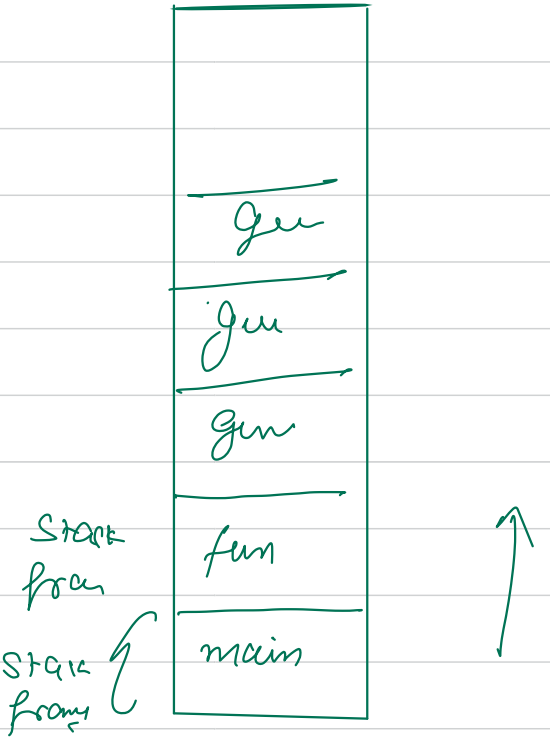
Recursion

↳ Recursion is a programming technique where any function calls itself again & again to solve some subproblem with a extra memory buffer.

So in the memory allocated to the process there are 2 parts

1. Stack memory → one stack frame
2. Heap memory

```
int main() {
    fun();
}
void fun() {
    gun();
}
void gun() {
    puts("shoot");
    gun();
}
```



Q: Given a number N, find N! recursively.

whenever we want solve a problem recursively →

→ 1) Recursion hypothesis or assumption ✓ (K x K-1)

