Todays Content:

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

Why Recursion?

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

Recursion: → function calling itself → Recursion

→ Solving 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

→ Smaller Instance of Same Problem / Sub Problem

How to write Recursive Code ?

Assumption : Decide what your function does

Main logic : Solve Assumption using Sub Problems

Smaller Instance of Same Problem

Base Condition : Inputs for which we need to stop

```
int sum(N){   Ass: Given N, Calculate & return sum of N Natural Numbers
    if(N==1){return 1;} // Base condition at start?
    return (sum(N-1) + N)
}
```

Sum of 1st N-1 Natural Numbers + N = Sum of N Natural

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

```
int fact(N){  Ass: Given N calculate & return N!
    if(N==1){return 1;}
    return [ fact(N-1) * N ]
}
```

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

(mul(add(x,y),30) = 900)

mul(add(x,y),30) → return 900

(add(x,y) = 30, 30)

add(x,y) {return x+y} → 30 (10, 20)

Data Structure :

obs1: When ever function call, Insert function call top

obs2: When we return function come out

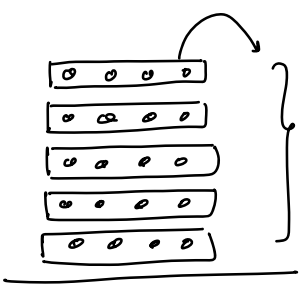
obs3: Insert at top & We delete at top ⇒ Stack

10 20
add(x,y) : return 30 // Once returned it will come out

30
mul(add(x,y),30) : return 900 // Once returned it will come out

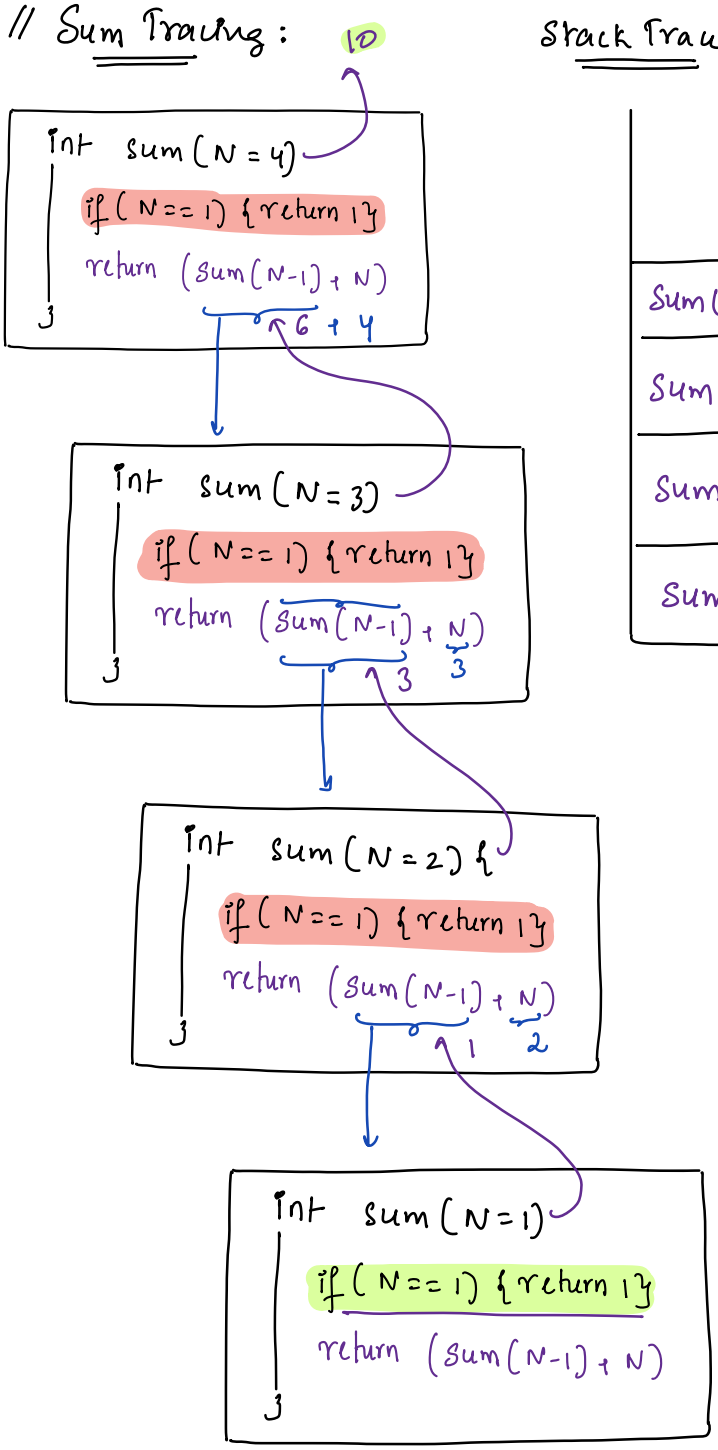
900
sub(mul(add(x,y),30),75) : return 825 // It will come out

