Pre- Requisites →

* Basic fundamental knowledge of any one language. (C++, Java, Python)
* conditionals
* loops
* functions
* A good logical aptitude will be reqd.

Syllabus

→ Introduction To Recursion

→ Dry Run

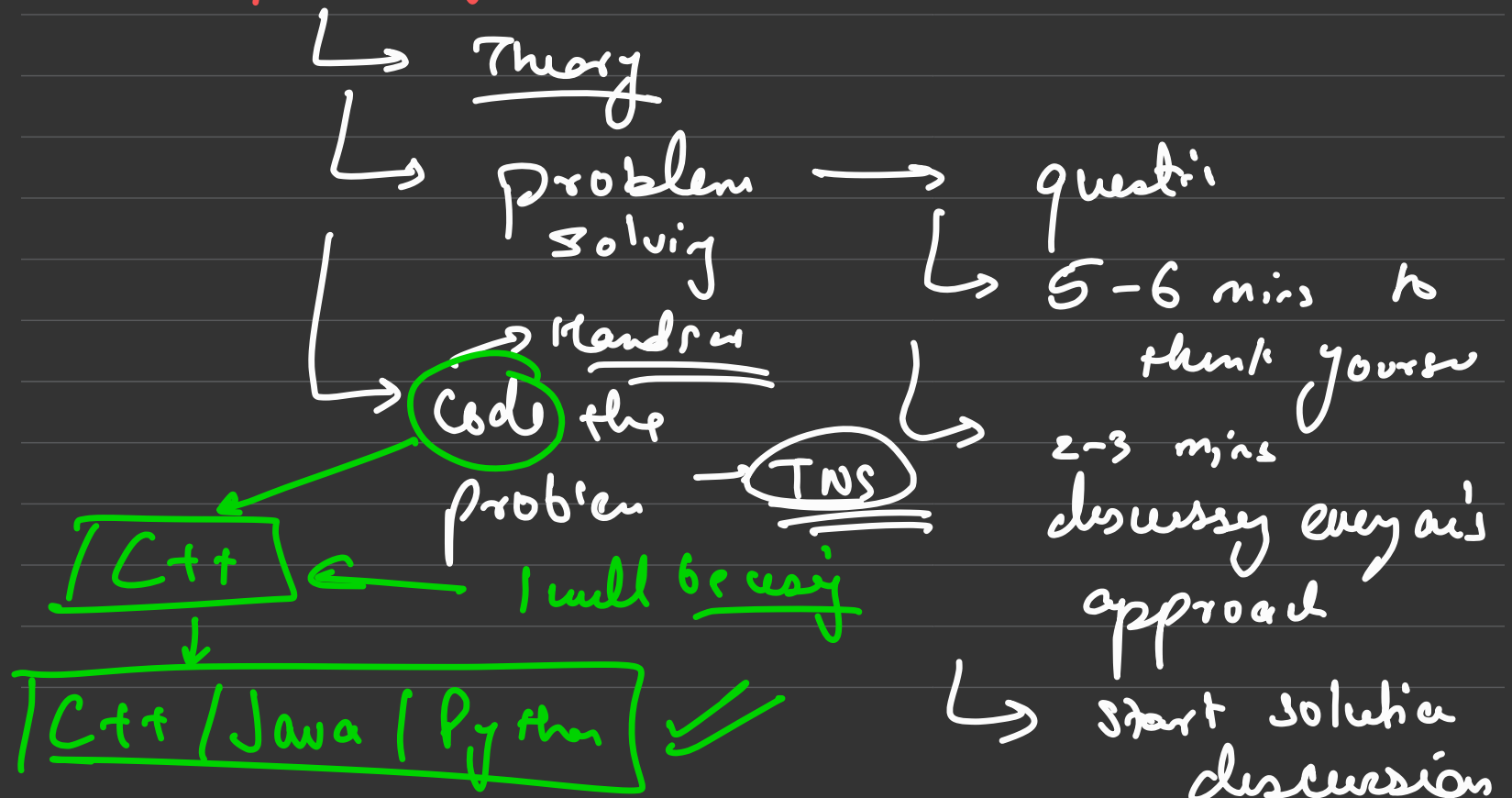
→ Recursion on arrays and strings

→ Introduction To Backtracking

→ Rat in a Maze, N Queen, N Knights, Knight Tour

→ Cp level ← hand problem solving

↳ flow of the class



Recursion

A child couldn't sleep, so her mother told a story about a little frog,
who couldn't sleep, so the frog's mother told a story about a little bear,
who couldn't sleep, so the bear's mother told a story about a little weasel
...who fell asleep.
...and the little bear fell asleep;
...and the little frog fell asleep;
...and the child fell asleep.

Sum of first n natural no.

Base Case → n = 1 → Sum = 1

Smallest problem for which we know the ans.

What is Recursion ??

↳ It is a programming technique using which we are able to solve bigger problems, by solving their smaller problems and then aggregating the soln. In this, we use a function which calls itself inside making it a recursive chain of function calls.

Recursive funcs are those which rely on themselves, to calc part of an answer.

- Base Case
- Self work
- Recursive Intuition / assumption

PMI

Principal Of Mathematical Induction → (PMI)

Q: Proove that Sum of first N natural no's is $\frac{N \times (N+1)}{2}$ ⟷

Base Case → N = 1 Sum = 1 ✓ $\frac{1 \times (1+1)}{2} = 1$

Assumption → assume the formula works for N = k. → $\text{sum}(k) = \frac{k(k+1)}{2}$ ✓ assum to be true

