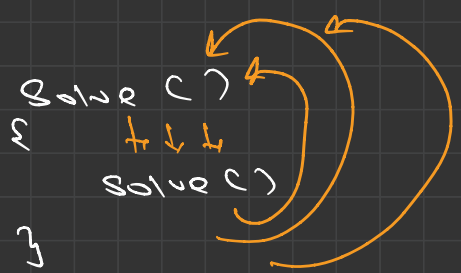
Recursion

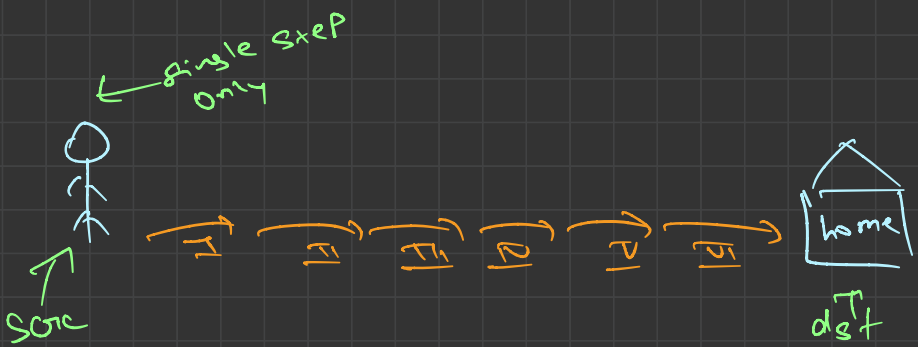
↳ Backish → when a fn() calls itself directly / indirectly



soln of bigger problem

↓ depend

soln of Chotti prob of same type



Big problem → step(6) → Break into → 1 + step(5)

↑ single step

Imp line :- 1 case solve karlo baki recursion sambhal lega

In mathematical terms

$$Solve(n) \rightarrow 2^n$$

Bigger Problem $\rightarrow 2^n$

$$\Downarrow$$

$$Solve(n) = 2 * 2^{n-1}$$

$$\Downarrow$$

$$Solve(n) = 2 * Solve(n-1)$$

Bigger problem

Chotta problem

$$Solve(n-1) = 2^{n-1}$$

```
int pow(int n)
{
    if (n == 0)
        return 1;
    int ans = 2 * pow(n-1);
    return ans;
}
```

What is recursion ?

Desi

↓

Bigger problem soln

↓

depend on

↓

small & same type of problem

Bookish

↓

When a func() call itself

↓

```
void solve() {
    ==
    solve()
}
```

for Ex

$2^5 = 32$

Bigger problem → 2^5

$2^5 = 2 \times 2^4$

2^4 ← small problem

B.P. = $2 \times$ S.P.

$f(n) = 2^n$

$f(n-1) = 2^{n-1}$

$2^n = 2 \times 2^{n-1}$

$f(n) = 2 \times f(n-1)$

factorial

B.P. = 5!

5 × 4! → S.P.

Problem Statement

↳ I/P = n

↳ O/P = n!

Ex

n = 5

O/P = 5! = 120

n = 3

O/P = 3! = 6

B.P = n!

↳ find factorial of n

$n! = n \times (n-1) \times (n-2) \times (n-3) \ldots \ldots 3 \times 2 \times 1$

$(n-1)!$

$n! = n \times (n-1)!$

n! → B.P.

(n-1)! → S.P.

