1 → Recursion.

Todays Content:

- → Recursion?
- → How to write a Recursive Code / Tracing
- → T.C → Next class.

Why Recursion?

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

Recursion :- function calling itself → Recursion.

↳ Solving problem, using smaller Instance of same Problem

Subproblem.

$Sum(N) = 1 + 2 + 3 + \dots N$

$Sum(4) = Sum(3) + 4$

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

Bigger Problem

↳ Subproblem

How do we write a recursive code?

1) Assumption :- you will assum your function works for subproblems.

↳ faith !

2) Main logic :- Solve bigger problem with subproblem.

3) Base Condn :- Just write the answer for smallest input you know.

```
int sum(N) {
    if (N==1) {
        return 1;
    }
    return sum(N-1) + n
}
```

Ans:- Given, any value $k<n$, sum(k) will give me the sum of first k numbers.

fact(3) = 1*2*3 , fact(4) = 1*2*3*4.

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

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

Ans:- fact function will work for anything smaller than N,

1 * 2 * 3 ... N-1

function call Tracing :-

```
int fun (N) {
    return gun (N+5)
}

int gun (N) {
    return sun (N+2)
}
```

2

64

3

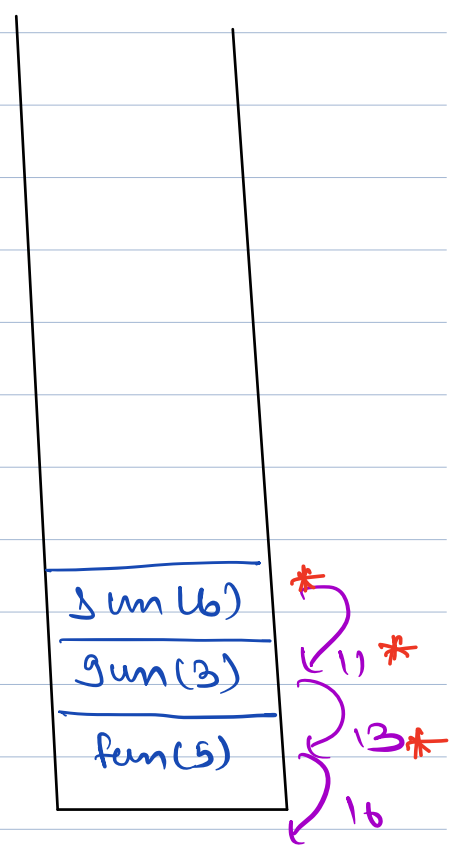
64

sun(8) *
gun(6) *
fun(1) *

```
int sum (N) {
    return N * N   » 64
}

main ( ) {
int r = fun (1);   ①
}
```

Ex 2 :-

```
int fun (N) {
    ret gun (N - 2) + 3
}

int gun (N) {
    ret sun (N+3) + 2
}

int sun (N) {
    ret 5 + N
}
```



```
main ( )  {
    print (fun(5))  → 16
}
```

Tracing , N = 4

```
int sum(N=4) {
    if (N==1) {
        return 1;
    }
    return sum(N-1) + n      → 10
}
```

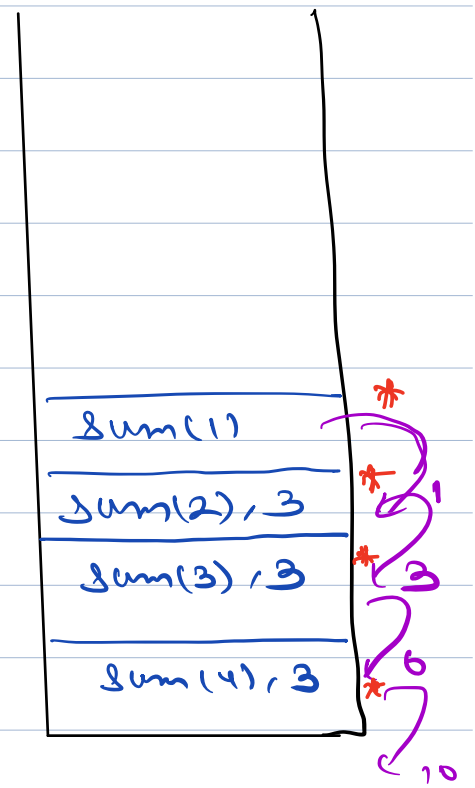
```
int sum(N=3) {
    if (N==1) {
        return 1;
    }
    return sum(N-1) + n      → 6
}
```

```
int sum(N=2) {
    if (N==1) {
        return 1;
    }
    return sum(N-1) + n      → 3
}
```

```
int sum(N=1) {
    if (N==1) {
        return 1;
    }
    return sum(N-1) + n
}
```