Selfwork → Prove that formula works for

$N = k+1$

$$\text{Sum}(k+1) = \text{Sum}(k) + (k+1)$$

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

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

$$= \frac{(k+1)(k+2)}{2} \quad \text{✓ IH}$$

Q. Calculate factorial of a no. N.

N = 5 ans → 120

Recursively

```
ans = 1
for(i = 2; i <= N; i++)
    ans = ans * i;
```

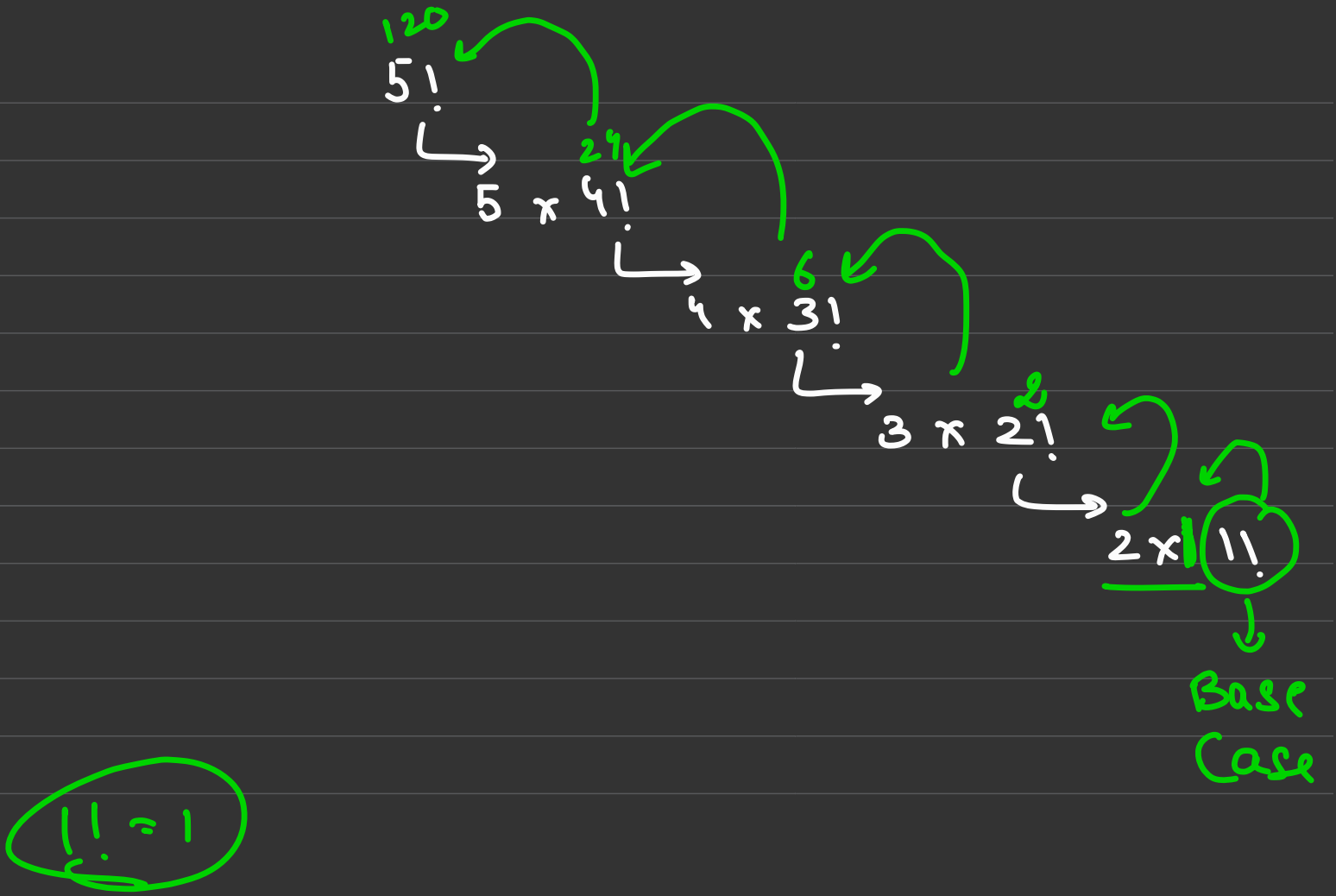
Base Case ⟶ N=1 N! = 1 ✓

Recursive Assumption → (N-1)!

Self work. ⟶ N x (N-1)!

$N! = N \times (N-1) \times (N-2) \cdots (2) \times (1)$

$N! = N \times (N-1)!$


120
5!
5 x 4!
24
4 x 3!
6
3 x 2!
2
2 x 1!
1!
Base Case
1! = 1

$f_n = n \times f_{n-1}$ → Recurrence Relation

returns $n!$

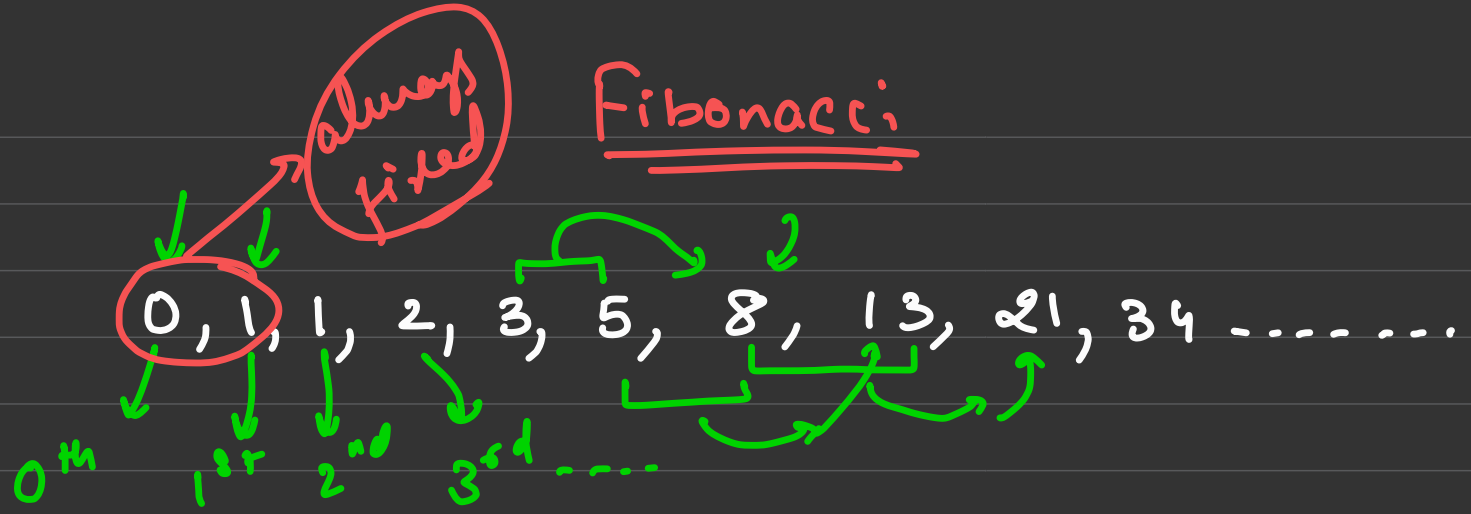
$f_1 = 1$ → Base Case (as $n \to 1$)

