

f(1) — Base Case

f(0) — Base Case

f(1) — B.C

f(0) — B.C

f(2) → 1

f(1) — B.C

f(0) — B.C

f(3) → 2

f(2) → 1

f(1) — B.C

f(0) — B.C

f(1) — B.C

For worst case → Total recursive calls

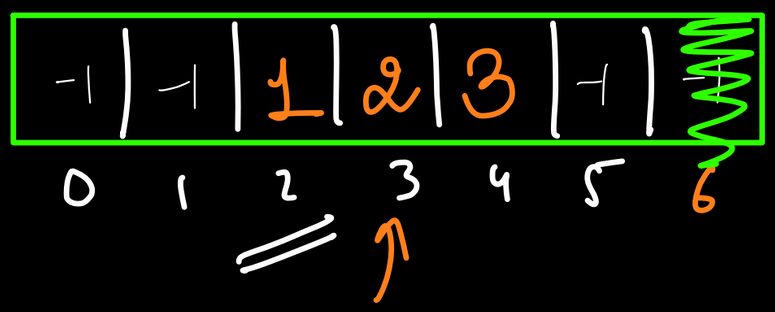
$$\underbrace{2^0 + 2^1 + 2^2 + 2^3 \cdots\cdots\cdots 2^n}$$

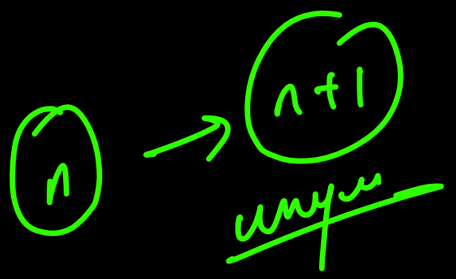
Sum of GP → $\frac{1 \times (2^{n+1} - 1)}{2 - 1} \rightarrow 2^{n+1} - 1$

↳ TC → $O(2^n)$

```
f(n) {
    if( n==0 || n==1 ) return n;
    return f(n-1) + f(n-2);
}
```

$f(n, m) = f(n, m-1) + f(n-1, m) \longrightarrow 2d$





if the value at an index of the array is not -1, means we have sometime back already computed it.