Insert at top & We delete at top

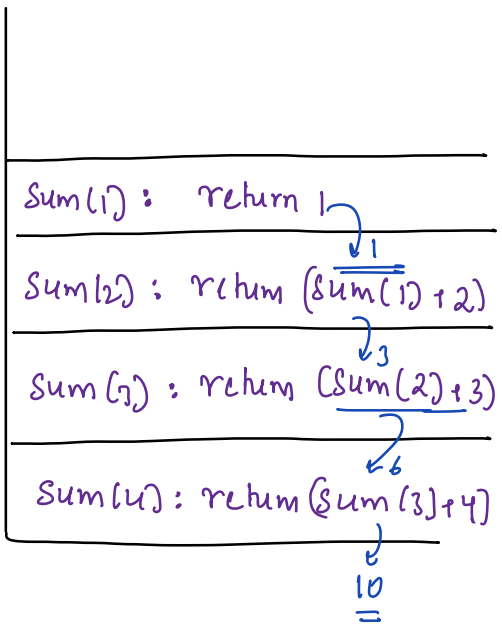


Stack of plates

// Sum Tracing :



Stack Trace :



Without Base Condition : Function Calls won't stop

→ we get below cases

Infinite loop

TLE

Insert function calls in Stack

Stack overflow / Memory limit Exceeded

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

N >= 0

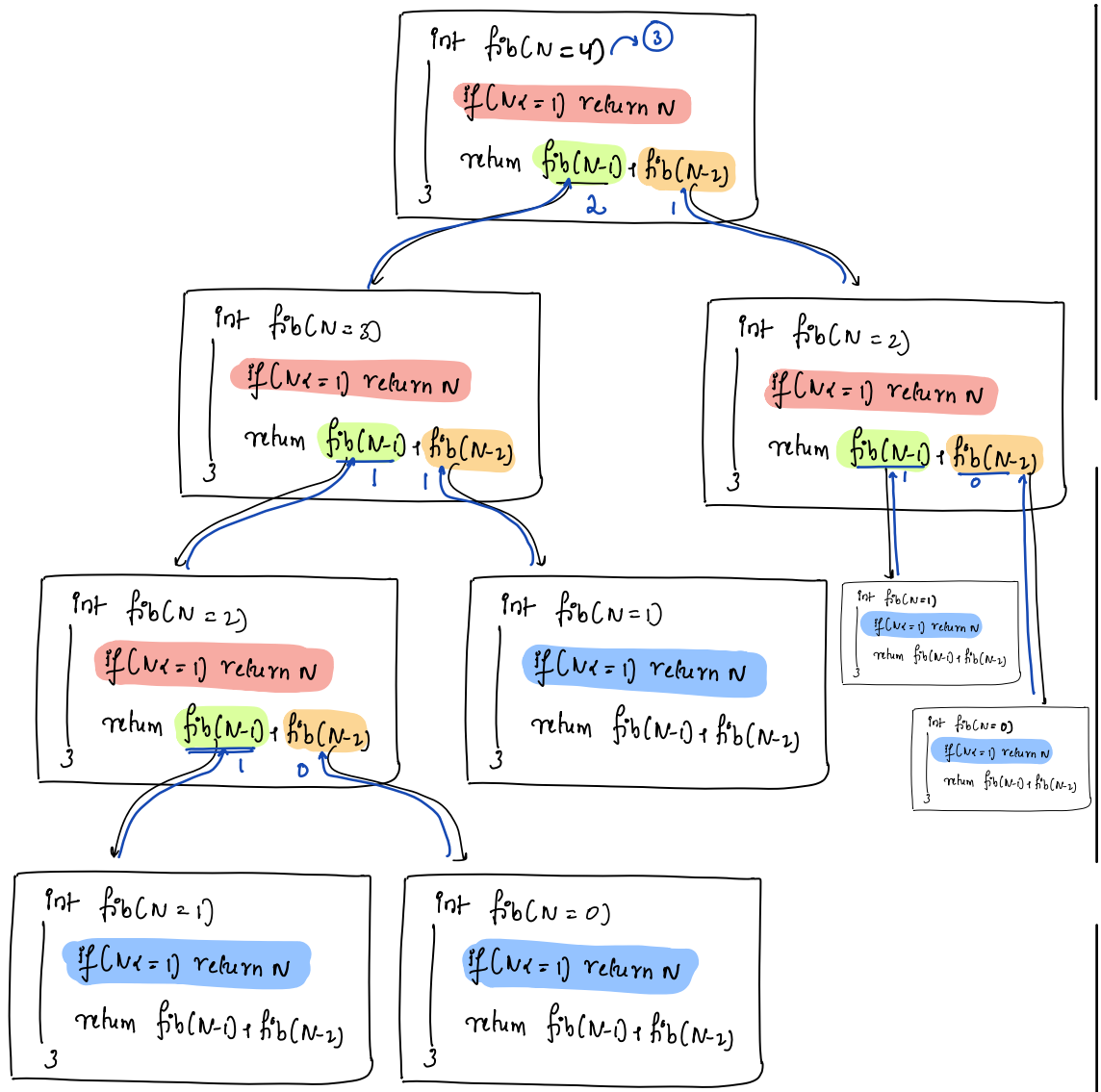
#Input	0	1	2	3	4	5	6	7	8	9	10
fib( ):	0	1	1	2	3	5	8	13	21	34	55