f_{n-1} → Recursive assumption

$f_{n-1} \times n$ → Self work.

Fibonacci

always fixed



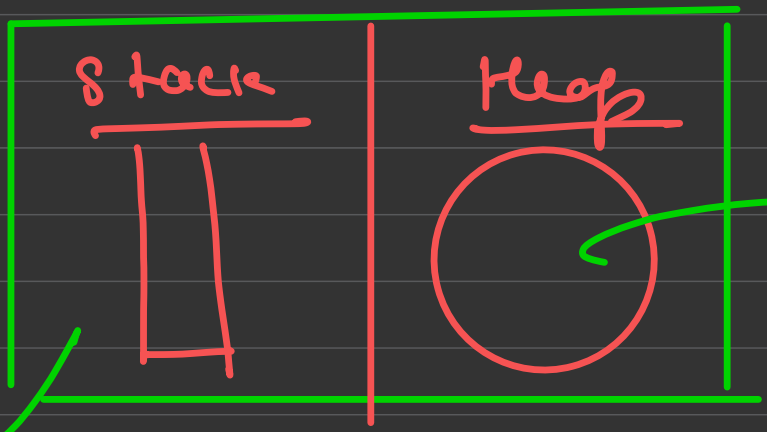
Base Case → N=0 → 0
N=1 → 1

Recursive assumption → f_{n-1}, f_{n-2} (2, 1)

Self work → $f_n = f_{n-1} + f_{n-2}$

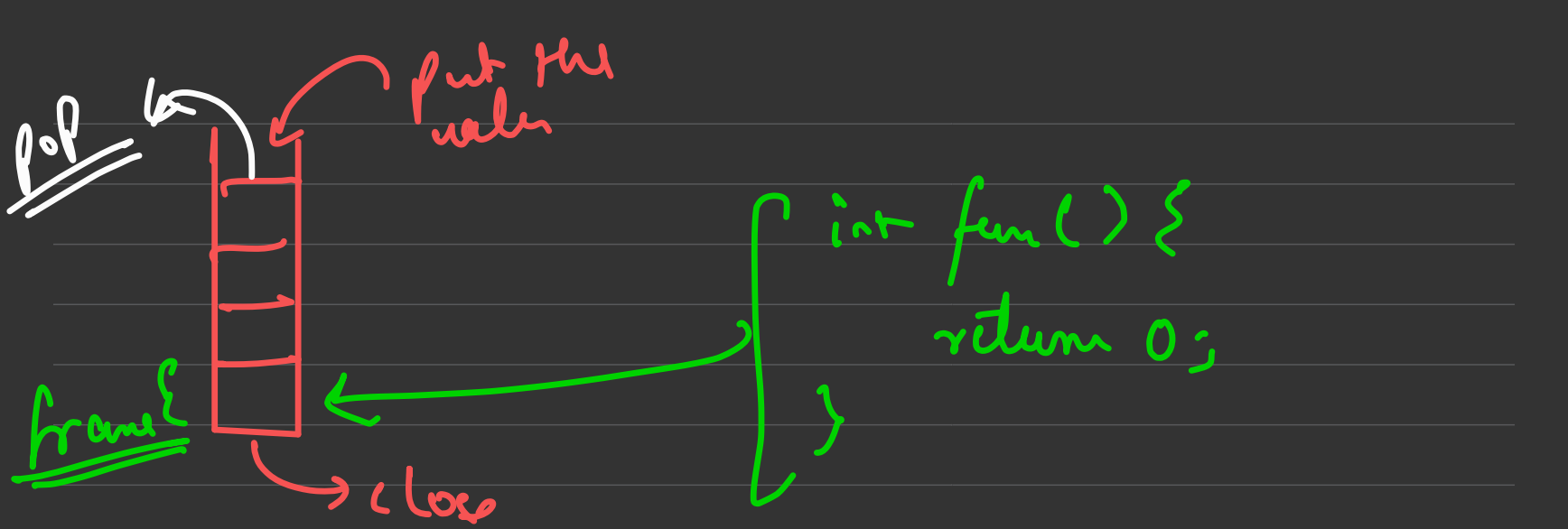
what happens behind the scenes.