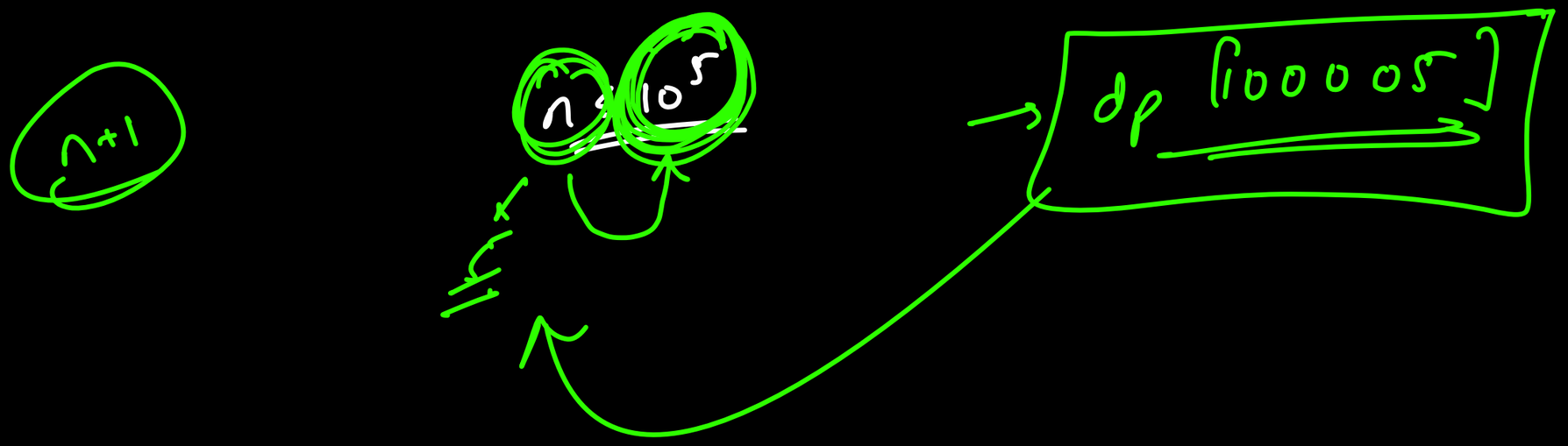
n-levels

on each level max 2 calls

$O(2^n)$

Top -> Down DP

Memoisation

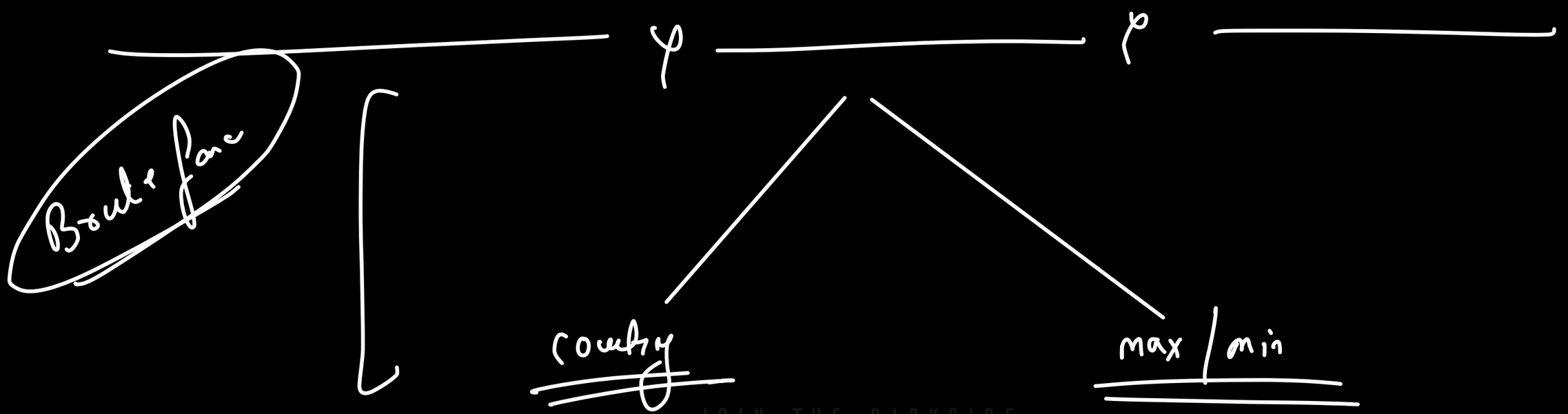


→ How to identify if a sol^n is a DP sol^n ??

→ Repeating Subproblems (overlapping subproblems)

→ Optimal Substructure

↳ if you have optimal ans of smaller subproblem, and using this optimal ans you can calc optimal ans of bigger problem, then the problem is said to have optimal Substructure