n → 1 multiply karne par n! bolta hu

(n-1) → 1 tak multiply karne par (n-1)! kahega

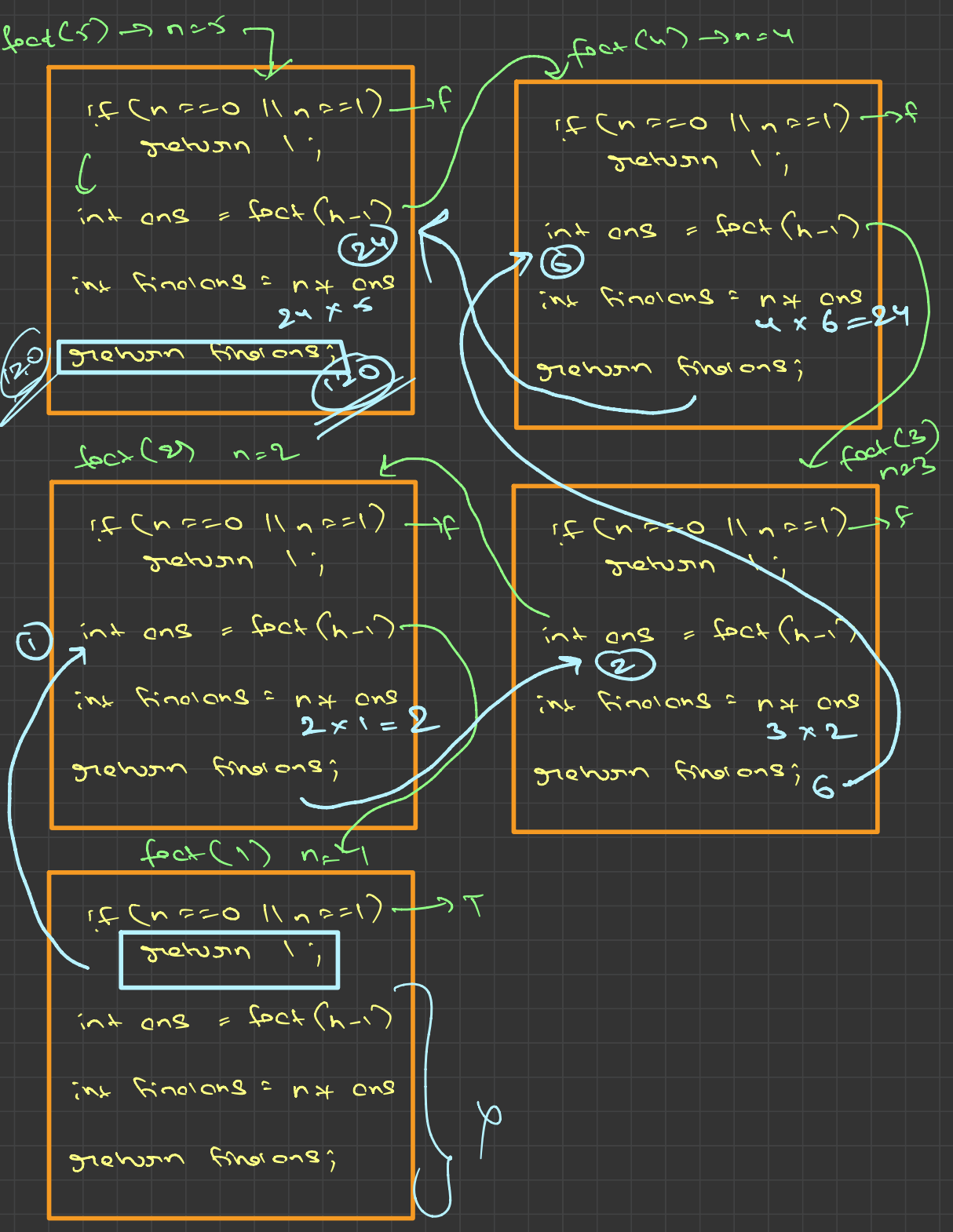
$f(n) = n \times f(n-1)$

```
int main() {
    int n = 5;
    int ans = factorial(n);
    cout << ans;
}
```

```
int factorial(int n) {
    if (n == 1) {
        return 1;
    }
    int ans = n * factorial(n-1);
    return ans;
}
```

Rukne ka kab hai

Again ye n! bolega to func() call hoke aayega


fact(5) → n=5
if (n==0 || n==1) → F
return 1;
int ans = fact(n-1)
int finalans = n * ans
24 x 5
return finalans;
120
fact(4) → n=4
if (n==0 || n==1) → F
return 1;
int ans = fact(n-1)
6
int finalans = n * ans
4 x 6 = 24
return finalans;
fact(3) n=3
if (n==0 || n==1) → F
return 1;
int ans = fact(n-1)
2
int finalans = n * ans
3 x 2
return finalans;
6
fact(2) n=2
if (n==0 || n==1) → F
return 1;
int ans = fact(n-1)
1
int finalans = n * ans
2 x 1 = 2
return finalans;
fact(1) n=1
if (n==0 || n==1) → T
return 1;
int ans = fact(n-1)
int finalans = n * ans
return finalans;

Counting

n = 5

reverse counting

↳ 5, 4, 3, 2, 1

f(n) = print counting from n to 1

f(5) = print counting from 5 to 1

f(5) = print(5) + f(4)

B.P. S.P.

Problem statement

↳ I/P = n

↳ O/P = reverse counting from n → 1

B.P. → f(n) → reverse counting from n → 1

n = 5 ⇒ 5 | 4 3 2 1

B.P. f(n-1) → 1

small problem

f(n) = print(n) + f(n-1)

B.P. S.P.