→ 2) Self Work ✓

→ 3) Base Case ✓ → Smallest subproblem for which we already know the ans

* Principal of Mathematical Induction

By induction

Let's prove that sum of first n natural no. is $\frac{(n)(n+1)}{2}$

$n=1$ ans $\rightarrow 1$ $\frac{1\times 1+1}{2} = 1$ $\rightarrow$ Base Case

assume

$n=k$ $\frac{k(k+1)}{2}$ $\longrightarrow$ hypothesis

Let's prove for $n=(k+1)$ self work

1) Base Case $n = 1$ $n! = 1$

2) Recursive hypothesis → $f(n) = n * f(n-1)$

go and calc $f(n-1)$

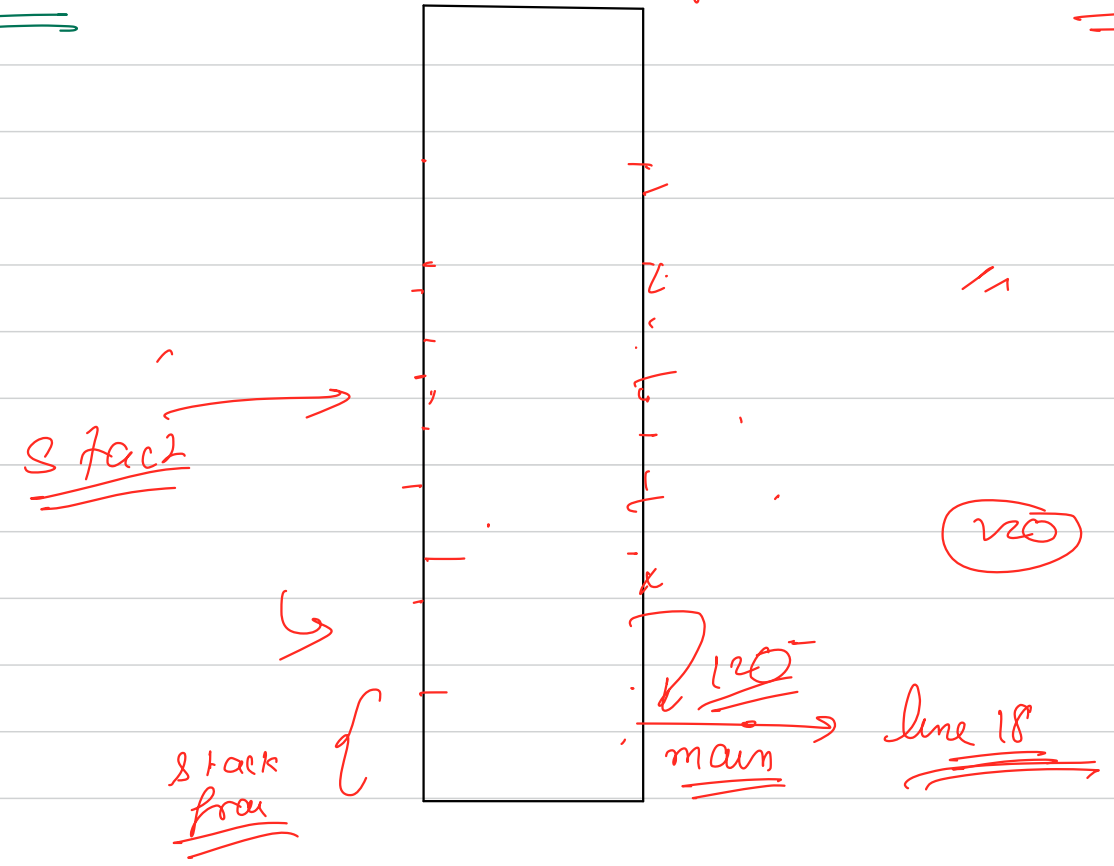
func^n f which returns $n!$