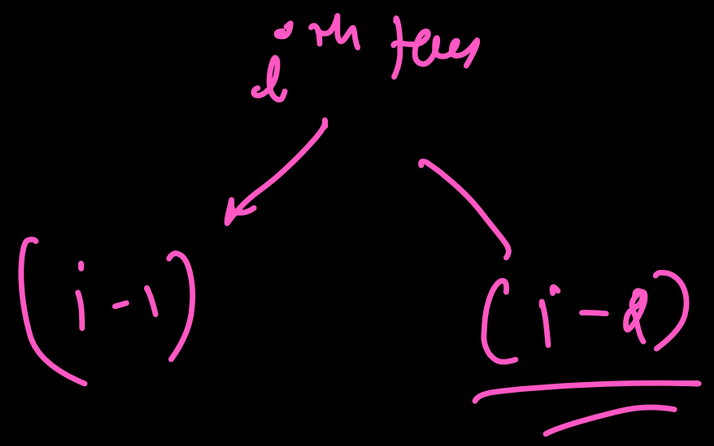


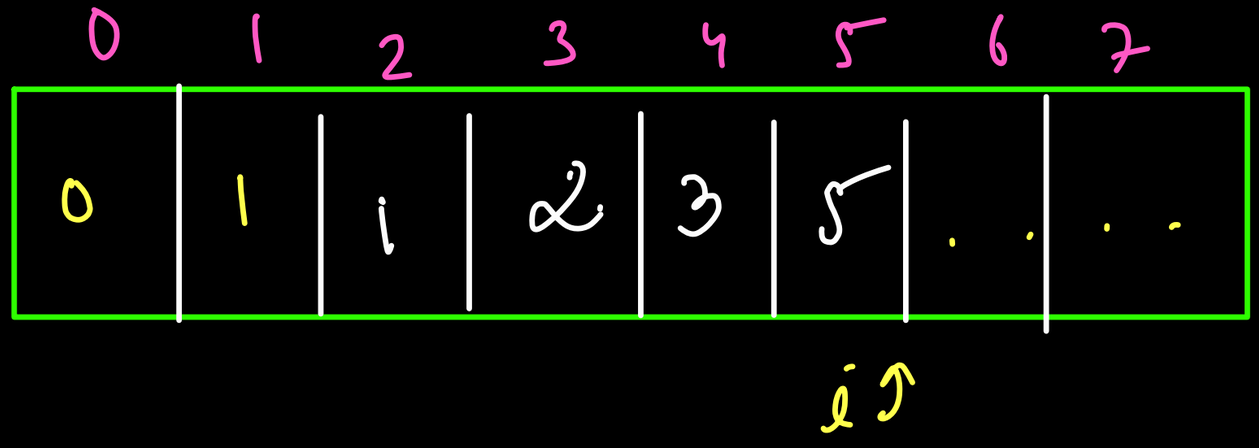
→ DP & DnC → merge sort

φ φ φ

we can build an iteration sol^n to solve the above prob.

$$f(n) = f(n-1) + f(n-2)$$





$(n == 0) \rightarrow 0$

$(n == 1) \rightarrow 1$

$dp[0] = 0$

$dp[1] = 1$

$dp[i] \rightarrow i^{th}$ fibonacci

for loop identifies the order in which we have to build the subproblem

Bottom Up DP → Tabulation

```
for (i=2 ; i<=n ; i++) {
    dp[i] = dp[i-1] + dp[i-2]
}
```

$\Rightarrow$

$f(i) = f(i-1) + f(i-2)$

N → friends N = 3 → 4 A B C

either keep them single

or pair them up.

way-1 → (A) (B) (C)

way-2 → (A B) (C)

way-3 → (A) (B C)

way 4 → (B) (A C)

N = 4 → 10

A B C D

(A) (B) (C) (D)

(A) (BC) (D)

(A) (BD) (C)

(A) (B) (DC)

(B) (AC) (D)

(B) (AD) (C)

(B) (A) (CD)

(C) (AB) (D)

(C) (AD) (B)

(C) (A) (BD)

i=3

A B C

Brute force

ans → f(n)

$f(i)$ = no. of ways $i-1$ friend can be paired $f(i-1)$ + $(i-1) * f(i-2)$

$f(i)$ → return no. of ways to pair i friends

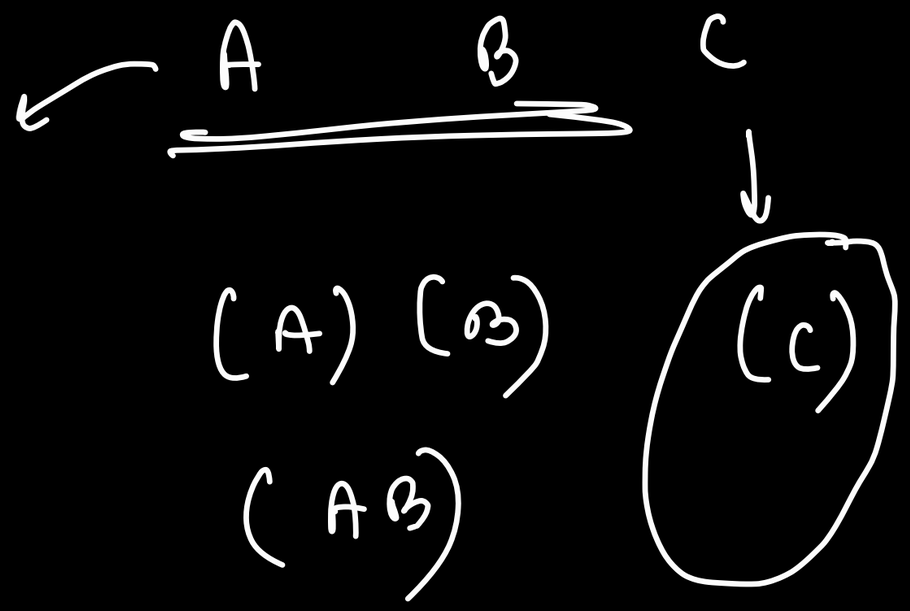
$f(i-1)$ → no. of ways to pair i friends, when i^{th} friend wants to be Single.

