Todays Content:

- Recursion?
- How to write a Recursive Code / Tracing
- TC/SC of Recursive Codes → Fridays Session

Why Recursion?

- Merge Sort / Quick Sort
- Binary Tree / BST / BBST / Segment Trees / Tries
- Dynamic Programming
- Backtracking
- Graphs

Recursion: function calling itself

↳ Solving a Problem, using Smaller Instance of Same Problem

↳ Sub Problem

↳ Sum(N) = 1 + 2 + 3 + .. N-1 + N

Sum(N) = Sum(N-1) + N

↑ ↑

Sum(4) = Sum(3) + 4 // Sum(3) is Sub Problem

How to write Recursive Code?

Assumption: Fix what your fun should do

Main logic: Solving assumption using Problem

Base Condition: Inputs, for which we want stop recursion

↳ at start of function

N >= 1

```
int sum(N) {
    if(n==1) { return 1}
    return [ Sum(N-1) + N]
}
```

Ass: Given N, calculate & return, Sum of 1^st N Natural Numbers

Sum(3) = 6

Sum(4) = 10

1+2+3 .. N-1 + N : Sum of N Natural Numbers

fact(3) = 3*2*1 = 6, fact(4) = 4*3*2*1 = 24, fact(5) = 120

N >= 1

```
int fact(N) {
    if(n==1) return 1;
    return fact(N-1) * N
}
```

Ass: Given N, calculate & return N!

fact(N) = 1*2*.. N-1 * N

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

↳ Tracing TODO

Function Call Tracing:

```
int add(N, M){
    return (N+M)
}

int mul(x, y) {
    return (x*y)
}

int sub(x, y) {
    return (x-y)
}
```

```
main(){
    x = 10, y = 20
    print(sub(mul(add(x,y), 30), 75))
}
```

print(sub(mul(add(x,y), 30), 75)) : 825

sub(mul(add(x,y), 30), 75) : return 825 (900)

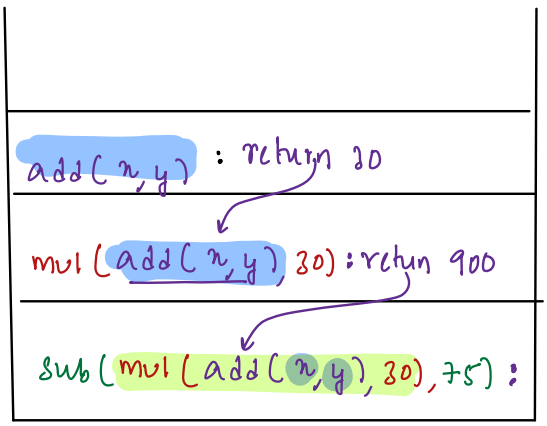
mul(add(x,y), 30) : return 900 (30, 30)

add(x,y) : return 30 (10, 20)

Data Structure :

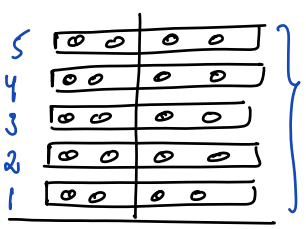
Obs1: When ever a function calls we add the function call at top