Recurrence Relation

3) Self work → multiply n & $f(n-1)$ & return

Stack trace

when we get return from func^n the stack frame is removed



merge sort, quick sort, sq root → Recursion

Recursion tree

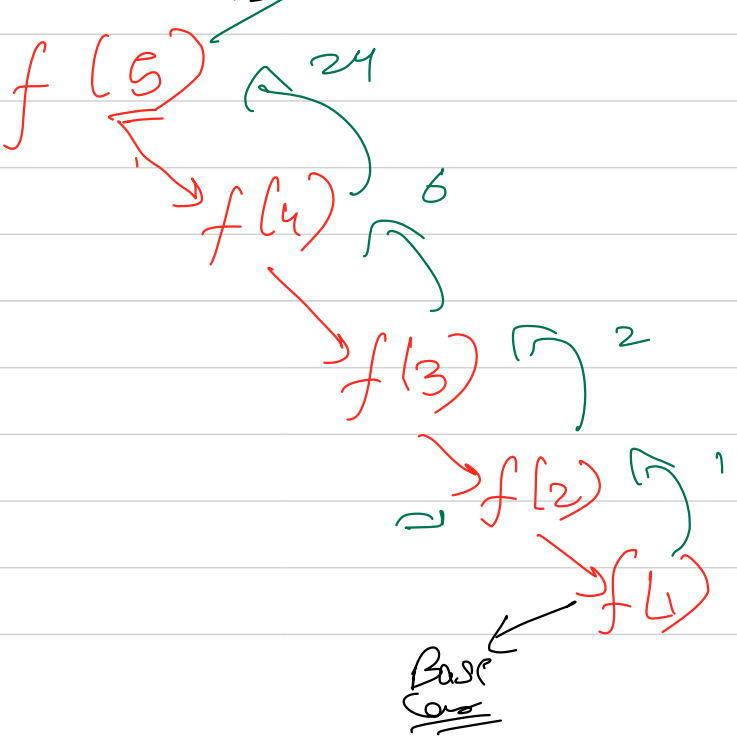
↳ $f(n) = n * f(n-1)$ → 120

operations

$O(N)$

space

$O(N)$



Q → Given the no. N, Calculate N^{th} fibonacci

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

0^{th} fibonacci

1^{st} fibonacci

2^{nd} fib

3^{rd} fib

4^{th} fib - - - - -

Recursively

Base Case

$n = 0 \rightarrow$ fib $\rightarrow 0$

