Recursion

1 — What is recursion?

2 — Steps for Recursion

3 — Interesting & Simple Problems.

Intro

→ Simple 2 mins video → Make observations.

Russian doll

main logic

1. smaller dolls = Subproblems
2. similar dolls = same fn ⇒ Assumption
3. end doll ⇒ sum(1) = 1
   ↳ Base condition.

Recursion

solving a problem using smaller instances of the same problem.

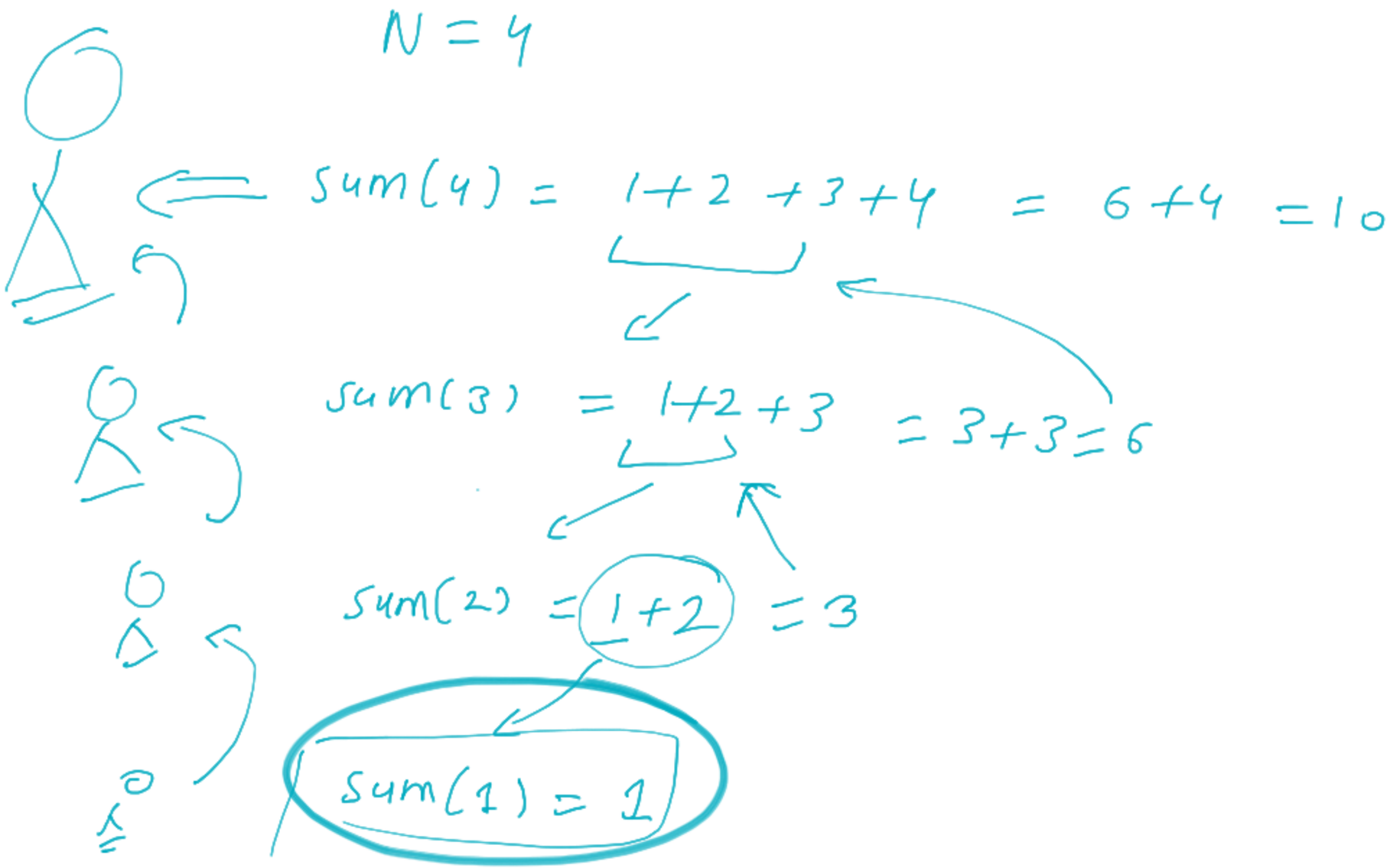
Q1) Sum of 1st N natural numbers.

N = 3 = 1 + 2 + 3

sum(3)

sum(2) = 1 + 2

sum(1) = 1


N = 4
sum(4) = 1+2+3+4 = 6+4 = 10
sum(3) = 1+2+3 = 3+3 = 6
sum(2) = 1+2 = 3
sum(1) = 1

$$sum(N) = \underbrace{1 + 2 + 3 + 4 + \_\_\_ (N-1)} + N$$

$$sum(N) = \underbrace{sum(N-1)} + N$$

subproblem.

1 # calculate sum of 1st N natural nums

```
def sum(N):
```

3
```
if (N==1):
    return 1
```

2 → return sum(N-1) + N

Steps for Recursion

1) Assumption

Decide what your function does/ what you want it to do.

2) Main Logic

Solve assumption using subproblems.

3) Base Condition

When does your recursive fn stops.

Q2 Factorial of given numbers.

N	0	1	2	3	4	5
facto	1	1	2	6	24	120

$4! = 4 * 3 * 2 * 1 = 24.$

$5! = 5 * (4 * 3 * 2 * 1)$

$5! = 5 * 4!$

(1) Assumption

return product of 1st N natural numbers.

```
def fact(N):
```

Main Logic

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

$N! = N * (N-1)!$

`fact(N)` = $N *$ `fact(N-1)`

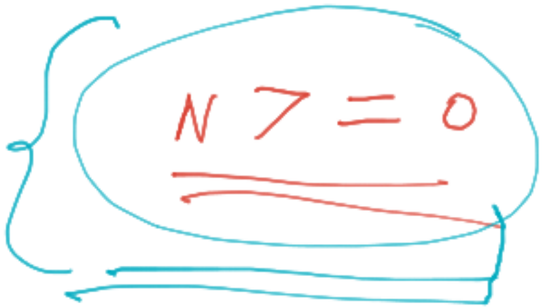
$N >= 0$

```
def fact(N):
    if N==1:
        return 1
    return N * fact(N-1)
```

fact(1) = 1

fact(2) = 2 * fact(1) = 2 * 1 = 2

fact(0) → 0 * fact(-1) → -1 * fact(-2) → - - -




```
def     fact (N):
1     if  N == 0
2           return 1
3     return  N * fact(N-1)
```

⇒

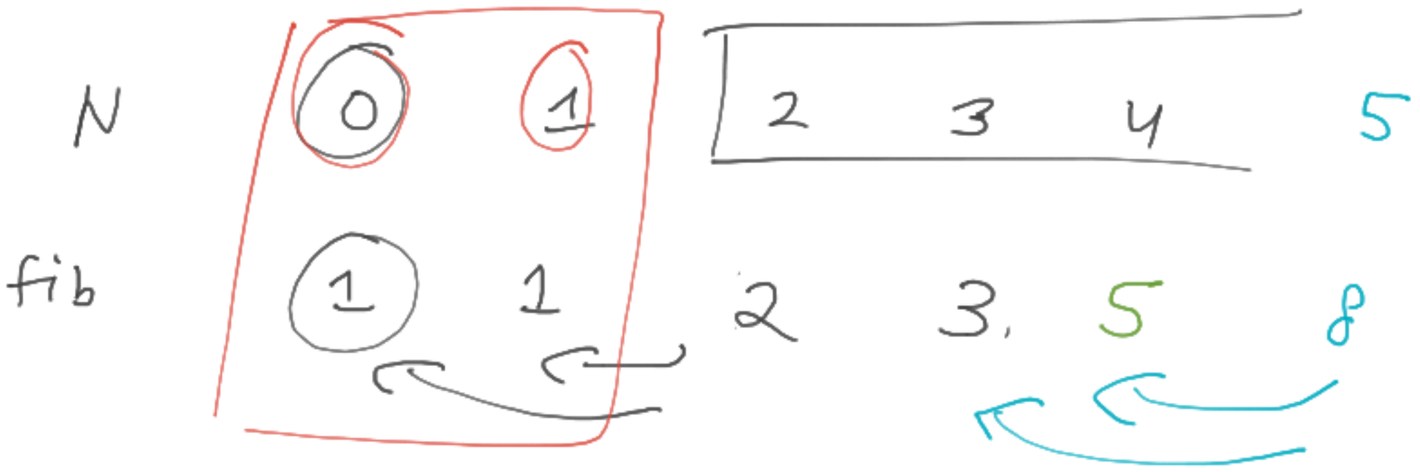
```
if  N == 0 || N == 1 :
        return 1
```

fact(1) = 1 * fact(0) = 1 * 1 = 1

fact(2) = 2 * fact(1) = 2 * 1 = 2

fact(0) = 1

Q3 Fibonacci



fib(5) = fib(4) + fib(3)

= 5 + 3 = 8.

