* (I) How write recursion
* (II) How it works
(III) TC / SC of recursive codes

- Merge & Quick sort
- Binary Trees / BST / Heaps
  Tries / Seg Trees
- Dynamic programming
- Backtracking

Recursion A function calling itself.

Sum(N) : 1 + 2 + 3 + 4 ---- + N

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

Sub problem

Steps

(I) **Assumption**
- Decide what your function does.
- Assume it returns the correct ans.

(II) **Main Logic** Solve the original problem using the sol of subproblems (Recursive eq)

(III) **Base Condition** When should your recursion stop.

Sum(N) using recursion.

```
int sum(N) {
  // Assumption: sum(N) returns the correct sum of N natural no.

  // Base condition
    if (N == 1) ret 1;

  // Main logic
  → ret  sum(N-1) + N;

}
```

fact(N) using recursion (non -ve numbers)

$3! = 3 \times 2 \times 1$

$5! = 5 \times 4 \times 3 \times 2 \times 1$

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

$N! = N \times \text{fact}(N-1)$

$1! = 1$

$0! = 1$

```
int fact(N) {
  // Assumption  fact(N) will return the correct value of N!

  // Base condition
      if (N == 0 || N == 1) ret 1;

  // main logic
      ret  N x fact(N-1);
}
```

N = 1

1 x fact(0)

N=0

0 x fact(-1)

×

fib(N) → N^{th} fibonacci no.

Golden Ratio

1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	5	8	13	21	34	55	89 ...
							↑ (N-2)	↑ (N-1)	↑ N^{th}	

```
int fib(N) {
// Assumption: fib(N) gives the correct Nth fibonacci no.

// Base condition
   if (N == 1 || N == 2) {
        ret 1;
   }

// Main logic
   ret fib(N-1) + fib(N-2);
}
```

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

$N = 2$

$fib(2) = fib(1) + fib(0)$ ✗

$N = 3$

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

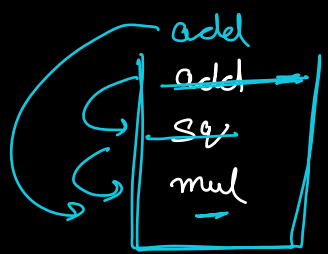
```
int add (N, M){
    ret N+M;
}
```

```
int sqm (N){
    ret N*N;
}
```

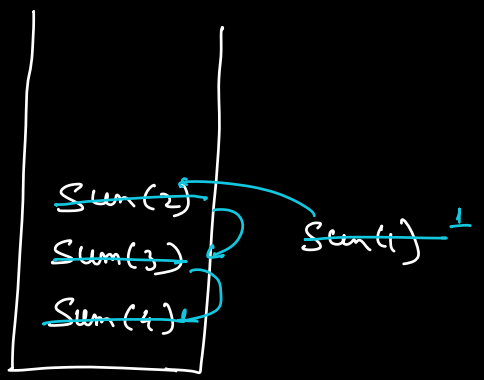
```
int mul (N, M){
    ret N * M;
}
```