Obs2: When function return we remove it from top



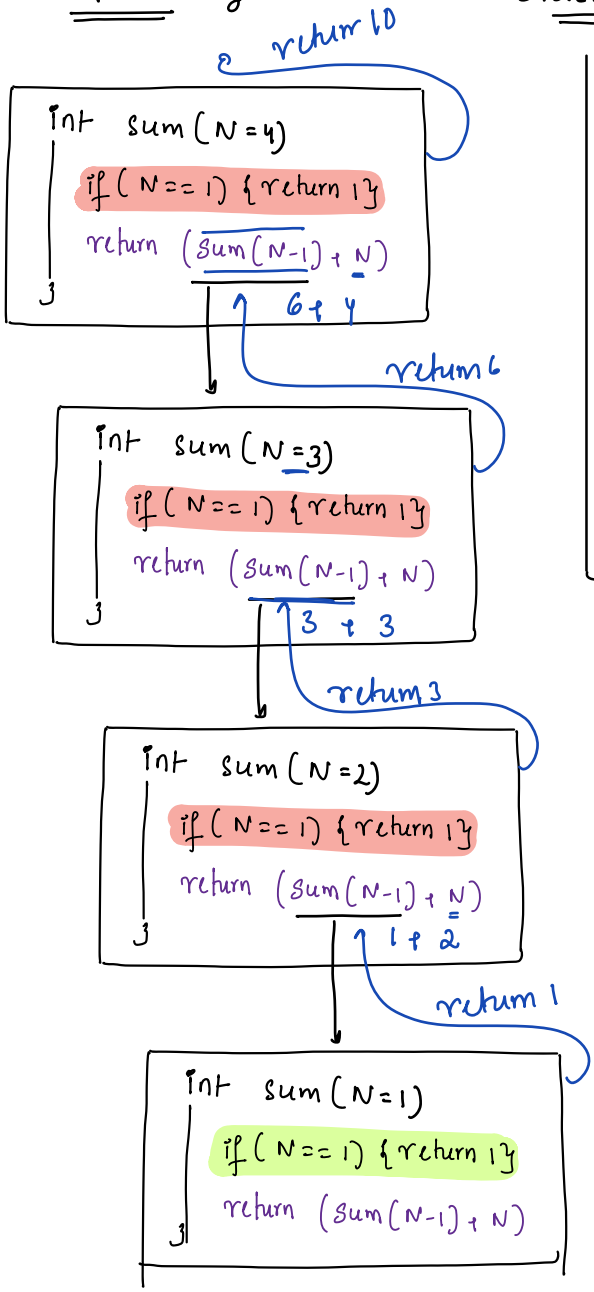
Note: All function calls are stored in Stack

Ex:

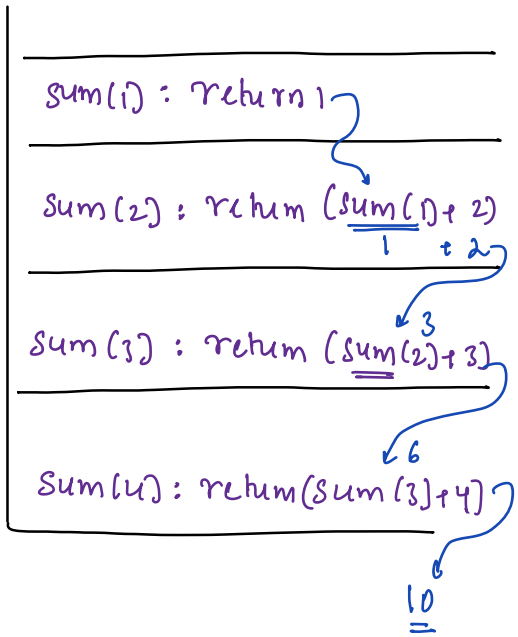


→ property: LIFO → Last in first out

// Sum Tracing :



Stack Trace :



Without Base Condition : Recursion won't stop, memory limit occurs first

- a) TLE
- b) Memory limit Exceeded / Stack Overflow

→ Stack limit can be changed, in your language of choice

Note: In Recursion, if your code gives Memory limit Exceed, that means, code is not properly stopped, verify base conditions

N >= 0

#Input	0	1	2	3	4	5	6	7	8	9	10
fib( ):	0	1	1	2	3	5	8	13	21	34	55

```
int fib(N) { Ass: Calculate & return N^th fib number
    if (N <= 1) { return N }
    return { fib(N-1) + fib(N-2) }
}
```