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

N-1^th fib number N-2^nd fib number

{ fib(0) = fib(-1) + fib(-2)
  fib(1) = fib(0) + fib(-1) }

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



→ TODO for Stack Trace

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

void Inc(N){ Ass: Given N, print all numbers from 1 - N

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

3

output

Inc(3) → 1 2 3

Inc(4) → 1 2 3 4

Inc(N) = 1 2 3 4 5 .. N-1 + N

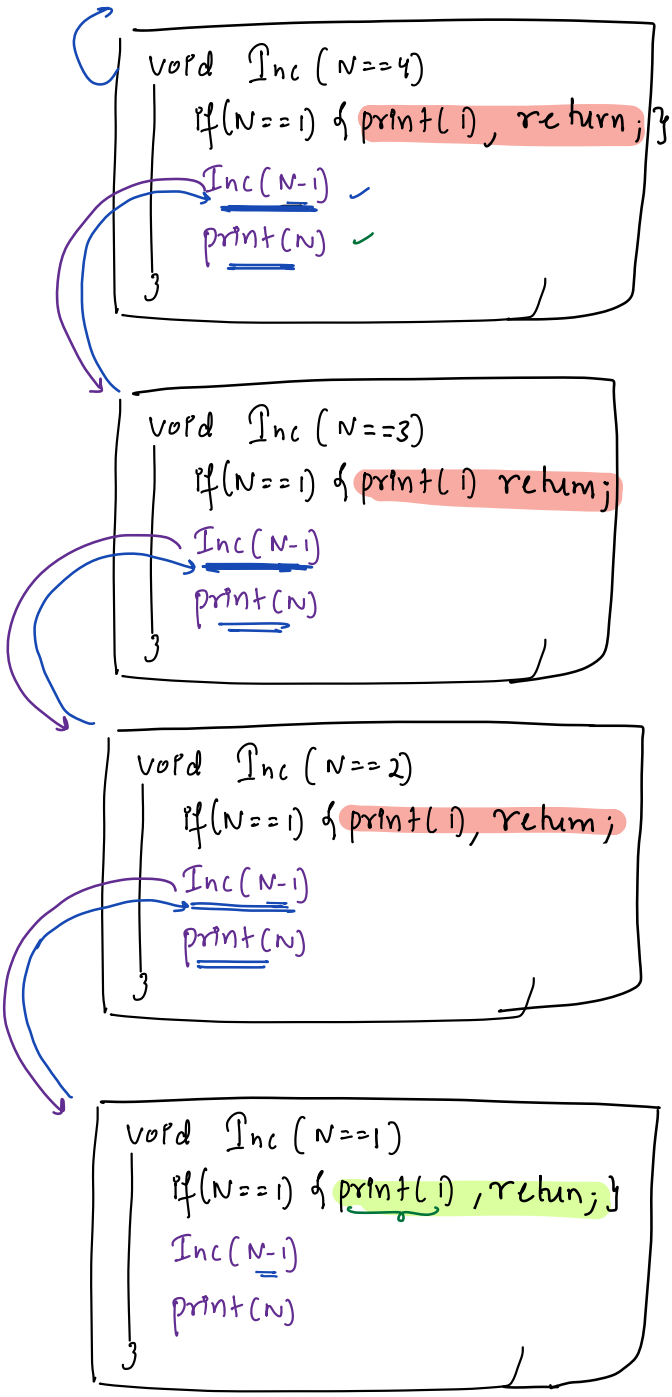
$$Inc(N) = \begin{cases} Inc(N-1) \\ print(N) \end{cases}$$

$$Inc(1) = \begin{cases} Inc(0) \\ print(1) \end{cases}$$

Base

TODO:

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



output

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

Note: When function is completely executed it will automatically return to function which calls

Q) Given a Substring Check if it's palindrom or Not?