$(i-1) * f(i-2)$ → i^{th} friend decides to pair up

No. of ways i friends pair up when i^{th} friend always go in a pair

Base case →
- $i == 0 \rightarrow 0$
- $i == 1 \rightarrow 1$
- $i == 2 \rightarrow 2$

A B → [(A)(B), (AB)]



if C decides to stay single, & we have any of how many A, B i.e C & find Combination, then this arr will be same as arr

g no. g ways in which C stays single.

A B C → Single

D

4

(A) (B) (C) (D)

(AB) (C) (D)

(AC) (B) (D)

(A) (CB) (D)

A B C D → pairing

1) D - A

2) D - B

(DB) (A) (C)

(DB) (AC)

3) D - C

(DC) (A) (B)

(DC) (AB)

2 ((DA) (B) (C)
  (DA) (BC)

$$\frac{(4-1)\times f(4-2)}{1}$$

$3 \times f(2)$

$3 \times 2 \rightarrow 6$

How many partner choices D has got?

3

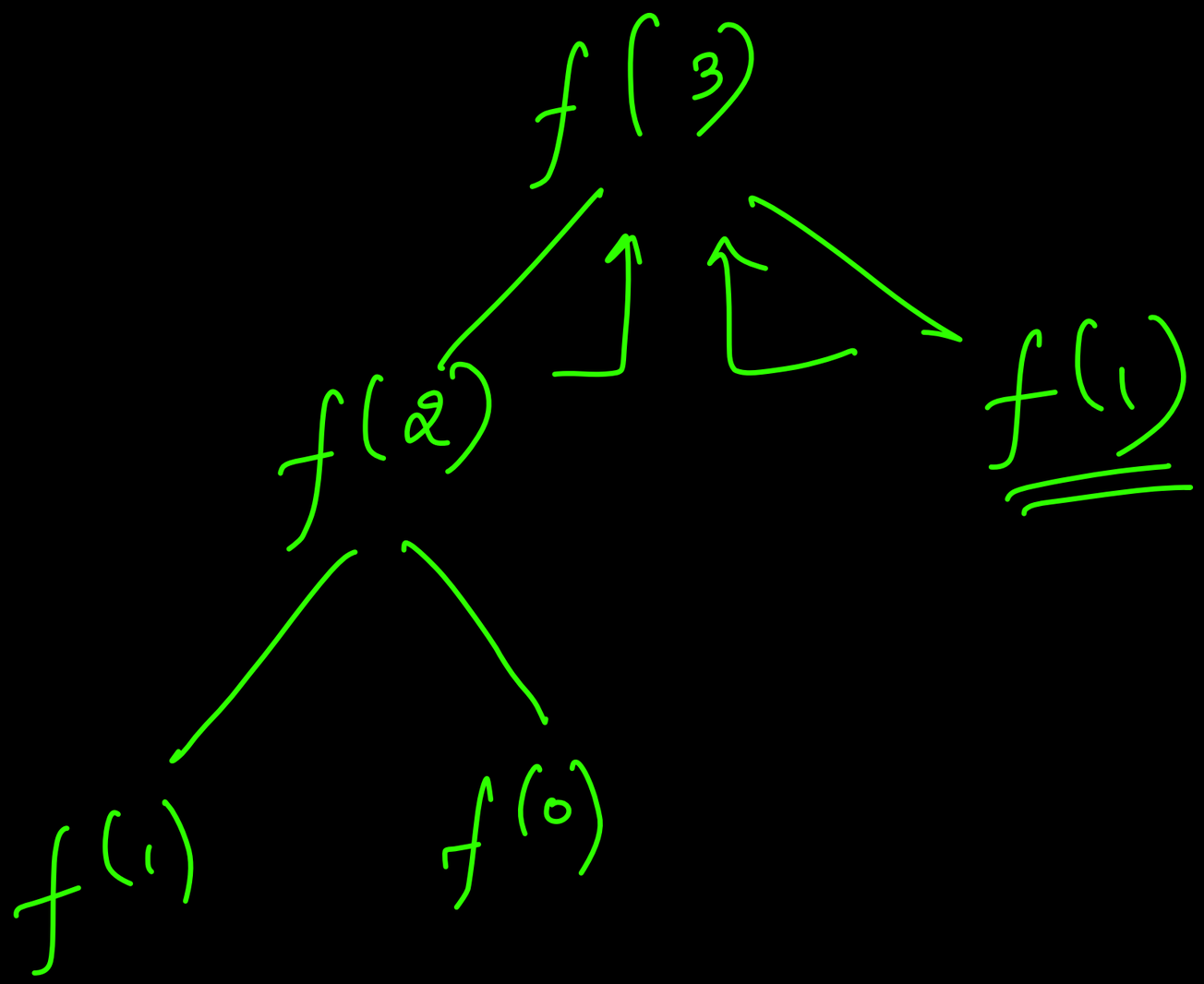
so for any general i, if i^{th} frnd wants a pair, then it has $(i-1)$ candidates to pair with.