fib(5) = 5
fib(7) = 13

fib(N) = sum of previous 2 fib numbers

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

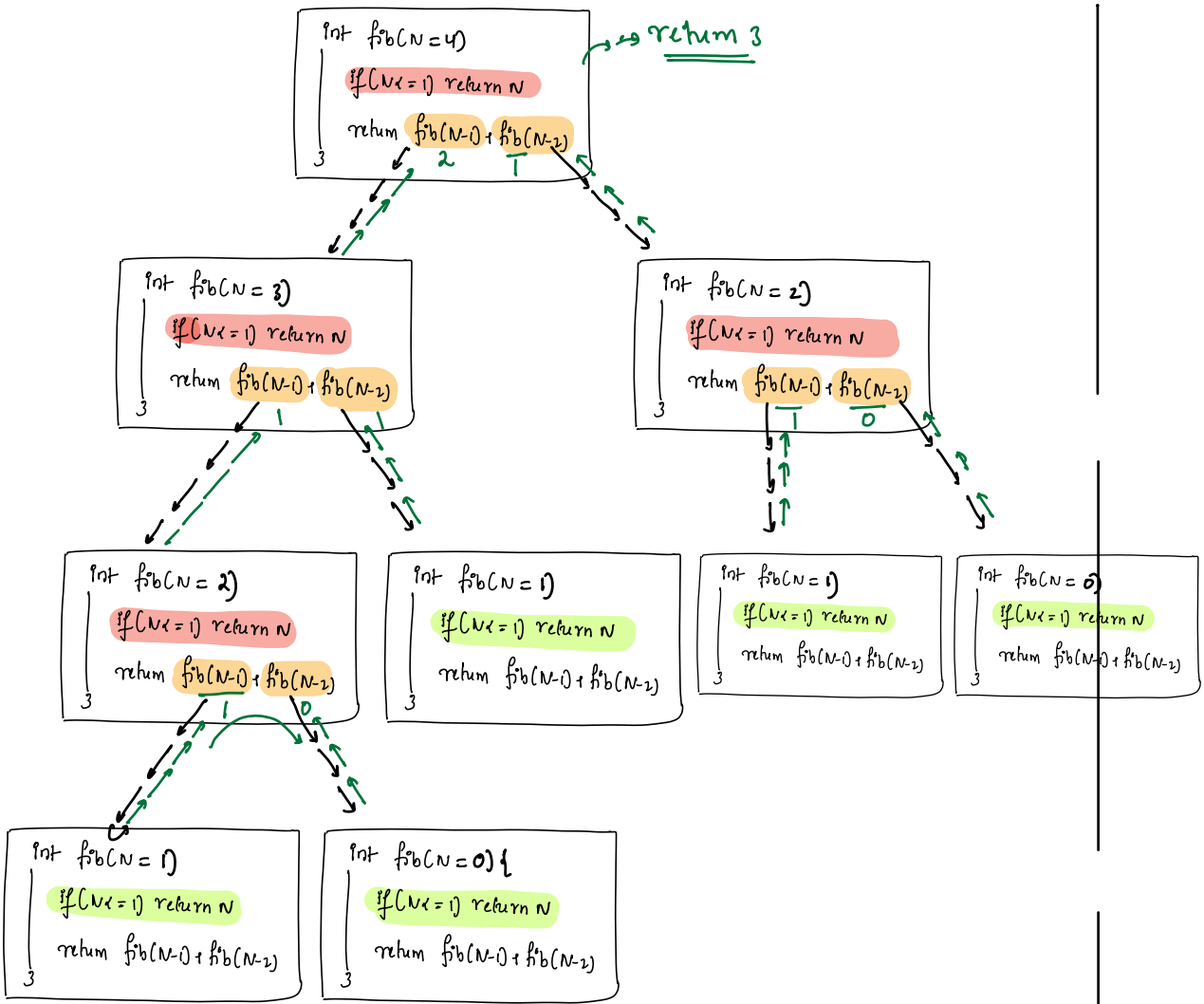
// Note: How to figure out conditions:

↳ For which valid, inputs expression is invalid

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

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

fib(2) = fib(1) + fib(0) ✓



→ Try tracing with stack → TODO

// given N, print all numbers from 1 → N in increasing order

void Inc(N){ Ass: Given N, print all numbers from [1, N] in Inc

```
if (N==1) { print(1), return }
Inc(N-1) // 1, 2, 3 .. N-1,
print(N) //                  N
```