How memory works ?:

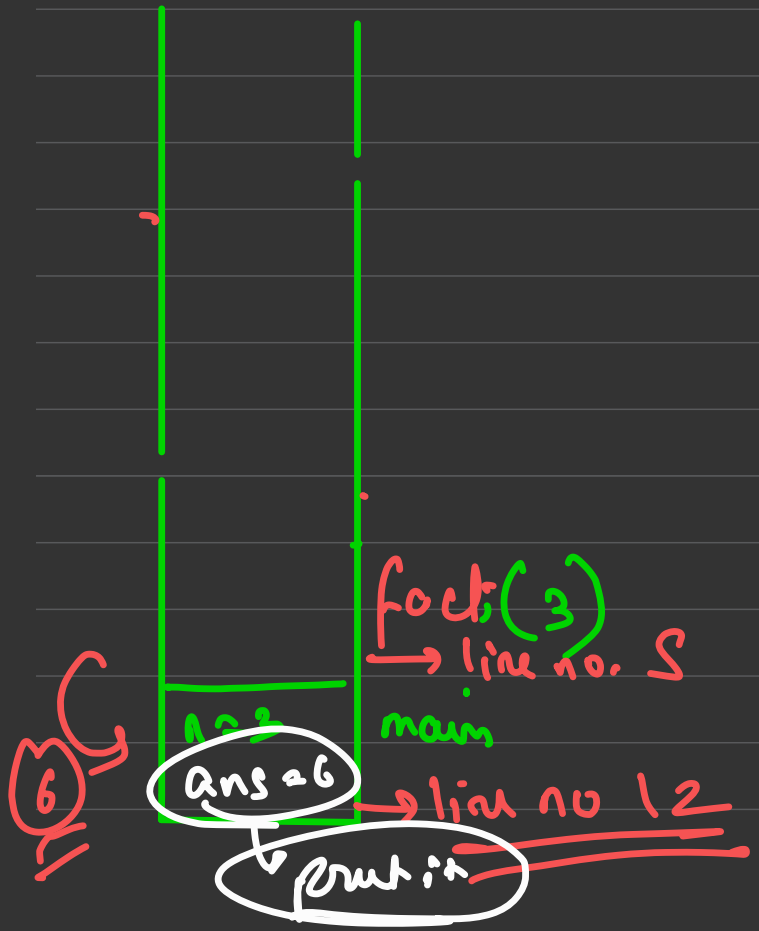


By pool of memory with