N-1

N-2

Assumption: Computes fib of Nth number

N >= 0

```
def fib(N):
```

~~if N == 0:~~
~~return 1~~
~~return fib(N-1) + fib(N-2)~~

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

fib(1) → fib(0) + fib(-1)

-2 -3

Be cautious when choosing base condition

Assumption: Computes fib of Nth number

N >= 0

```
def fib(N):
1 X     if N == 1:
            return 1
2
3       return fib(N-1) + fib(N-2)
```

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

$fib(1) = 1$

$fib(0) = fib(-1) + fib(-2)$

$fib(2) = fib(1) + fib(0)$

Assumption: Computes fib of Nth number

N >= 0

```
def fib(N):
    if N<=0:
        return 1
    return fib(N-1) + fib(N-2)
```

fib(1) (1) = fib(0) + fib(-1)

= 2 1 + 1

fib(N) = fib(N-1) + fib(N-2)

Assumption: Computes fib of Nth number

N >= 0

```
def fib(N):
    if (N <= 1):
        return 1
    return fib(N-1) + fib(N-2)
```

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

$fib(1) = 1$

$fib(0) = 1$

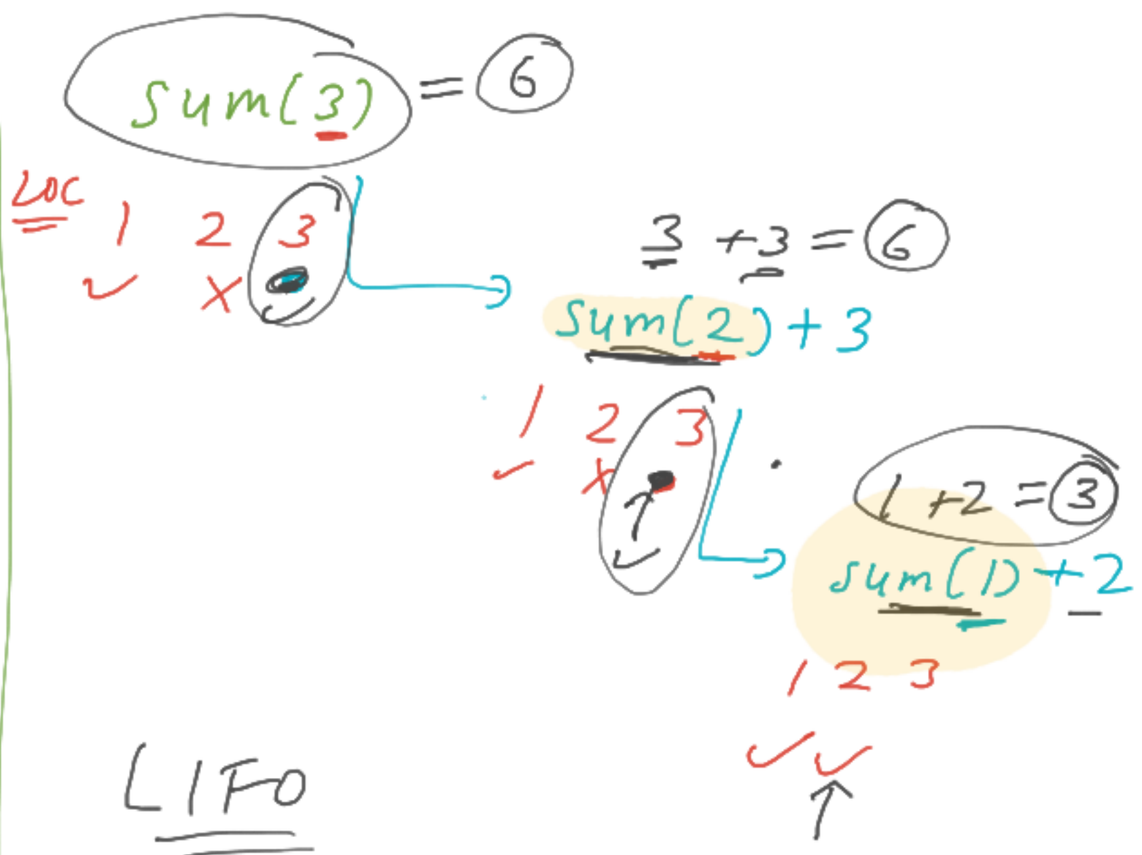
$fib(2) = 1 + 1 = 2$

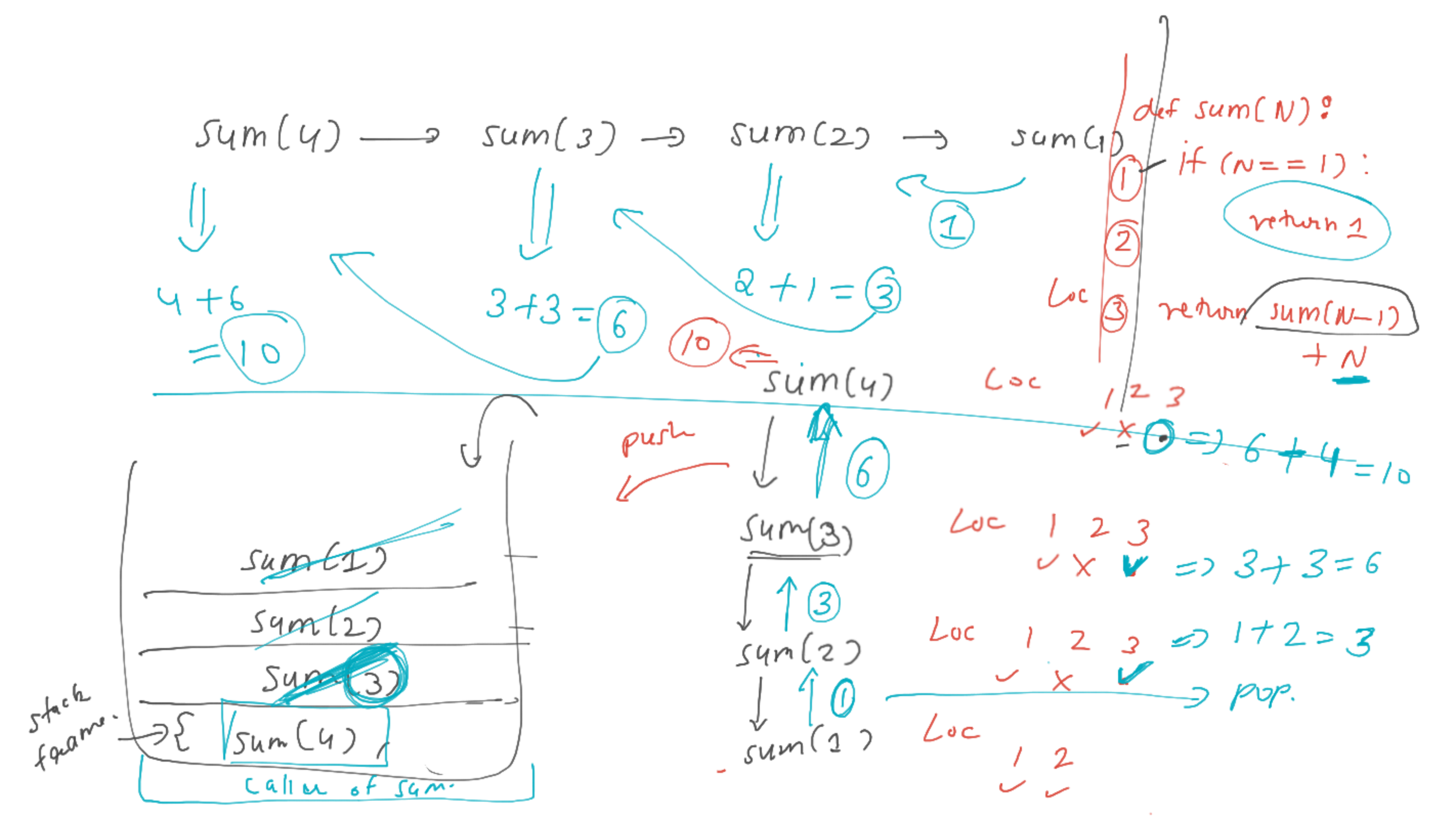
$fib(3) = 2 + 1 = 3$

Flow of sum function, How it works internally?

```
def sum(N):
1     if (N==1):
2         return 1
3     return sum(N-1)+N
```

```
print(sum(3))
```




sum(4) → sum(3) → sum(2) → sum(1)
4+6 = 10
3+3 = 6
2+1 = 3
1
10
6
3
def sum(N):
if (N==1):
return 1
return sum(N-1) + N
Loc 1 2 3
✓ X ✓ => 6+4=10
sum(4)
push
sum(3)
sum(2)
sum(1)
Loc 1 2 3
✓ X ✓ => 3+3=6
Loc 1 2 3
✓ X ✓ => 1+2=3
pop.
Loc 1 2
✓ ✓
sum(1)
sum(2)
sum(3)
sum(4)
stack frame
Callee of sum.

Very Imp:

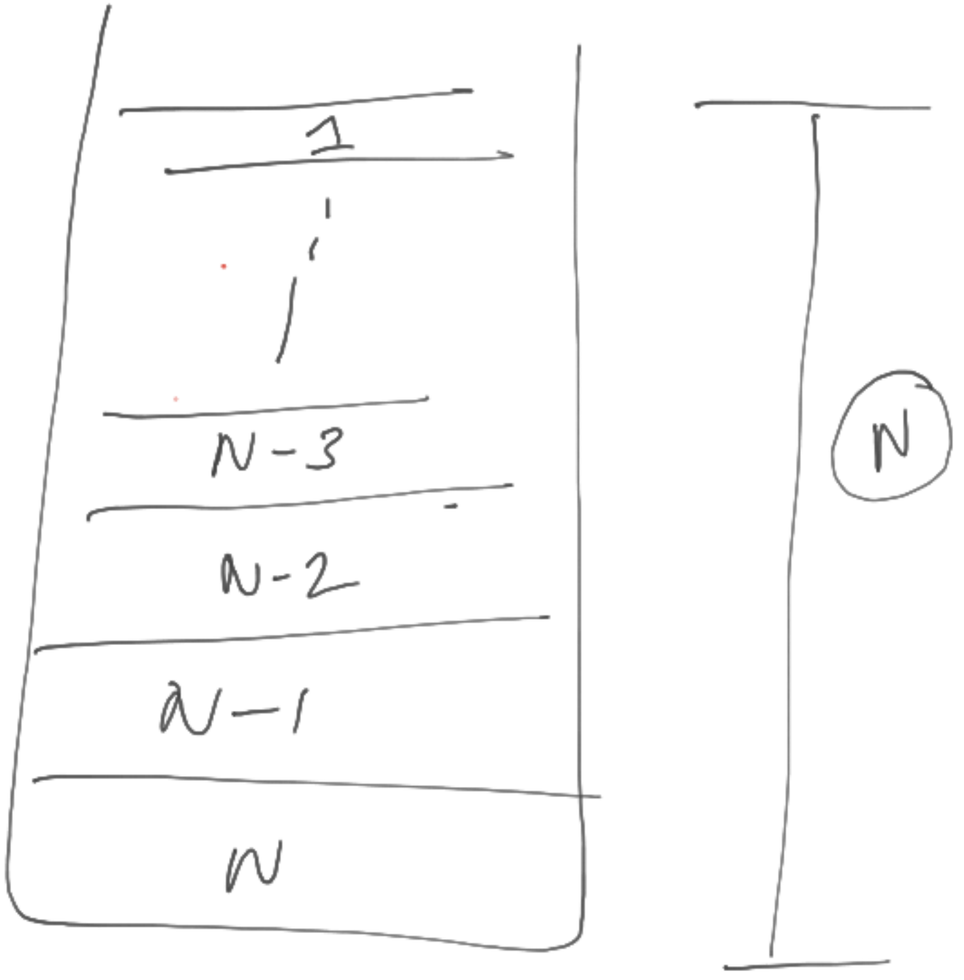
$SC = O(N)$

Using an explicit stack

⇓

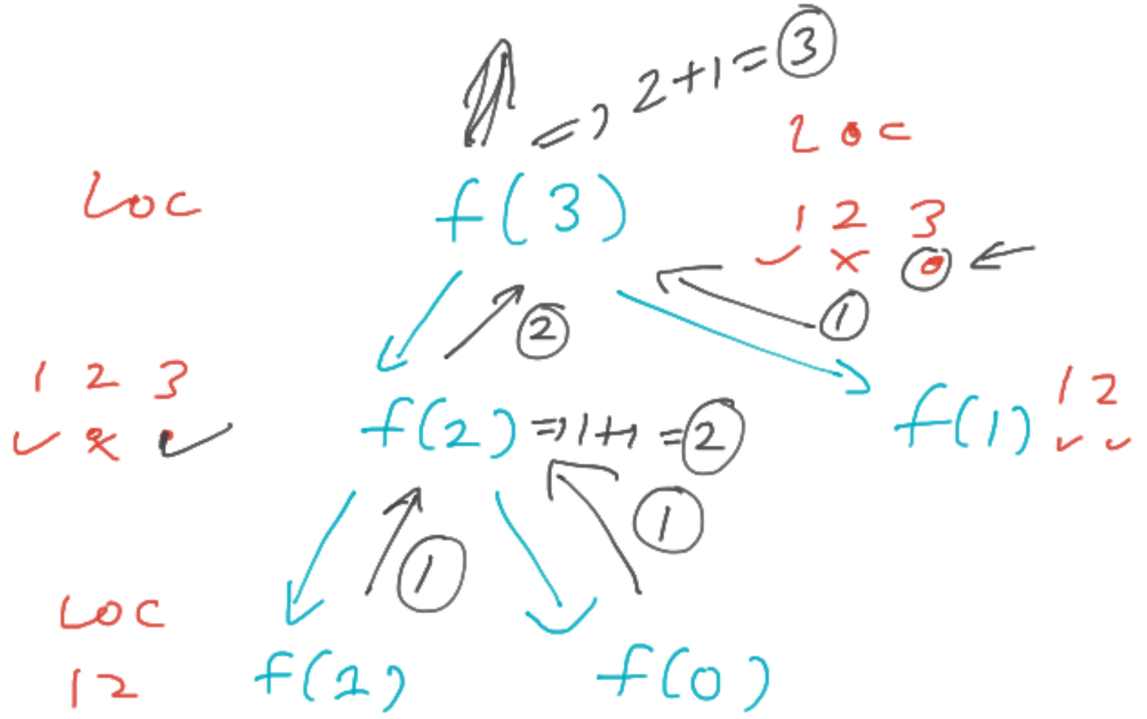
you can convert any recursive code to iterative code