}

Stack Trace → TODO

Inc(3) → 1 2 3

Inc(4) → 1 2 3 4

Inc(N) : 1 2 3 .. N-1, N

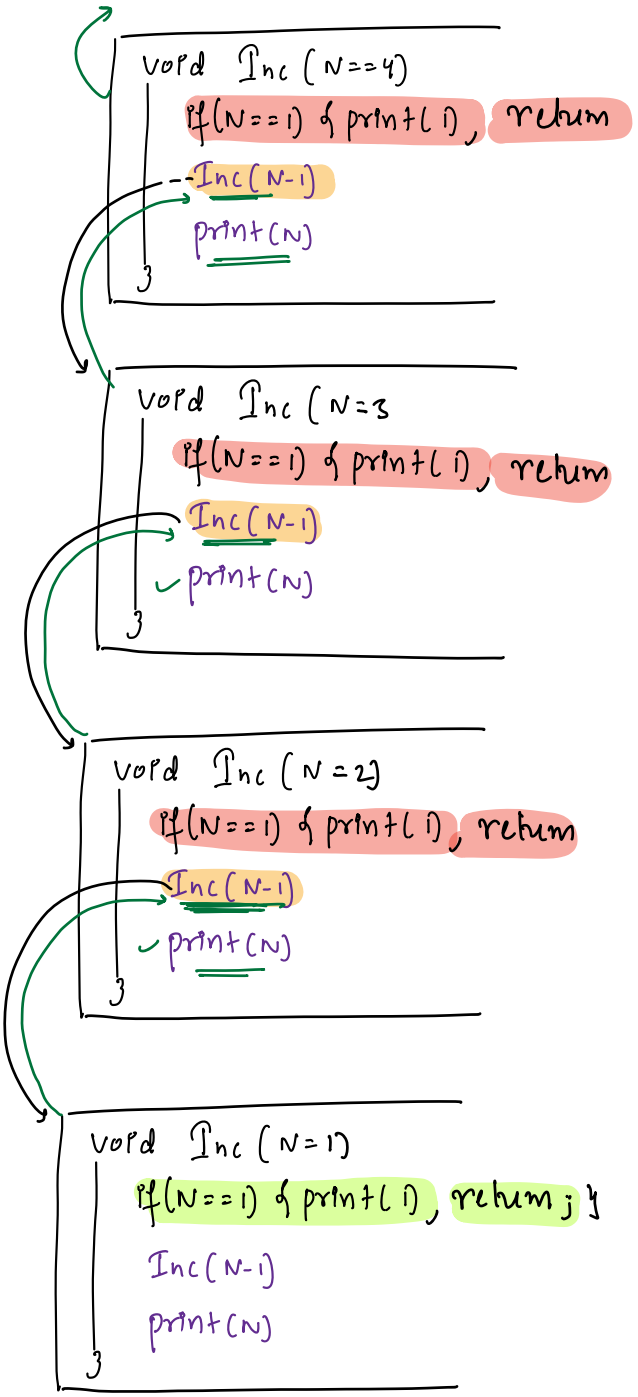
Inc(N-1)
print(N)

Base:

Inc(1) = [Inc(0) print(1)] *

TODO:

void dec(N): Ass: Given N print all number from N → 1



output

print(1)
print(2)
print(3)
print(4)

Note1: Even for void return type we can return inside function.

Note2: Once a function is completely executed it will go back to the parent function

5(8) Given a Substring Check if it's palindrome or Not?

Eg1: g o o d d a d (0 1 2 3 4 5 6) s=4, e=6 → return True

Eg2: g o o d d a d (0 1 2 3 4 5 6) s=2, e=5 → return false

Ass: Return, if substring from [s - e] is palindrome or Not?

```
bool ispal(char ch[], int s, int e){
    if(s > e){return True}
    if(ch[s]==ch[e] &&
            ispal(ch, s+1, e-1))
        return True
    return false
}
```

s >= e

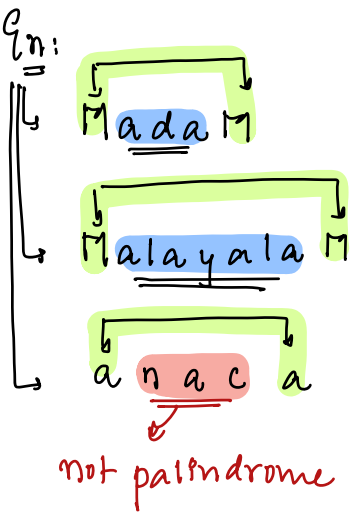
ch[ ]: s, s+1, ..., e-1, e

→ if(ch[s]==ch[e])