// sum(4)

```
int sum(N) {
    ① if (N==1) {
    ②     return 1;
        }
    ③ return sum(N-1) + n
}
```



10:04 → 10:10 pm.

Note:- In recursion, if your codes gives memory limit Exceeded (Stack Overflow), that means code is not properly stopped.

< verify base Conditions >

:- Wrong Base case :-

TLE * Stack Overflow

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

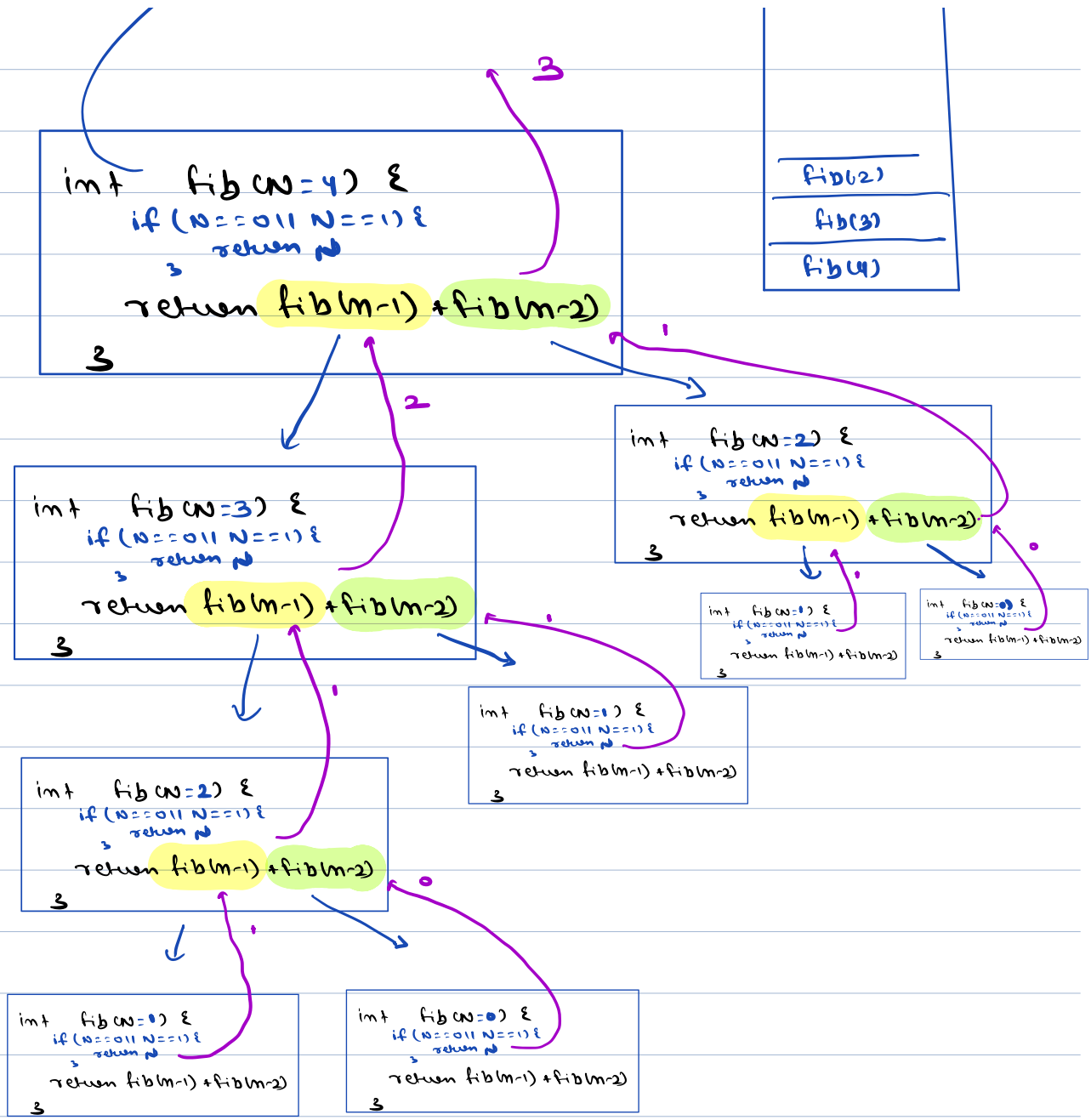
```
int fib(N) {
    if (N==0 || N==1) {
        return N
    }
    return fib(n-1) + fib(n-2)
}
```