almost no structure

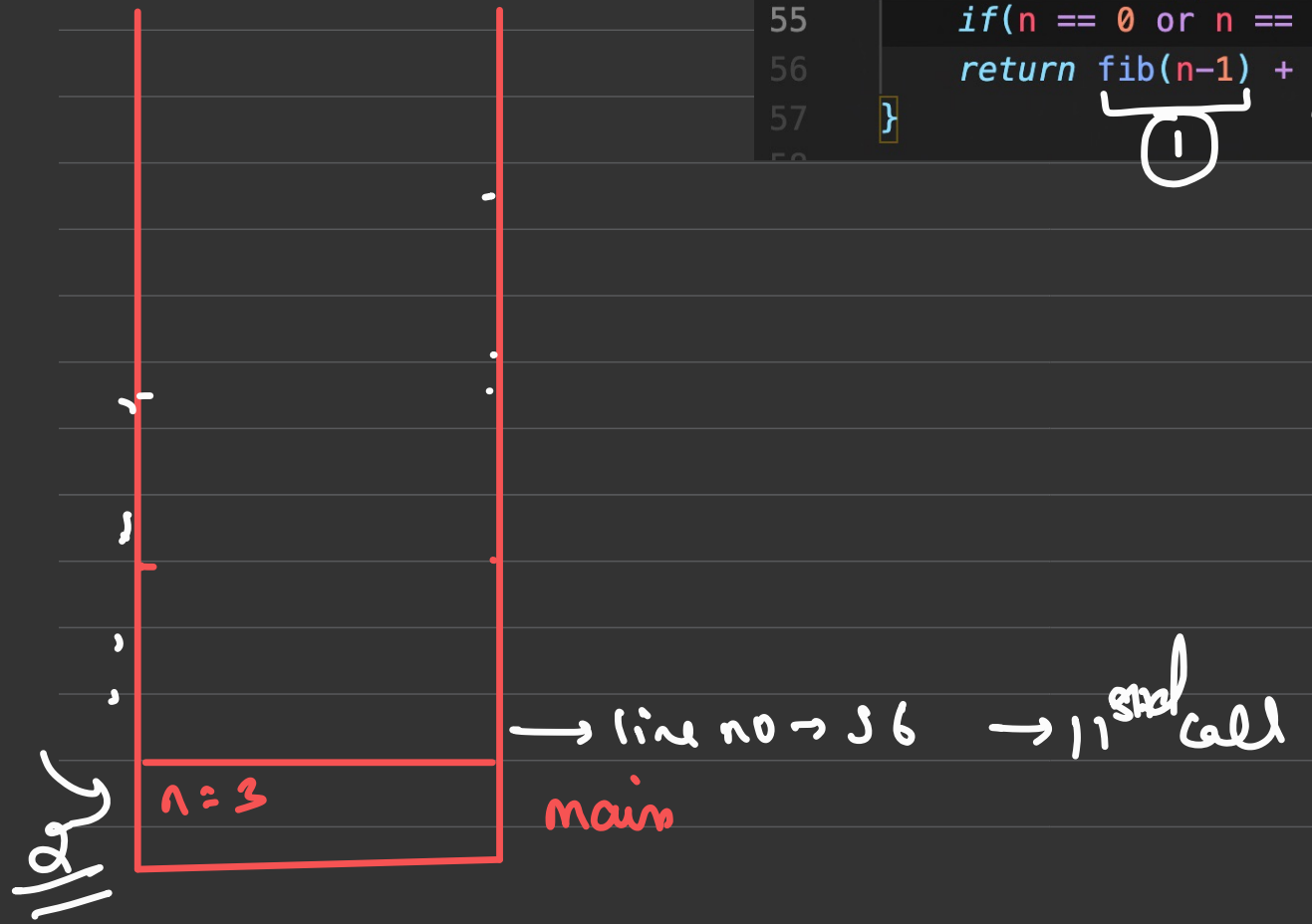
linear data structure.