Space complexity of a recursive function = height of the implicit stack



Fibonacci

```
def fib(N):
1       if N <= 1:
            return 1
2
3       return fib(N-1) + fib(N-2)
```

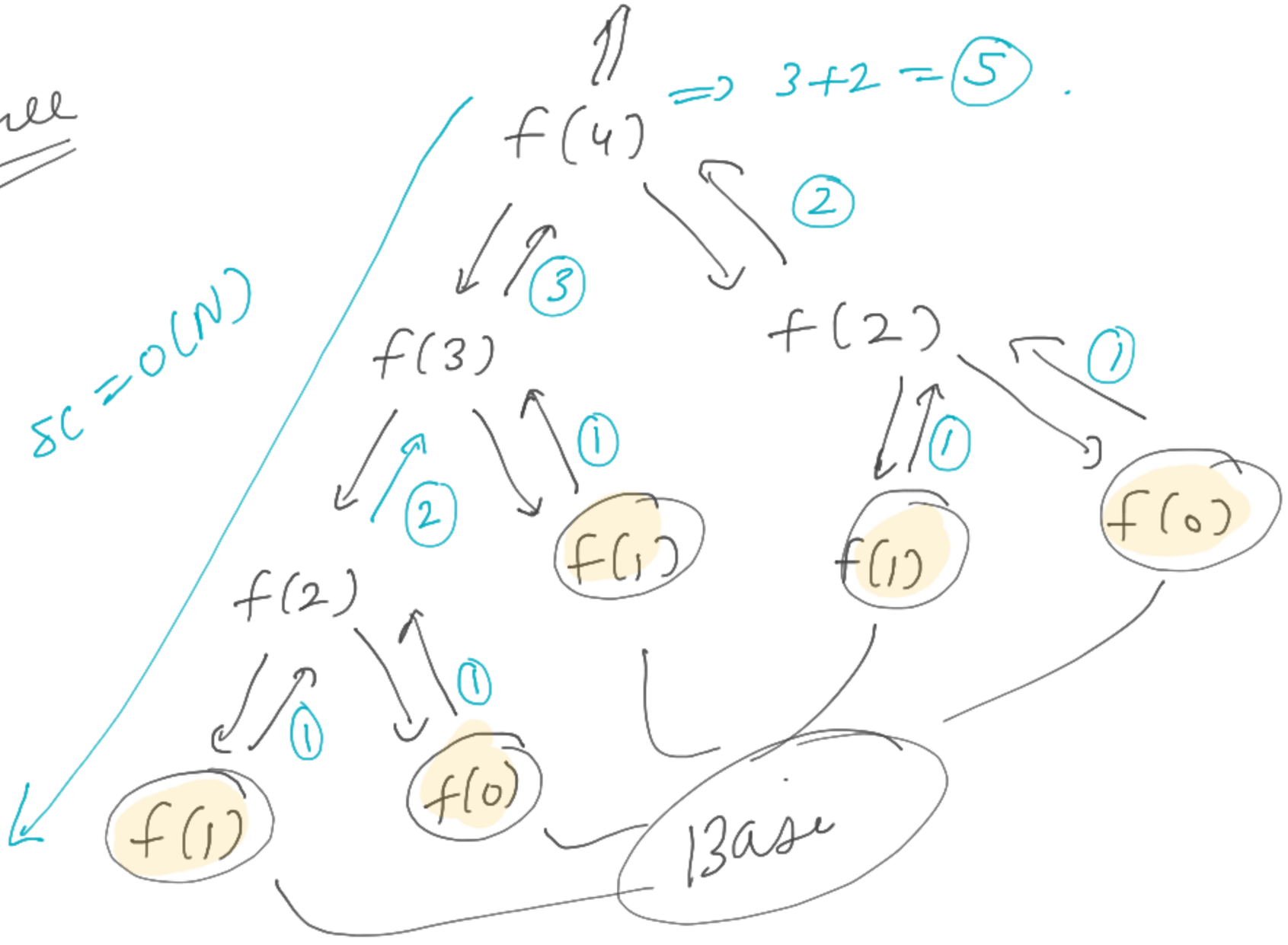


fib(3)	~~fib(2)~~	~~fib(1)~~	~~fib(1)~~

Max size = 3 ≤ N

SC = O(N)

Tree

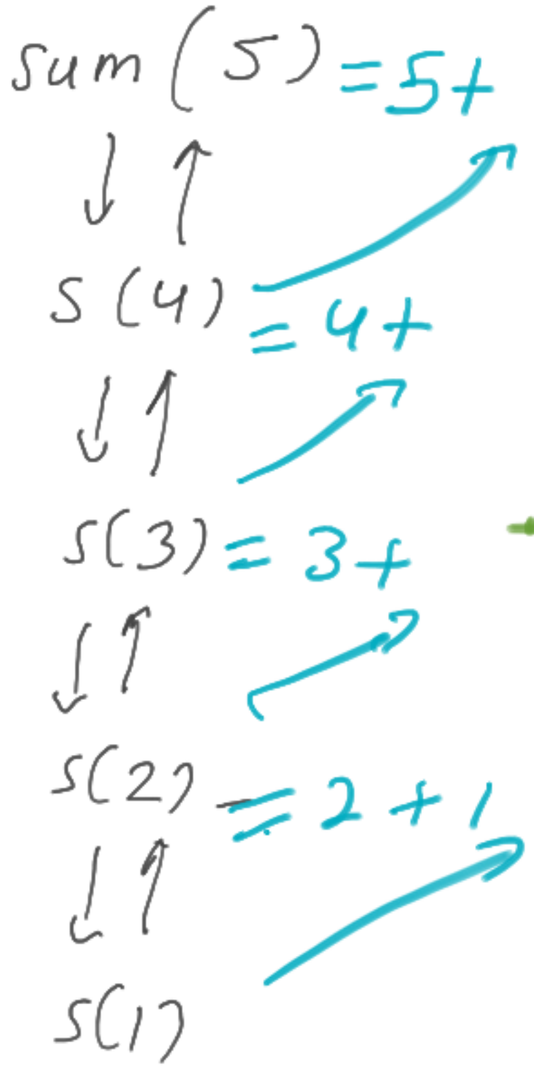


Time (Intuitively)

next →
- Recursive Relation Approach
- → Recursive Tree Diagram
- → Master's Theorem