Ex1: 0 1 2 3 4 5 6
g o o d d a d s=4, e=6 → return True

Ex2: 0 1 2 3 4 5 6
g o o d d a d s=2, e=5 → return False

Ass: Return if given substring [s e] is palindrom 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
}
```

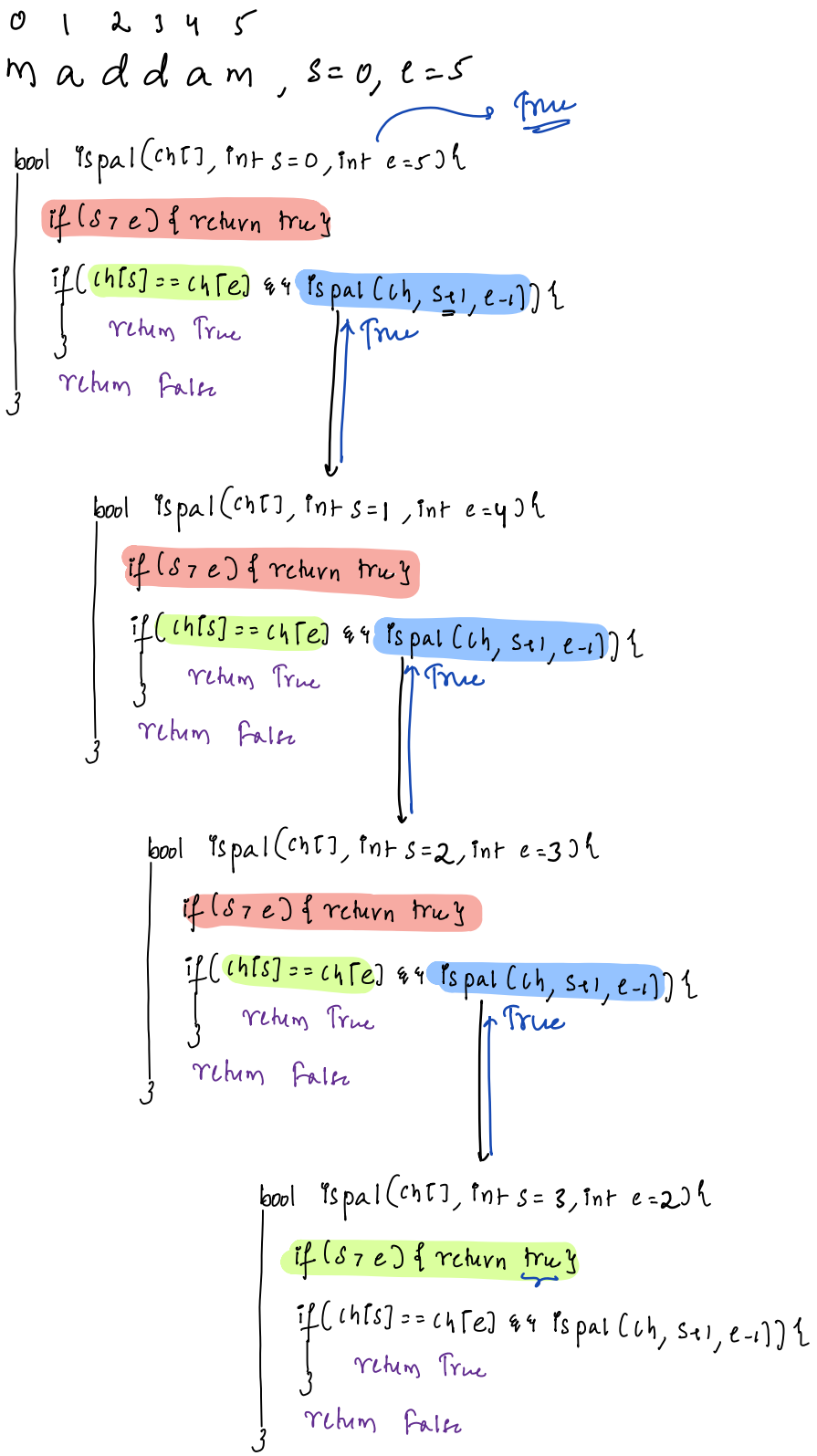
ch[]: s | s+1 ... e-1 | e

To check, ch[s - e] is palind
ch[s] == ch[e] &&
Substring from [s+1, e-1] should also be a palindrome

Ex1: m a d a m

Ex2: M a l a y a l a M

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;
}
```

false

// Increment:

postfix prefix

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

```
int n = 10
int y = n++
print(y)
```

10

```
int n = 10
int y = ++n
print(y)
```

11