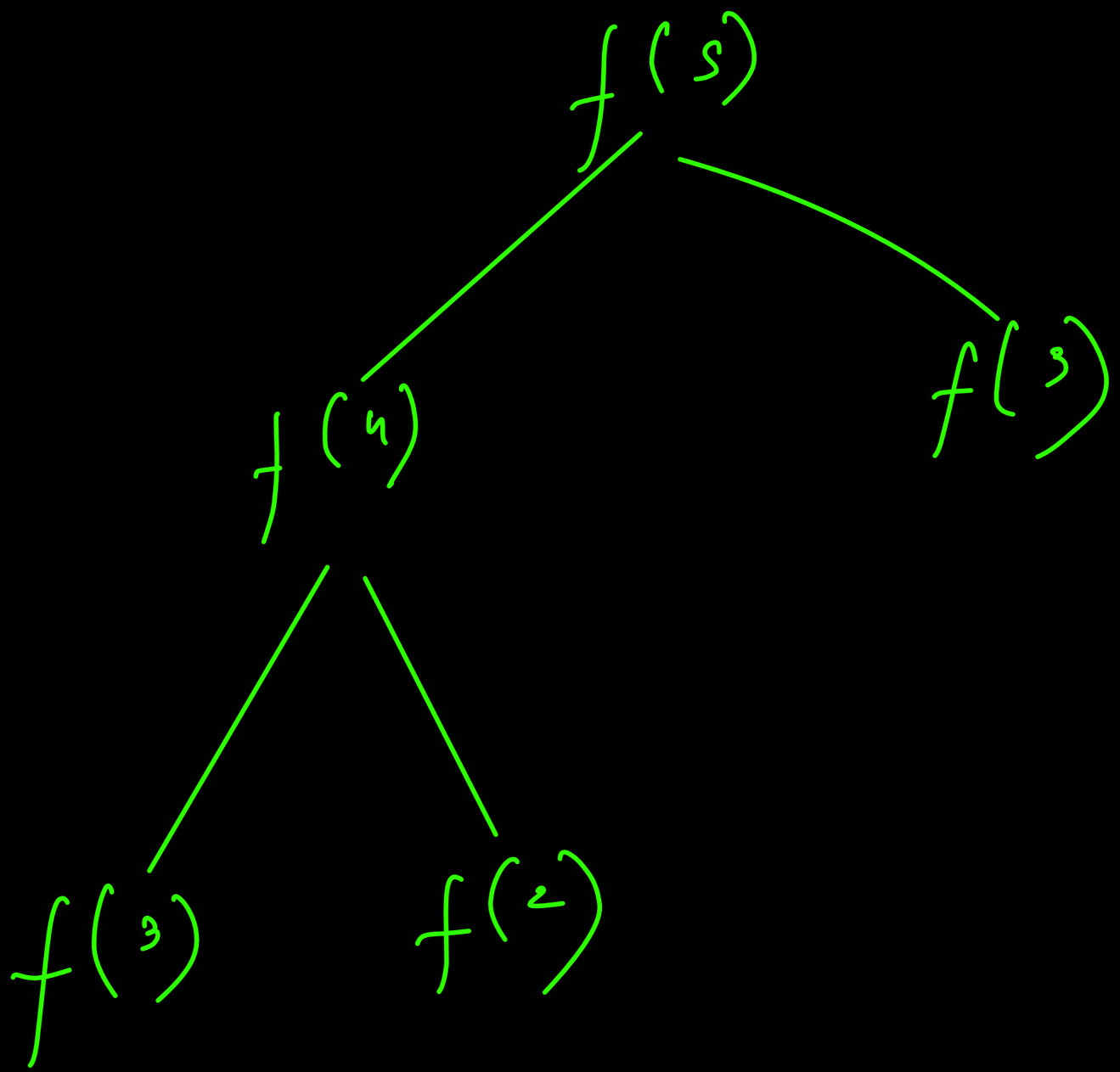
$$(i-1) \times f(i-2)$$

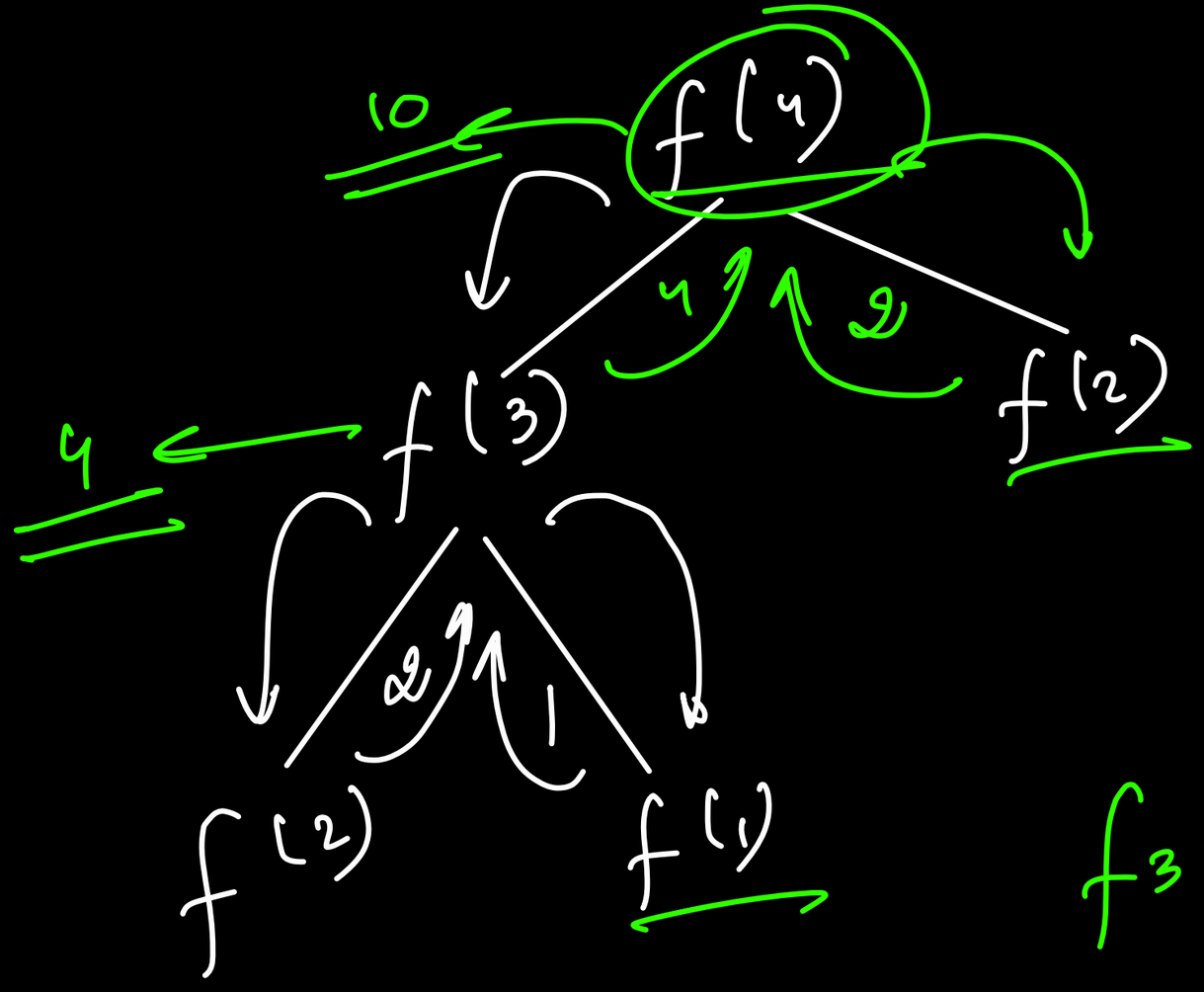
D-A D-C D-B

$$\frac{f(i-2) + f(i-2) + f(i-2)}{3 f(i-2)}$$

$$(i-1)^{x} f(i-2)$$







$f_4 = f_3 + (4-1) f_2$

$= 4 + (3) \times 2$

$\Rightarrow 4 + 6$

$= 10$

$f_3 = f_2 + (3-1) f(1)$

$\Rightarrow 2 + 2 \times 1$

$\Rightarrow 4$