if N == 1:
    return 1

return s(N-1) + N

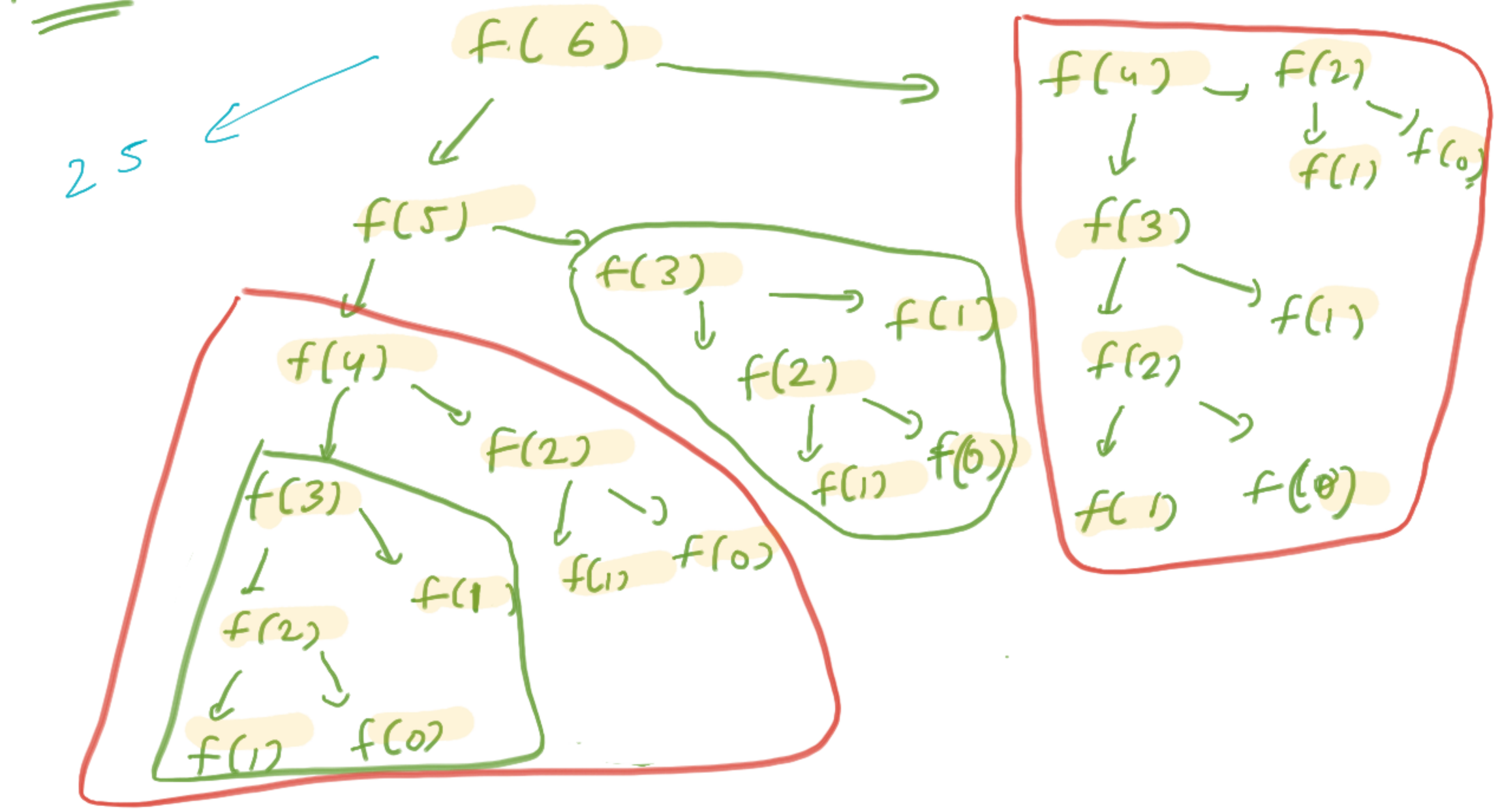


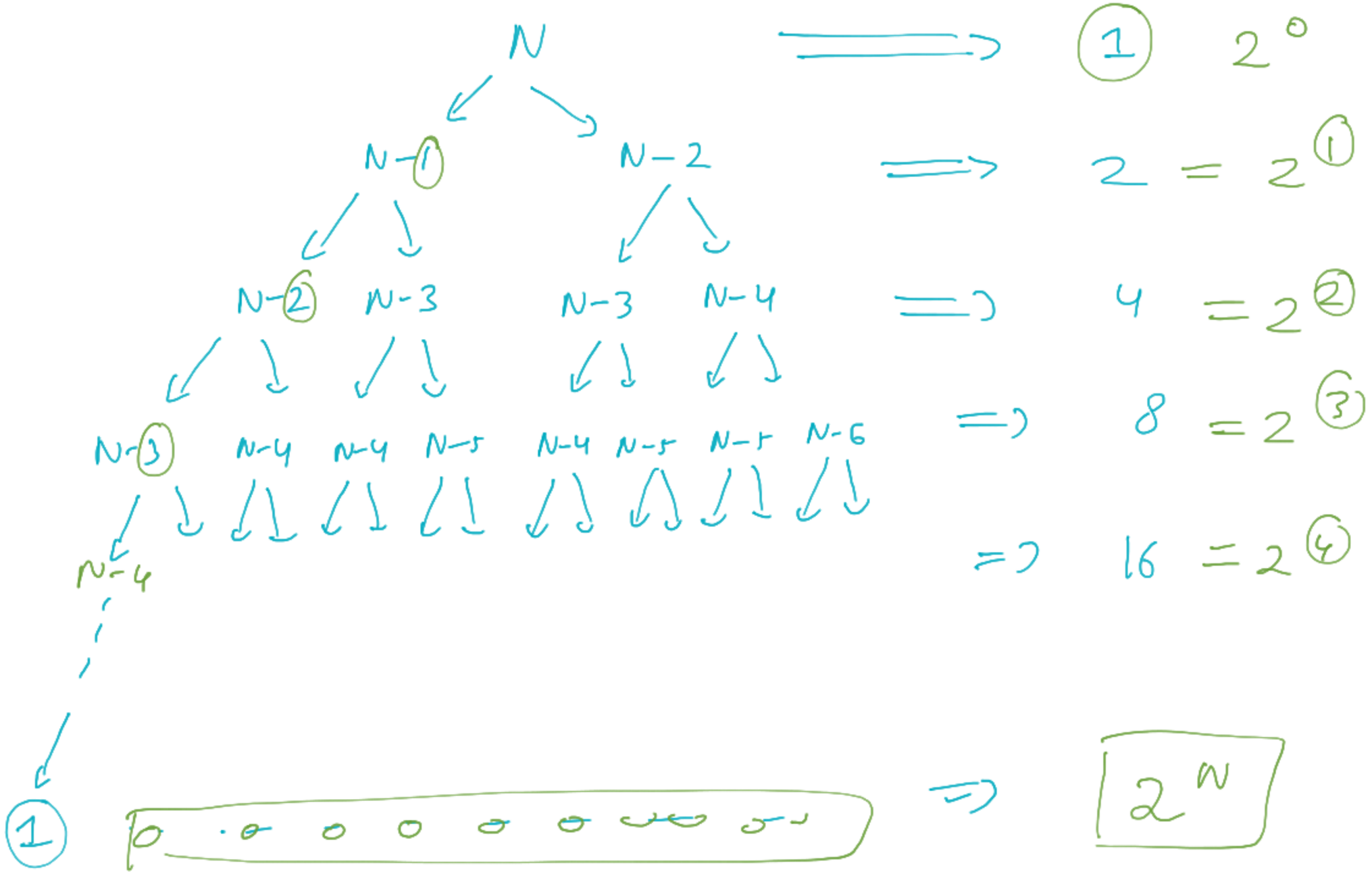
Total function calls * Time per fn call

Sum(N) ⇒ N * O(1)

= O(N)

Fib(6)


f(6)
25
f(5)
f(4)
f(3)
f(2)
f(1)
f(0)
f(3)
f(1)
f(2)
f(1)
f(0)
f(4)
f(2)
f(3)
f(1)
f(2)
f(1)
f(0)
f(3)
f(1)
f(2)
f(1)
f(0)
f(4)
f(2)
f(1)
f(0)
f(3)
f(1)
f(2)
f(1)
f(0)


N
N-1
N-2
N-2
N-3
N-3
N-4
N-3
N-4
N-4
N-5
N-4
N-5
N-6
N-4
1
1
2^0
2 = 2^1
4 = 2^2
8 = 2^3
16 = 2^4
2^N

No. of f^n calls

$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^N$ (N+1) terms

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

$\frac{a(r^N-1)}{r-1}$

$= 2 \cdot 2^N - 1$

TC. $O(2^N)$

(Q4) Given N numbers, print all the numbers in increasing order from 1 to N.