// Assumption, fib() will work for smaller inputs.

fib(n-1) → N-1th fib

fib(n-2) → N-2th fib.

```
int fib(N=4) {
    if (N==0 || N==1) {
        return N
    }
    int x = fib(n-1)
    int y = fib(n-2)
    return (x+y).
}
```



Todo for Stack Trace :-

Ques) Given N, print all numbers from 1-N in increasing order.

N = 5 → 1, 2, 3, 4, 5

N = 3 → 1, 2, 3.

```
void PrintIncreasing (N)
{
        if (N==1) {
            Print(1)
            return;
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

PrintIncreasing (5)
↓
PrintIncreasing (4)
Print (5).

Assumption → PrintIncreasing will print numbers from 1 to k for k<n.

```
void PrintIncreasing (N=4)
{
        if (N==1) {
            Print(1)
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

```
void PrintIncreasing (N=3)
{
        if (N==1) {
            Print(1)
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

```
void PrintIncreasing (N=2)
{
        if (N==1) {
            Print(1)
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

← output →

1

```
void PrintIncreasing (N=1)
{
    if (N==1) {
        Print(1)
    }
    PrintIncreasing(N-1)
    Print(N)
}
```

```
void PrintIncreasing (N=0)
{
    if (N==1) {
        Print(1)
    }
    PrintIncreasing(N-1)
    Print(N)
}
```

* Correct one :-

: output :-

1
2
3
4

```
void PrintIncreasing (N=4)
{
    if (N==1) {
        Print(1)
        return;
    }
    PrintIncreasing(N-1)
    Print(N)
}
```

↓

```
void PrintIncreasing (N=3)
{
    if (N==1) {
        Print(1)
        return;
    }
    PrintIncreasing(N-1)
    Print(N)
}
```

```
void PrintIncreasing (N=2)
{       if (N==1) {
            Print(1)
            return;
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

```
void PrintIncreasing (N=1)
{       if (N==1) {
            Print(1)
            return;
        }
        PrintIncreasing(N-1)
        Print(N)
}
```

Ques) Given a substring check if its palindrome 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.

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

    } else {
        return false;
    }
```

Assumption:- checksubstring knows how to check smaller strings.

Ex1:- m a d a m

Ex2:- Malayalam

0 1 2 3 4 5 6 7 8
Malayalam

1 2 3 4 5 6 7
a layala

2 3 4 5 6
l a y a l

3 4 5
a y a

s e
4
y

s→5 e
3

True

s→4, e→4,

s e
3

4 4

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

checkString (5, 3)

True

```
    return false;
}
```