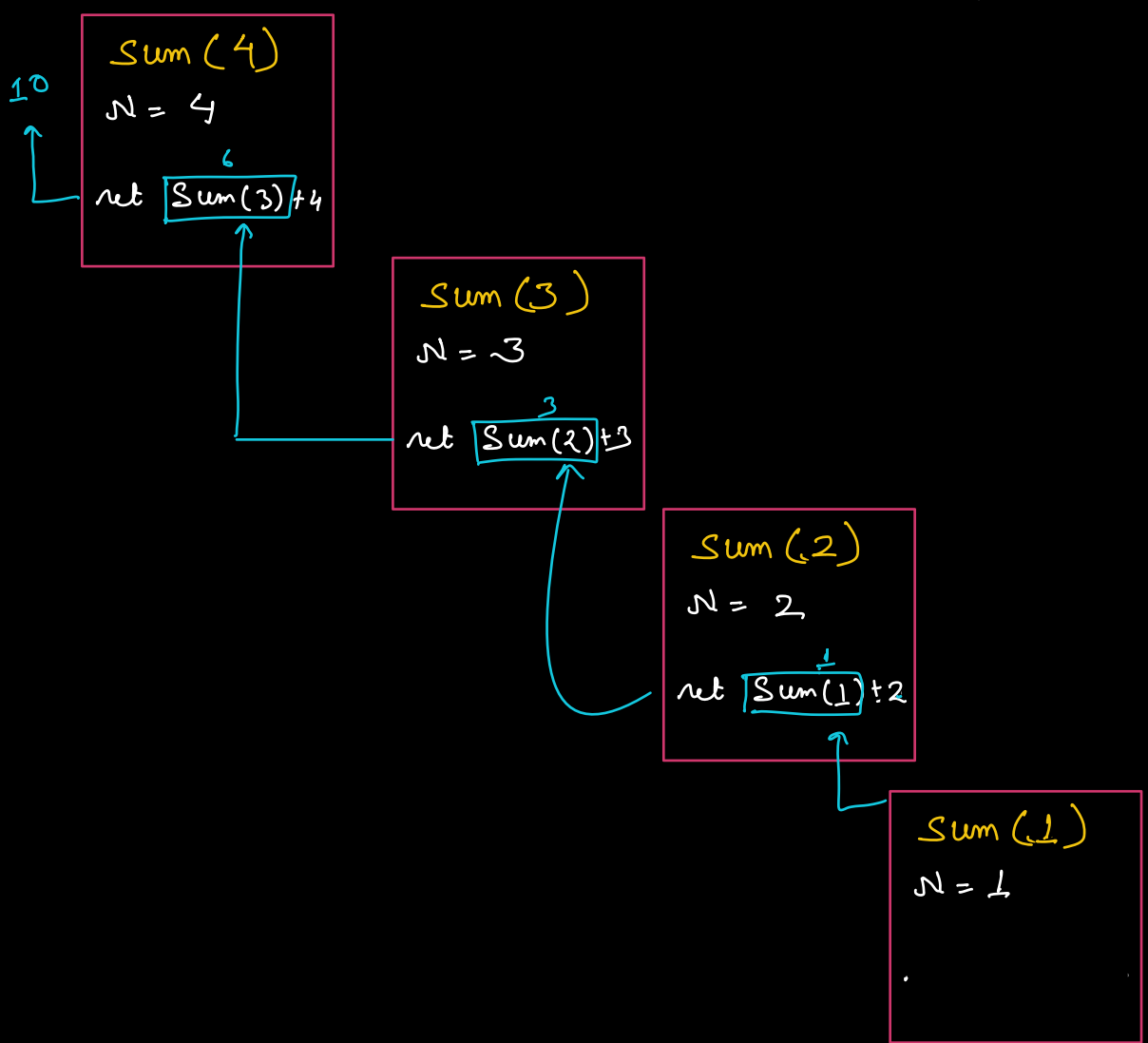
```
⇒ int main (N, M){
    int a = add (N, M);
    int b = sqm (a);
    int c = mul (b, a);
    ret c;
}
```

```
ret mul (sqm (add (N, M)), add (N, M));
```

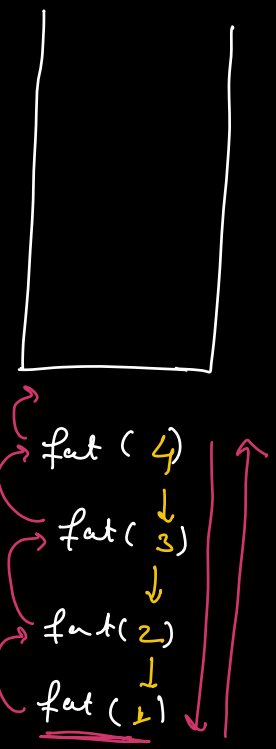


```
int sum(N) {
  if (N == 1) ret 1;
  ret sum(N-1) + N;
}
```





```
int fact(N) {
    if (N == 0 || N == 1) ret 1;
    ret N * fact(N-1);
}
```



24

fact(4)
N = 4
ret fact(3) * 4
6

fact(3)
N = 3
ret fact(2) * 3
2

fact(2)
N = 2
ret fact(1) * 2
1

fact(1)
N = 1

Q. Print no. from 1 to N in increasing order using recursion.

```
void PrintIncreasing (N) {
// Assumption : PrintInc (N) prints no. from
                1 to N in increasing order
// Base condition
   if (N == 1) {
       Print(1)
       return
   }
// Main logic
   PrintIncreasing (N-1);
   Print(N).
}
```

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

(No from 1 to N-1 in increasing or)

1, 2, 3, 4, 5, ... N-1, N

1, 2, 3, 4 ... N-1, N

PrintInc (3)
N=3
PrintInc (2)
Print(3)

PrintInc (2)
N = 2
PrintInc (1)
Print (2)

PrintInc (1)
N = 1
Print(1) ①
ret

1
2
3

Print in decreasing order.

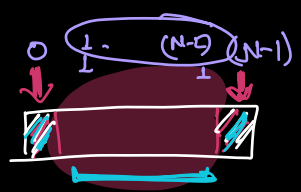
N , N-1 , (N-2), (N-3) , (N-4) ... 4, 3, 2, 1

↓

Print Dec (N-1)

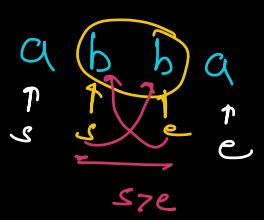
Q Given a string.
Write a recursive program to check if it is a palindrom.