Only Use Recursion

N = 5

=) 1, 2, 3, 4, 5

Target

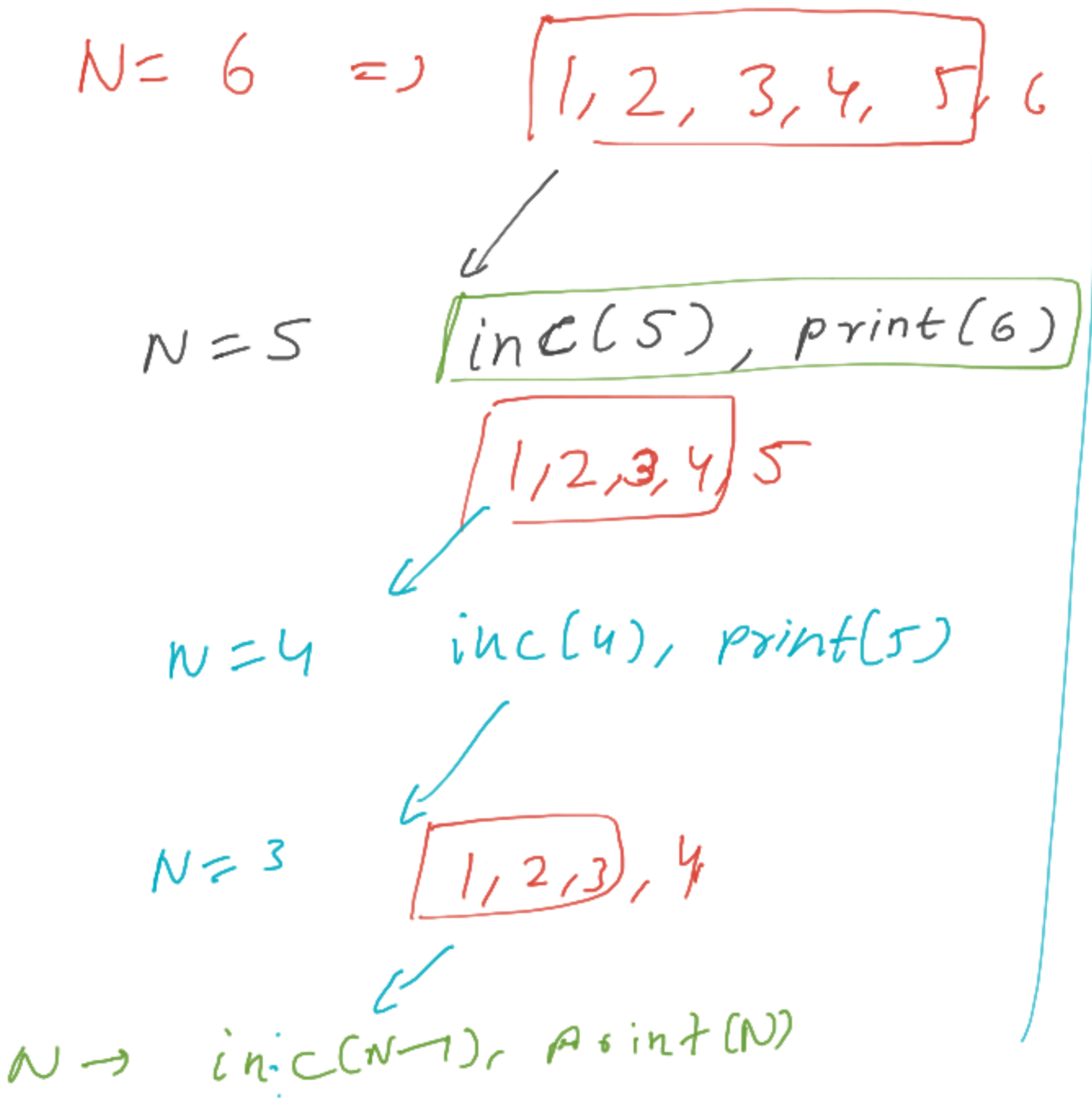
↳ confident in writing recursive codes.

Ass-umption ↑ $N \geq 1$

```
def inc(N):
    # main logic
```

Assumption: It will print from 1 to N in inc. order.

Main Logic



inc(1)

```
if (n == 1):
    print(1)
    return
```

inc(0)

```
if(n == 0):
    return.
```

```
def inc(N):
    if (N==1):
        print(1)
        return
    inc(N-1)
    print(N)
```

1

```
def inc(N):
    if N==0:
        return
    inc(N-1)
    print(N)
```

2

diff = Base Condition.

Todo:

Flow Diagram

Q5

{* Advance Problem}

Given an array, reverse it using recursion.

In-place.

I/P $l = [1, 2, 3, 4, 5, 6]$

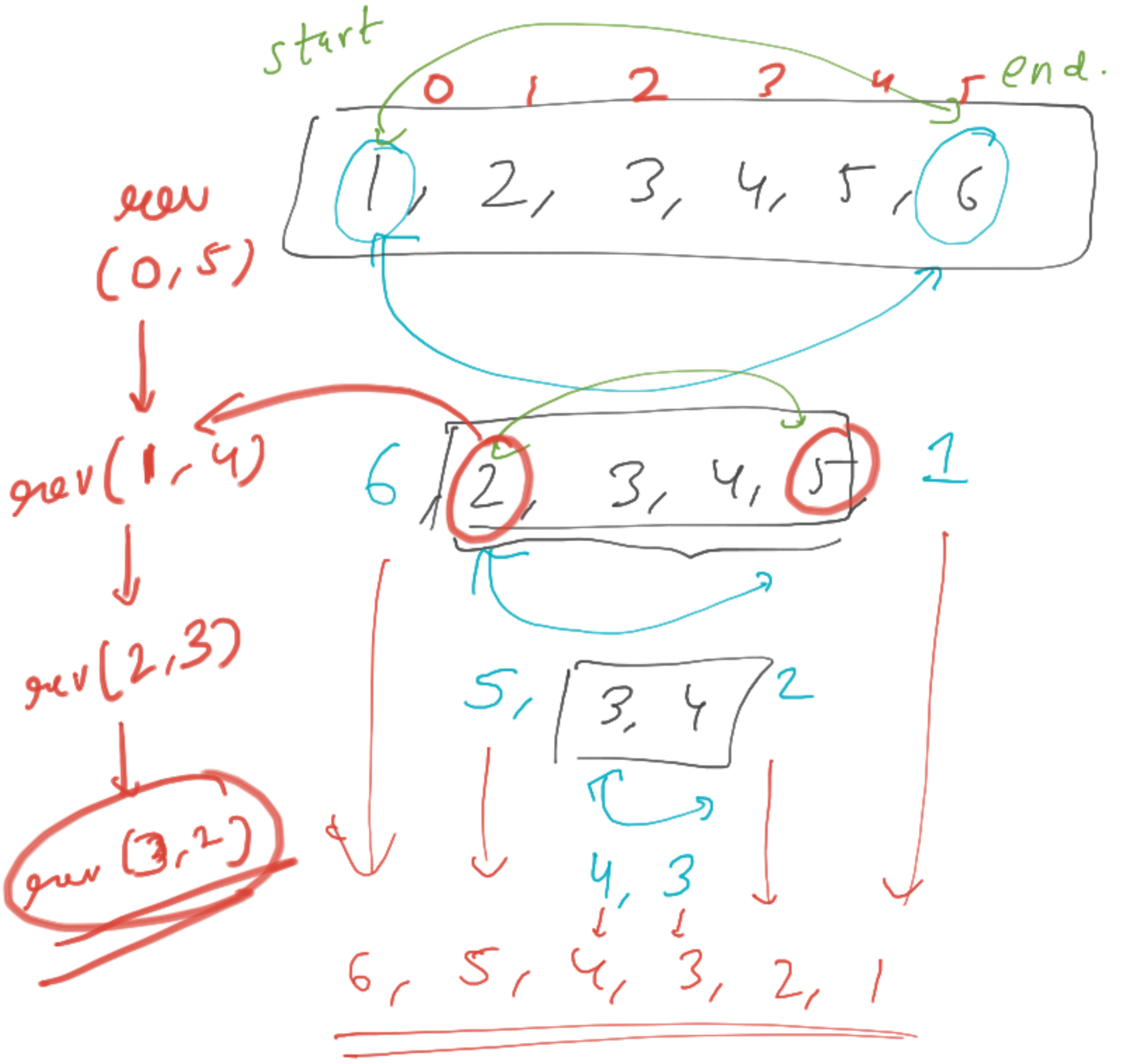
O/P: $l = [6, 5, 4, 3, 2, 1]$

X

```
def reverse(arr):
```

← Reverse the entire array

does the capture info of current size of smaller problem.