$n = 1 \rightarrow$ fib $\rightarrow 1$

Recursive hypothesis $\rightarrow$ function $\rightarrow f(n)$ ✓

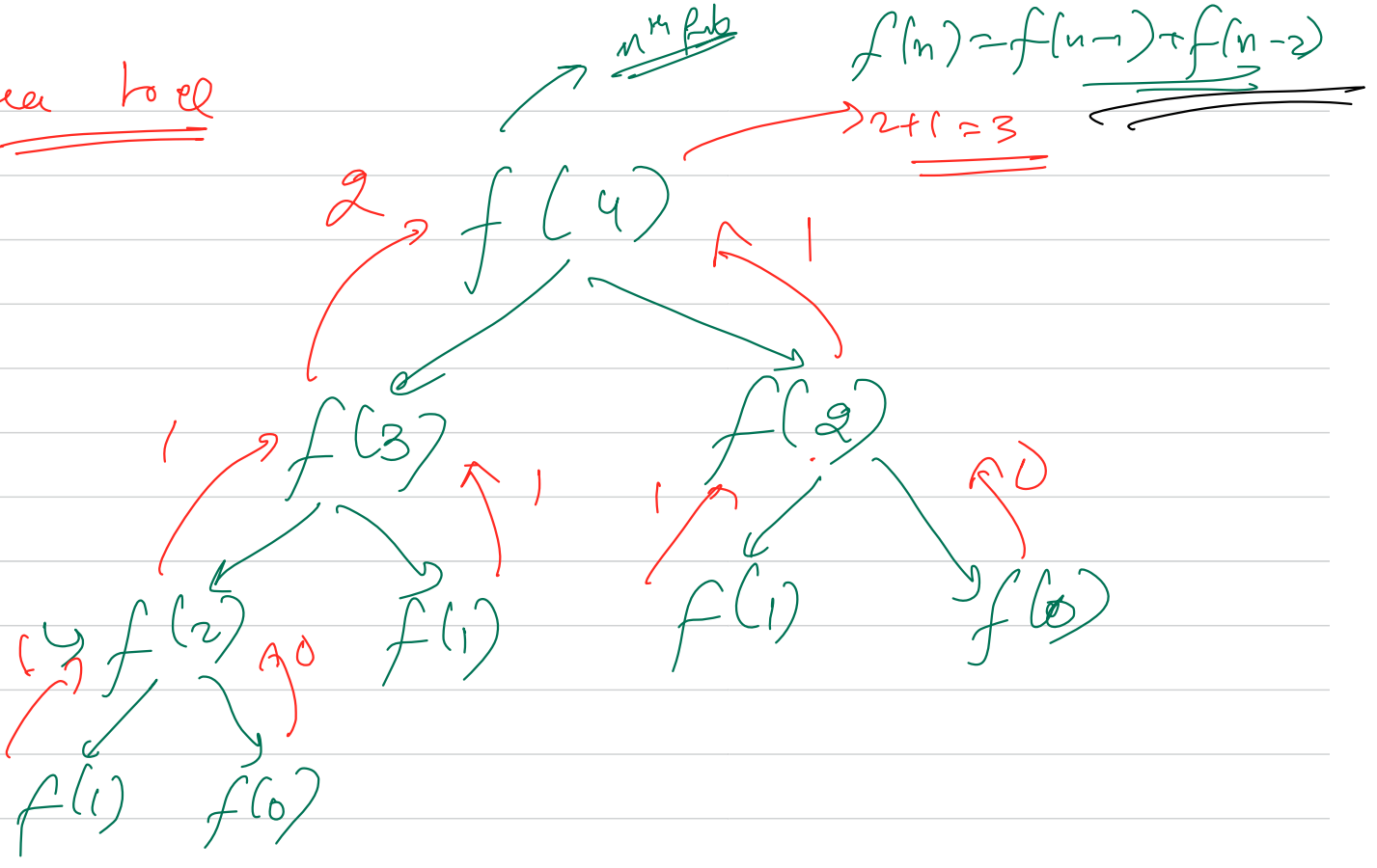
returns n^{th} fibonacci

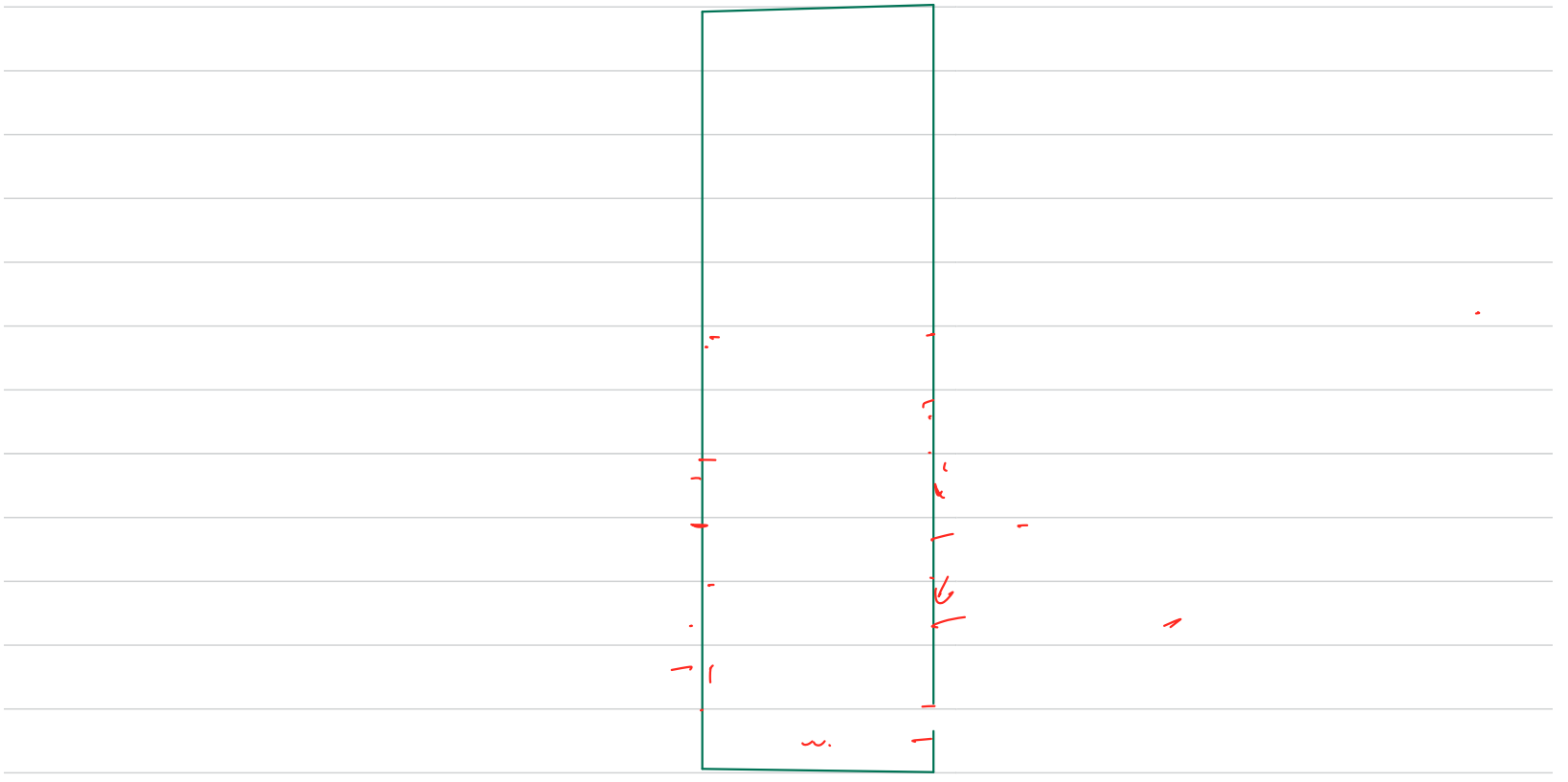
$\rightarrow f(n-1)$ ✓

$\rightarrow f(n-2)$ ✓

Self work $\rightarrow$ $f(n-1) + f(n-2) = f(n)$

Recursion tree





Q. There are n friends who want to party after covid-19 lockdown

What they decided is one person can either go alone or can go in a pair in the party

In how many different ways n friends can go to party.