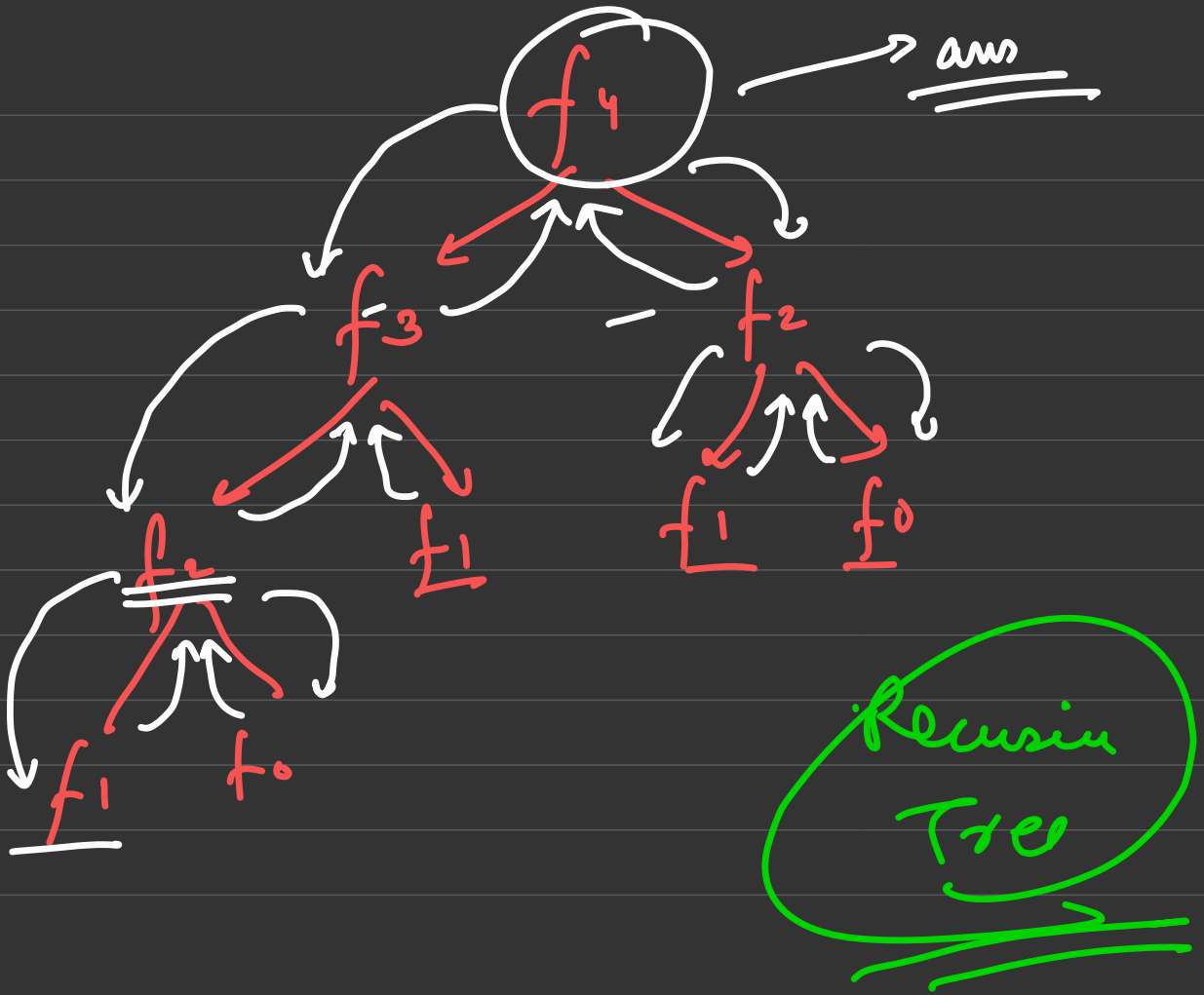

Pop
Put the value
frame
close
int fun() {
return 0;
}

```
3   int fact(int n) {
4       if(n == 1) return 1;
5       return n * fact(n-1);
6   }
```



```
54  int fib(int n) {
55      if(n == 0 or n == 1) return n;
56      return fib(n-1) + fib(n-2);
57  }
```