Every subproblem if we know the start & end!

Main logic

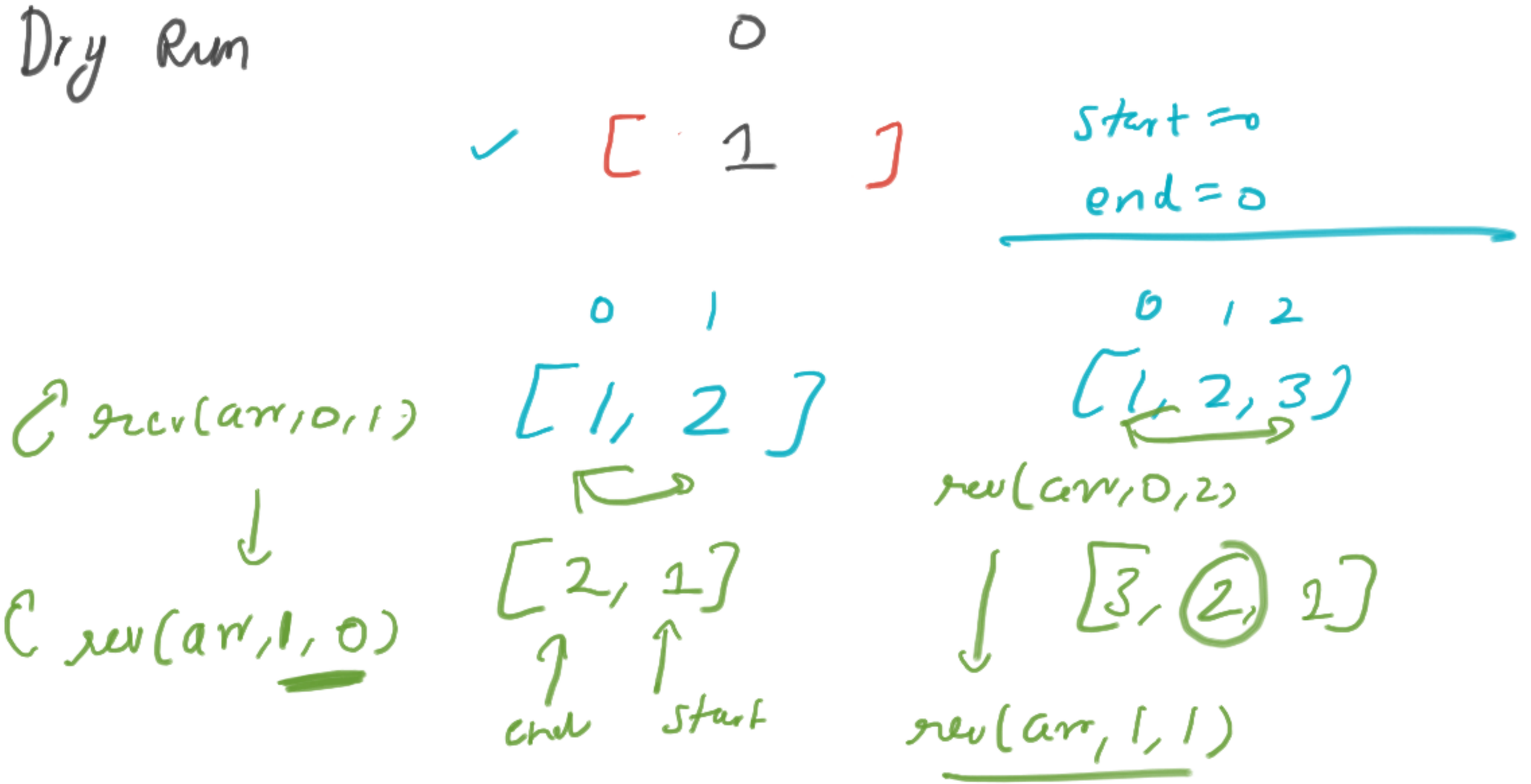
Assumption:

Reverse the array from start idx to end idx

```
def reverse(arr, start, end)
    if start >= end:
        return
    # reverse arr[start] & arr[end]
    arr[start], arr[end] = arr[end], arr[start]
    # reverse the part from (start+1) to (end-1)
    reverse(arr, start+1, end-1)
```

```
l = [1,2,3,4,5]
reverse(l, 0, len(l)-1)
```

Dry Run



To do: Flow diagram for N = 6/7

TC.

1, n

rev(0, N-1)

↓

rev(1, N-2)

↓

rev(2, N-3)