Ex A B C

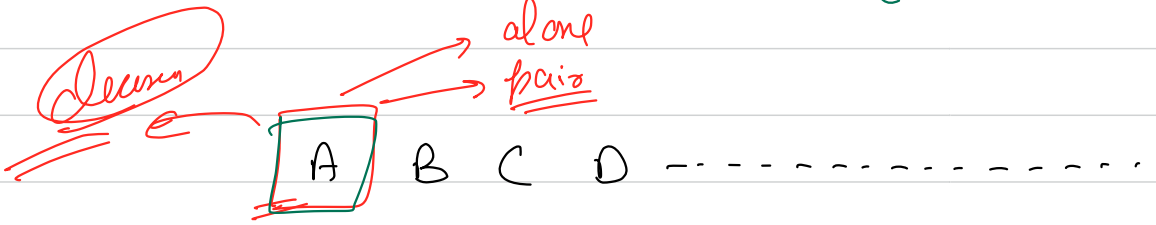
→ 4

→ (A) (B) (C) → 1 way
(A B) (C) → 1 way
(A C) (B) → 1 way
(B C) (A) → 1 way

Base Case → $n=1 \rightarrow 1$

$n=2 \rightarrow$ 

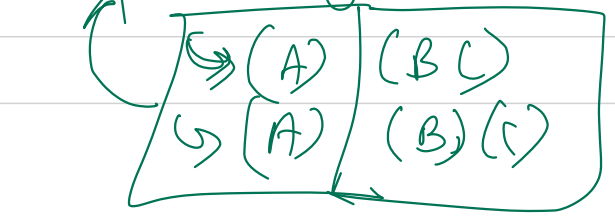
(A) (B) — 1
(AB) — 1
2



Define a func $f(n)$ → returns the no. of ways n friends can go to a party

Let's say A wishes to go alone → $f(n-1)$

A, B, C



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

if A deads
to go alone

alone

A, B, C, D

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

$f(n-1)$

⇒ 4

A B CD → #0

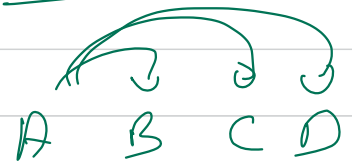
go in a pair

I will go in pair

A B C D ---------- (n-1) cadet

How may pairs A can make

$f(n) = (n-1) \times f(n-2)$



(AB) (C) (D)

(AB) (CD)

6 = 2

(AC) (D) (B)

(AC) (DB)

2

(AD) (B) (C)

(AD) (BC)

2 → 6

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

$n = 1 \rightarrow (1)$

$n = 2 \rightarrow (2)$

I will go in a pair ⇐

$(n-1)$

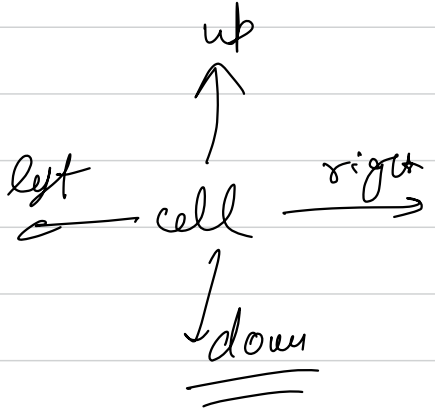
$n=4$

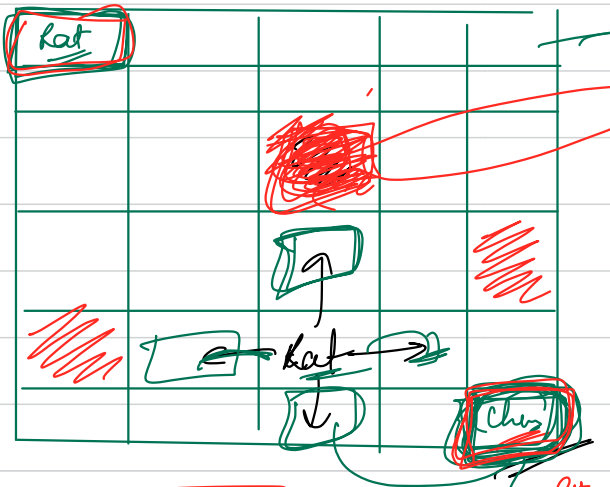
$$f(n-2) + f(n-2) + f(n-2)$$

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

$$3 \times f(n-2)$$

Backtracking → It is an algorithm that tries to find a solution to a given parameter. It builds the subproblems recursively & abandons those which can't fulfill the condition.