Palindrom ⇒ → NAMAN ←
→ MADAM ←
→ NAYAN ←
→ DOOD ←
→ LEVEL ←




```
boolean isPalindrome ( Str, s, e ) {
// Assumption : isPalindrom (Str, s, e) returns true
                if the sub-string from s to e is a
                palindrom
// Base condition
if (s >= e)  ret true;

// Main logic
if ( Str[s] == Str[e]) {
        ret isPalindrom (Str, s+1 , e-1);
}
ret false;
}
```



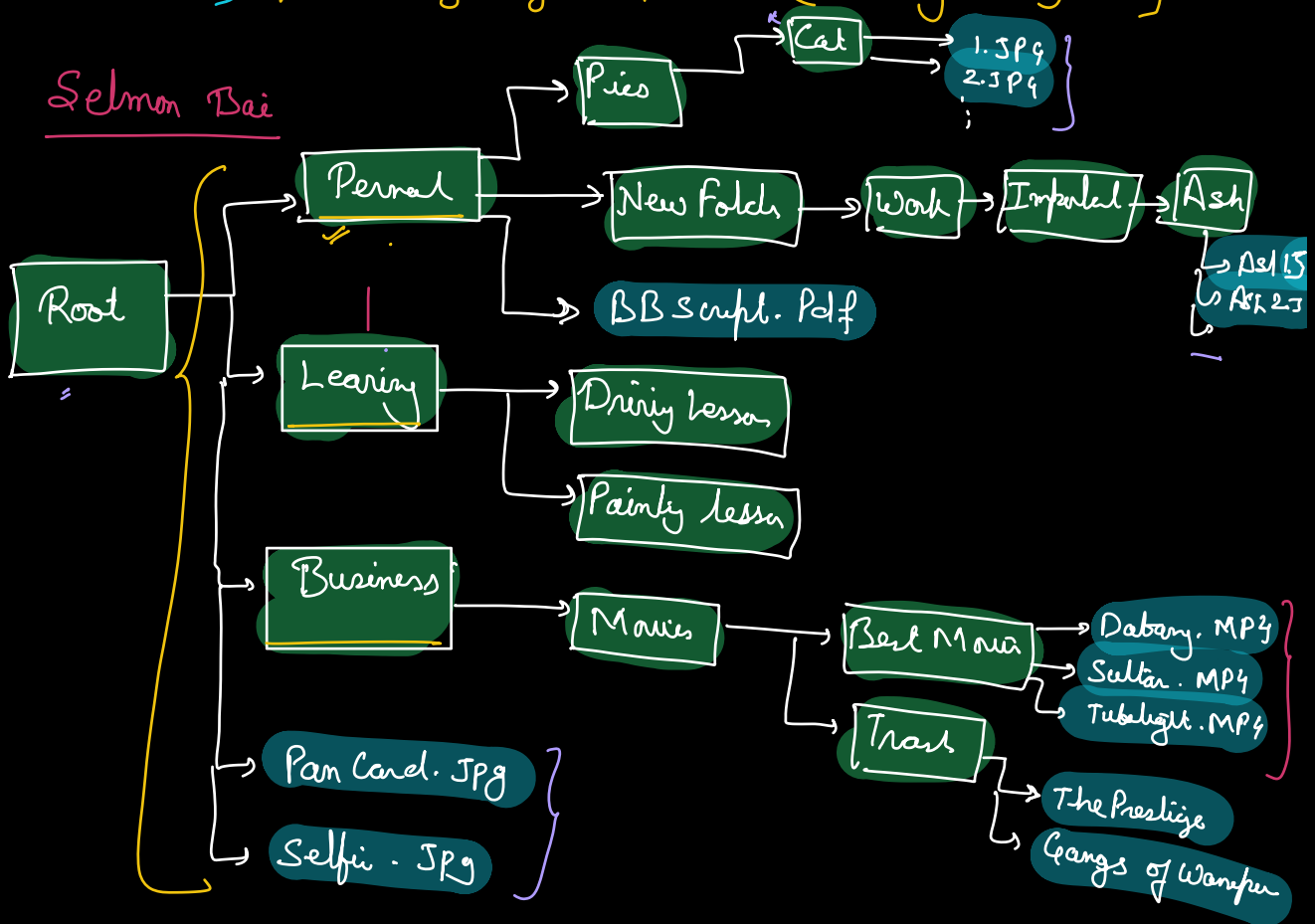
Facebook Google Q Given a dir structure search a file in it.

Given

List<Str> getAllDirectories(DirectoryName)

⇒ List<Str> getAllFiles(DirectoryName)

Selmon Bai



Root, Dabang.MP4

Pers, Dabang.MP4

Pic,

<Movie>

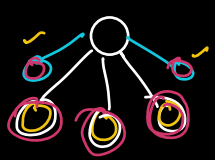
↓

<Best Movie, Trash>

```
boolean search ( DirectoryName, FileName ) {
  // Assumption : search(DN, FN) returns true
                  if FN is present anywhere inside the directory.

  // Check if FN is present directly inside the DN.
  List<Stg> files = getAllFiles ( DirectoryName );
  for ( i=0; i < files.size(); i++ ) {
      if ( files[i] == fileName ) {
          ret true;
      }
  }
```

Base Cond



```
  List<Stg> directories = getAllDir ( DirectoryName );
  for ( i=0; i < directories.size(); i++ ) {
      if ( search ( directories[i], fileName ) )
          ret true;
  }
}
ret false;
}
```

Main Logic