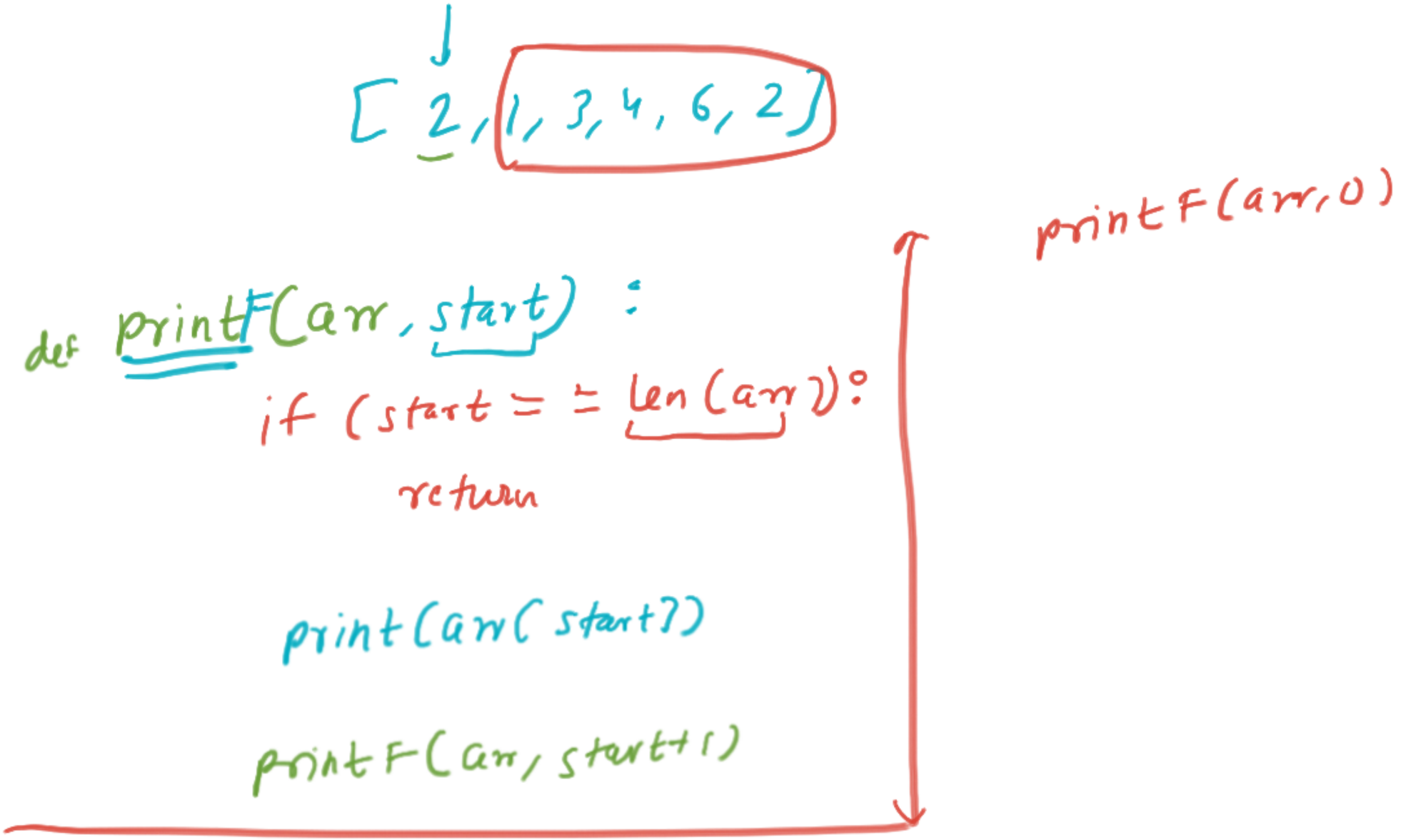
↓

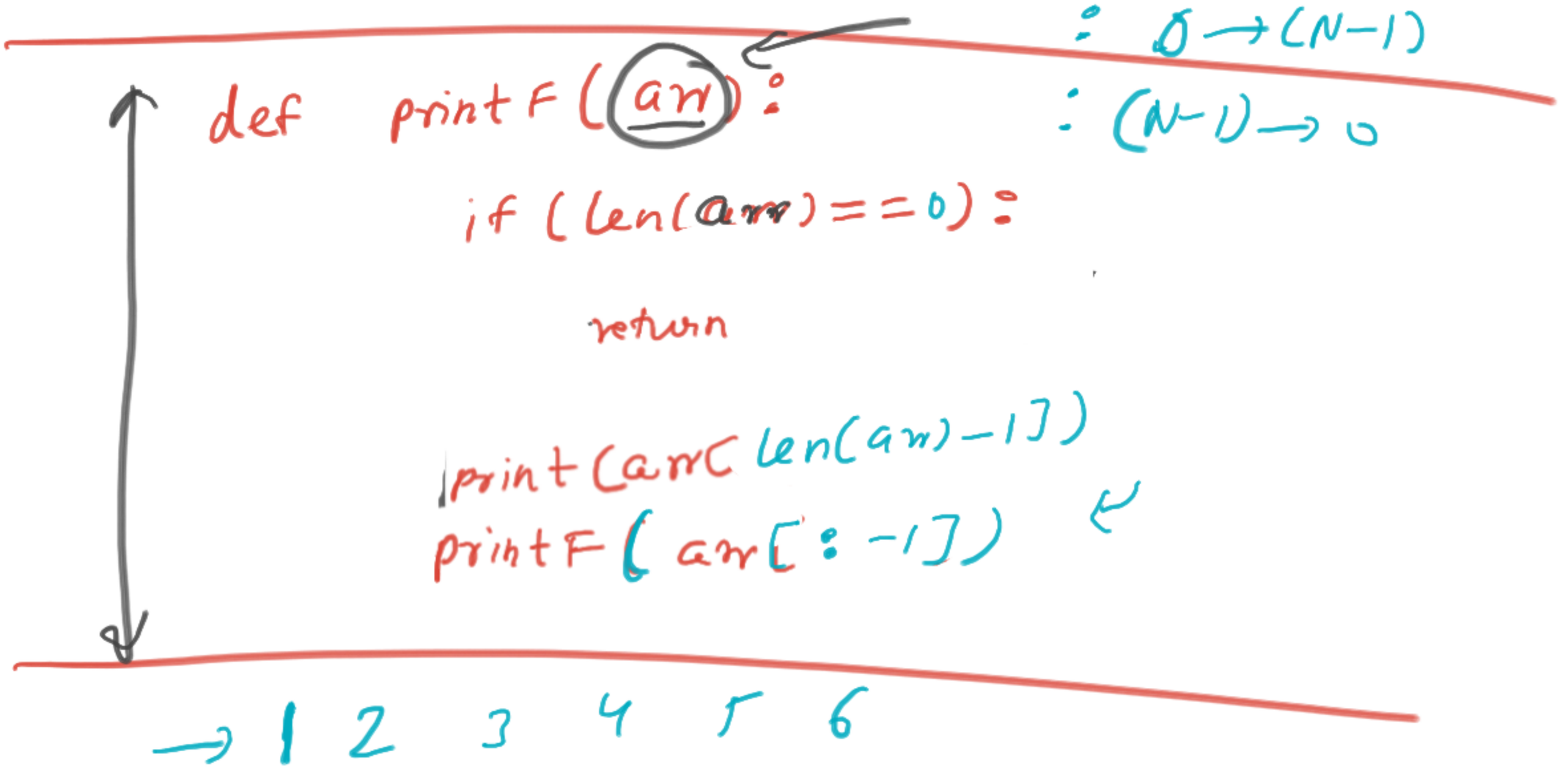
rev(N/2, N/2)

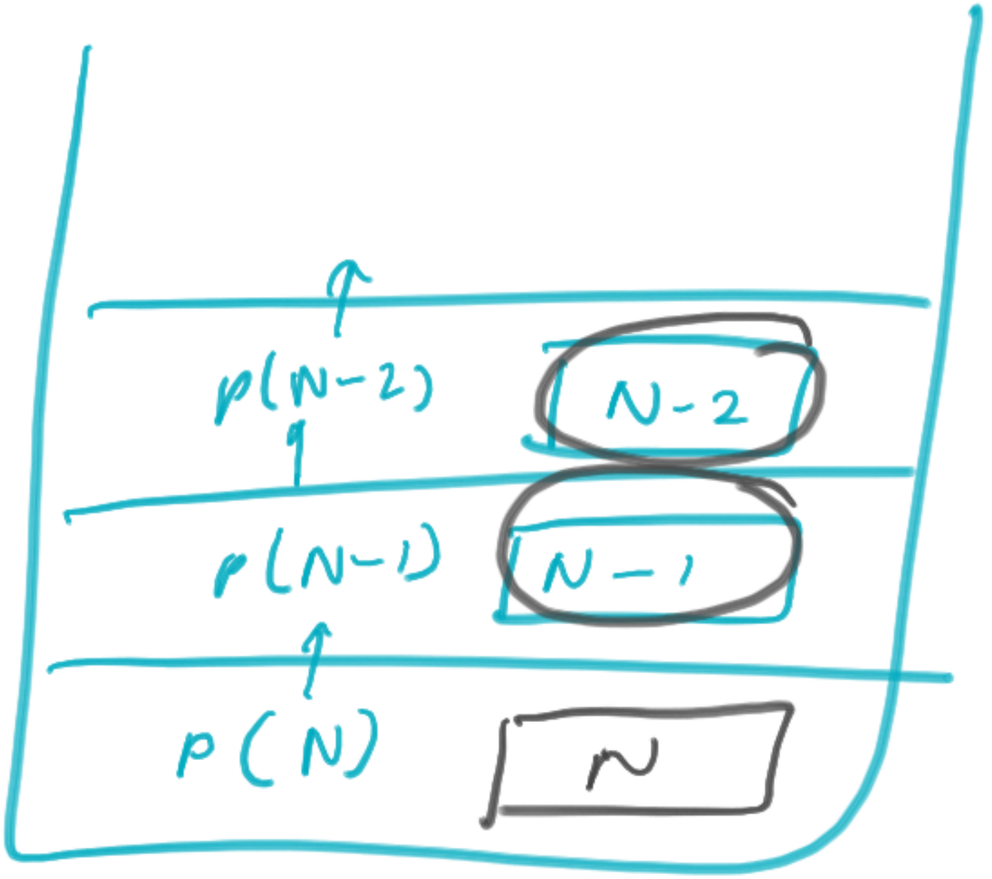
→ $\frac{N}{2} * O(1)$

$\Rightarrow TC = O(N)$

→ $S.C = O(N)$


[2, 1, 3, 4, 6, 2]
printF(arr, 0)
def printF(arr, start) :
if (start == len(arr)):
return
print(arr[start])
printF(arr, start+1)


: 0 → (N-1)
: (N-1) → 0
def printF(arr):
if (len(arr)==0):
return
printF(arr[len(arr)-1])
printF(arr[:-1])
→ 1 2 3 4 5 6



$$SC = N + (N-1) + (N-2) + \cdots + 1$$

$$= O(N^2)$$