return 0

state → obtained

i, j → coordinate of any cell

1

Base Case

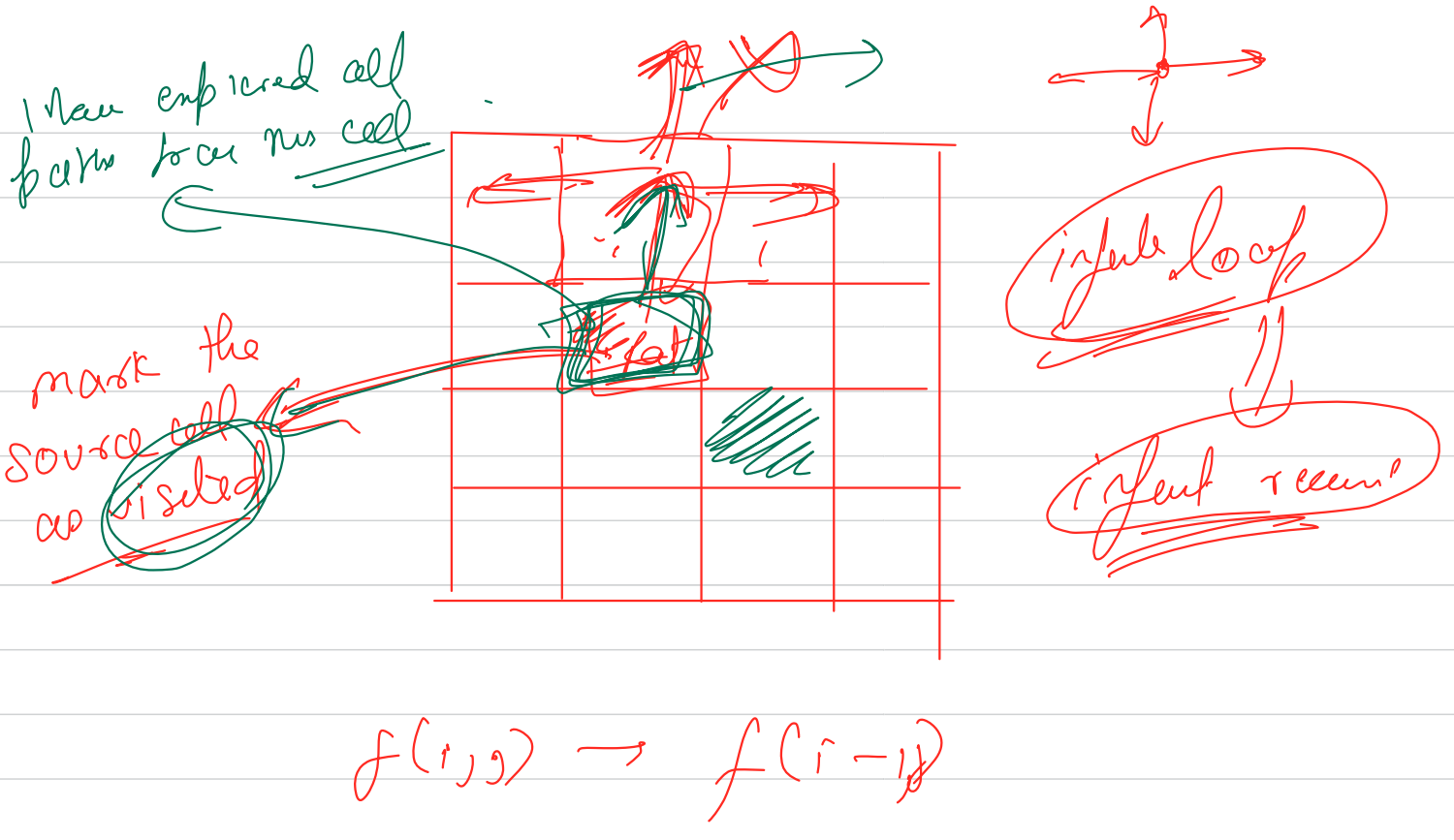
→ Recursive hypothesis

Self work

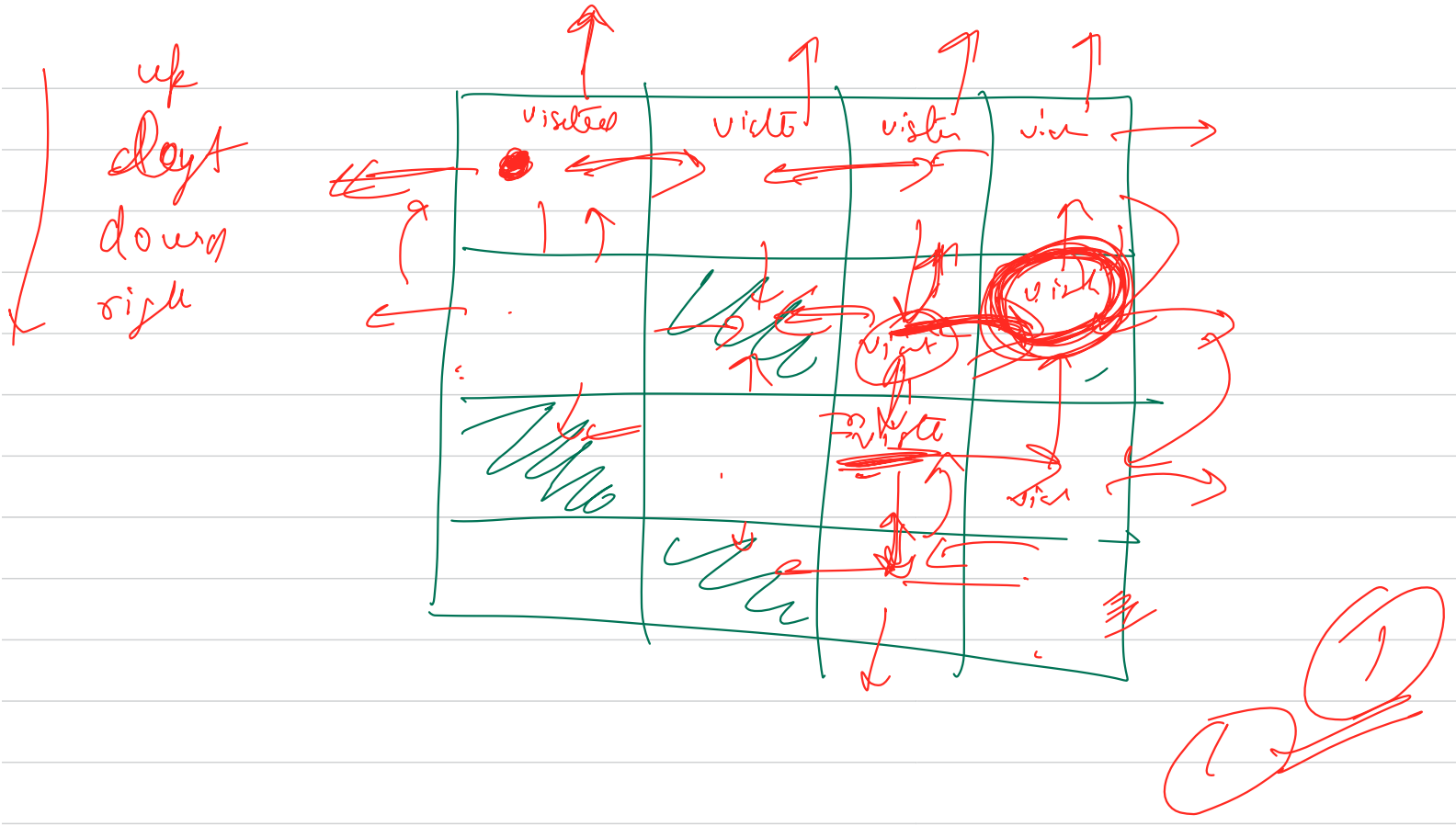
$f(i,j)$ = no. of ways to reach $i,j \to n,n$

$= f(i, j+1)$ (right) $+ f(i, j-1)$ (left) $+ f(i-1, j)$ (up) $+ f(i+1, j)$ (down)

not blocked



$$f(i,j) \rightarrow f(i-1,j)$$


up
down
right

space complexity → max space your process took during runtime