f4
ans
f3
f2
f2
f1
f0
f1
f0
Recursion Tree

Q: Given a no. n, print all the natural no. upto N, in increasing order

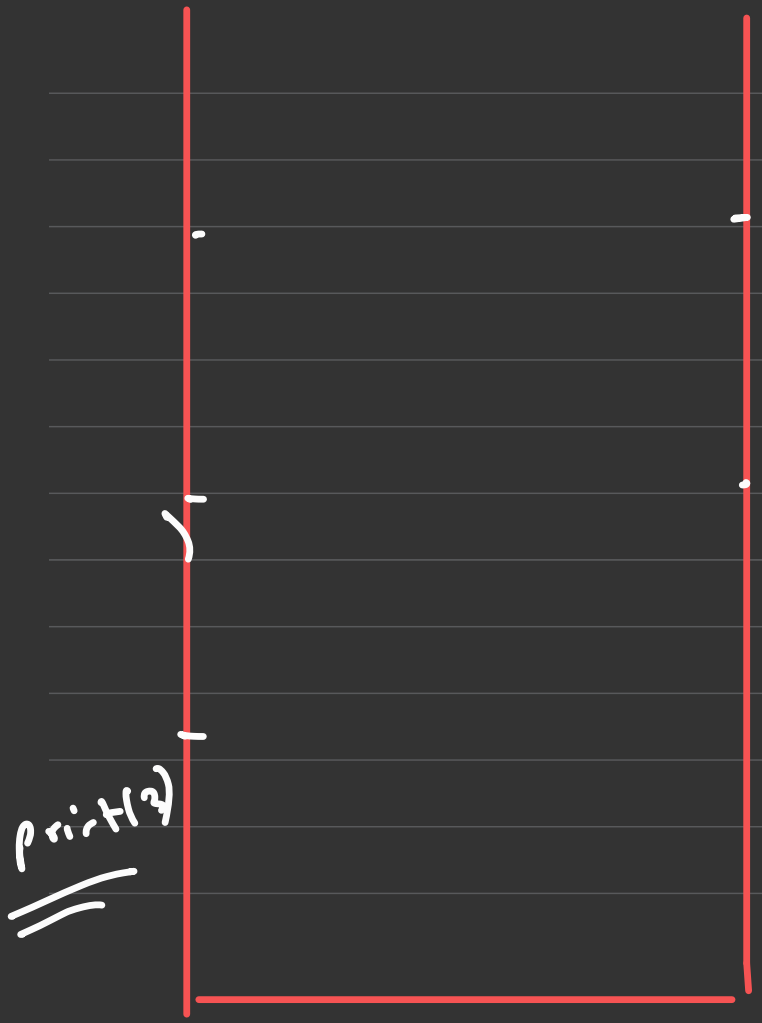
n = 5,

1
2
3
4
5

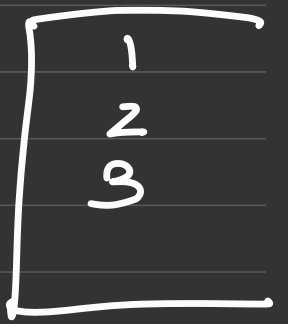
Base Case → (N-1)

Assumption → print (N-1)

Self work → cout << N




```
void print (N) {
  if (N==1) {
    cout<<1<<"\n";
    → return;
  print(N-1)
  cout<<N<<"\n";
}
```



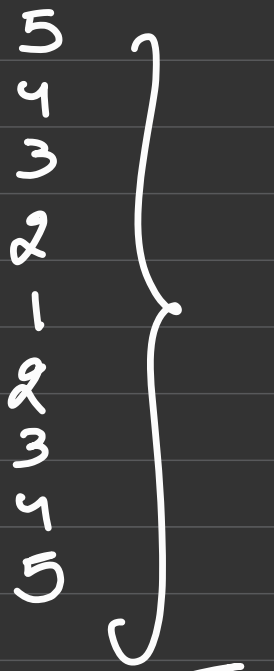
print(3)

Q → Given a value of N, print first N natural nos, in decreasing order.

N = 5 →

5
4
3
2
1

Q. N = 5



Q: Given an array, check whether it is sorted or not, recursively.

1, 2, 3, 5, 4 → false

Base Case → N=1 → only 1 elmt → sorted

Recursive assumption →