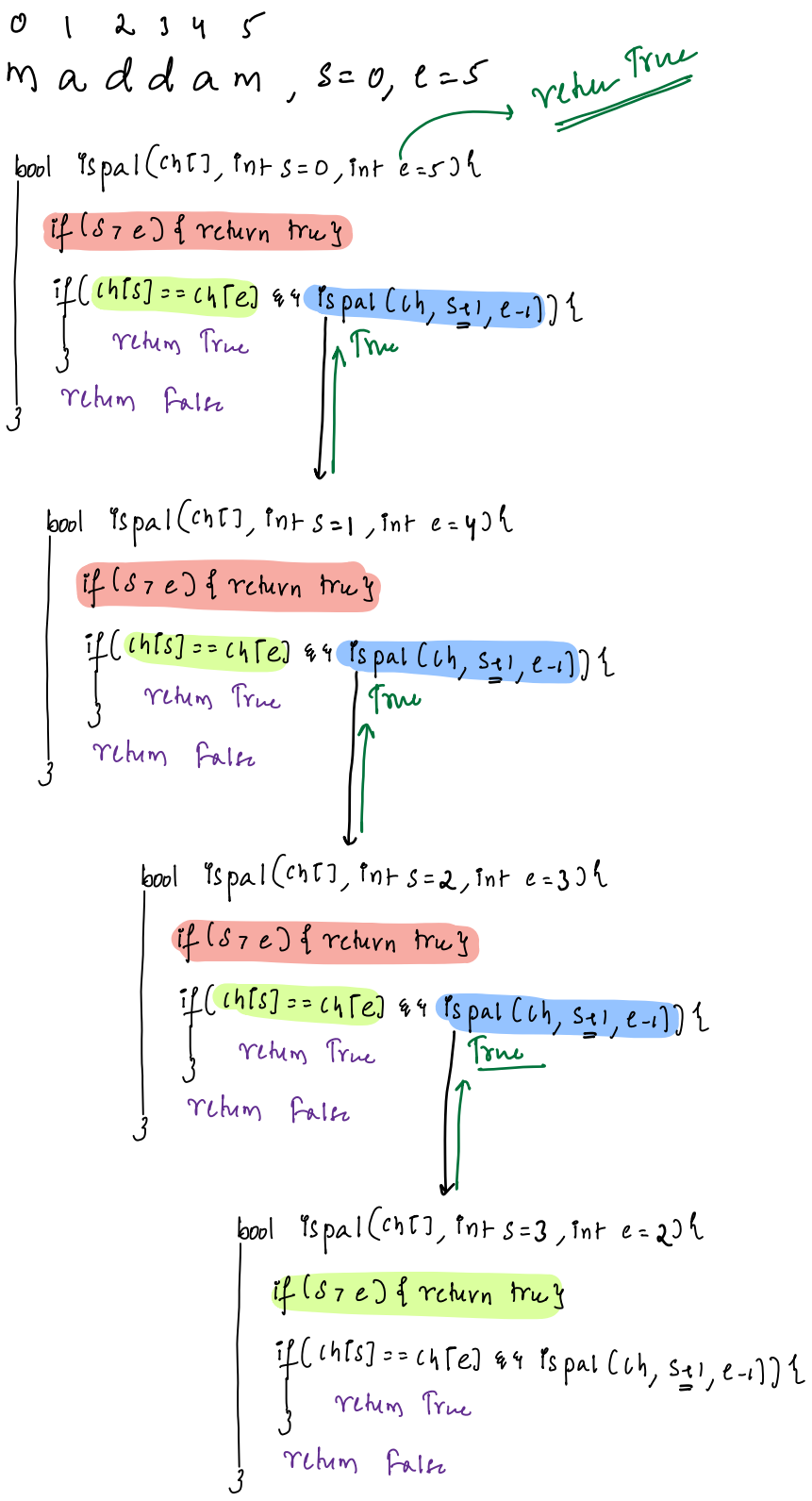
→ Substring from [s+1, e-1] should be a problem

ispal(ch, s+1, e-1)

Eg:



Input: m a d d a m , s=0, e=5



Input: 0 1 2 3 4 5 6
a n m e t n a s = 0, e = 6

```
bool ispal(ch[], int s=0, int e=6){
    if(s > e){return true}
    if(ch[s]==ch[e] && ispal(ch, s+1, e-1)){
        return True
    }
    return false
}
```

false

```
bool ispal(ch[], int s=1, int e=5){
    if(s > e){return true}
    if(ch[s]==ch[e] && ispal(ch, s+1, e-1)){
        return True
    }
    return false
}
```

false

```
bool ispal(ch[], int s=2, int e=4){
    if(s > e){return true}
    if(ch[s]==ch[e] && ispal(ch, s+1, e-1)){
        return True
    }
    return false
}
```

F

```
bool ispal2(char ch[], int s, int e){
    if(s > e){return True}
    if(ch[s]==ch[e]){
        return ispal(ch, s+1, e-1)
    }
    return false
}
```

This function will return, if substring from [s+1, e-1] is palindrn or not

Note: Don't use post increment operators, while writing recursion, Use pre increment