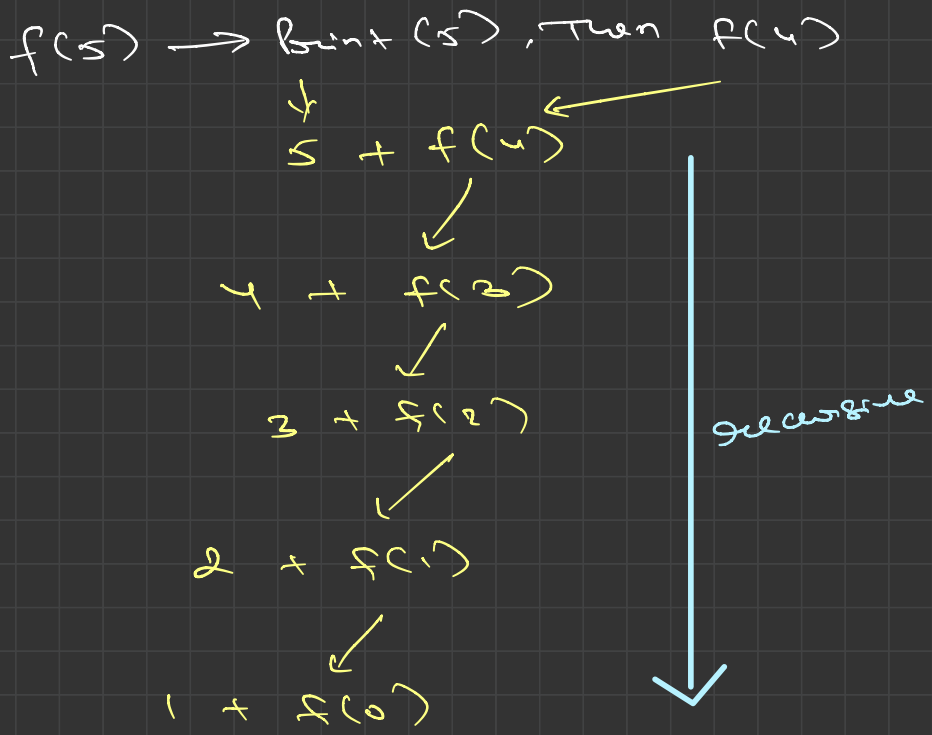
Ex:

n = 5

O/P = 5, 4, 3, 2, 1

n = 2

O/P = 2, 1


f(5) → Print (5) . Then f(4)
5
5 + f(4)
4 + f(3)
3 + f(2)
2 + f(1)
1 + f(0)
Recursive

Where we apply the concept ?

Problem statement

↓

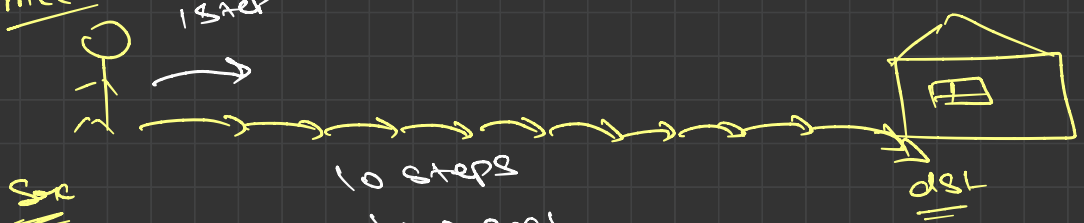
Relation / formula

↓

Bigger problem ——depends——→ small problem
(some type) (same type)

for ex

mee



10 steps = Bigger prob

Agar mein 1 step chal lijo to 9 step
mere liye small prob

9 step = small prob

B.P = 1 + S.P.

total no. of step

total no. of step

Recursive code

Mandatory
- Base condition (stopping condition)
- Recursive relation / recursive call
  - Ex: $f(n) = n \times f(n-1)$
- Processing (optional)
  - Calculations, Print

Problem statement

- I/P = n
- O/P = Print (2^n)

Ex

n = 3

O/P = $2^3 = 8$

I/P = 6

O/P = $2^6 = 64$

B.P. $\Rightarrow 2^n$

$2^n = \underbrace{2 \times 2 \times 2 \times 2 \times \dots \times 2}_{n \text{ times}}$

$$2^n = 2 \times 2^{n-1}$$

(2^n: B.P., 2^{n-1}: S.P.)

$$f(n) = 2 \times f(n-1)$$

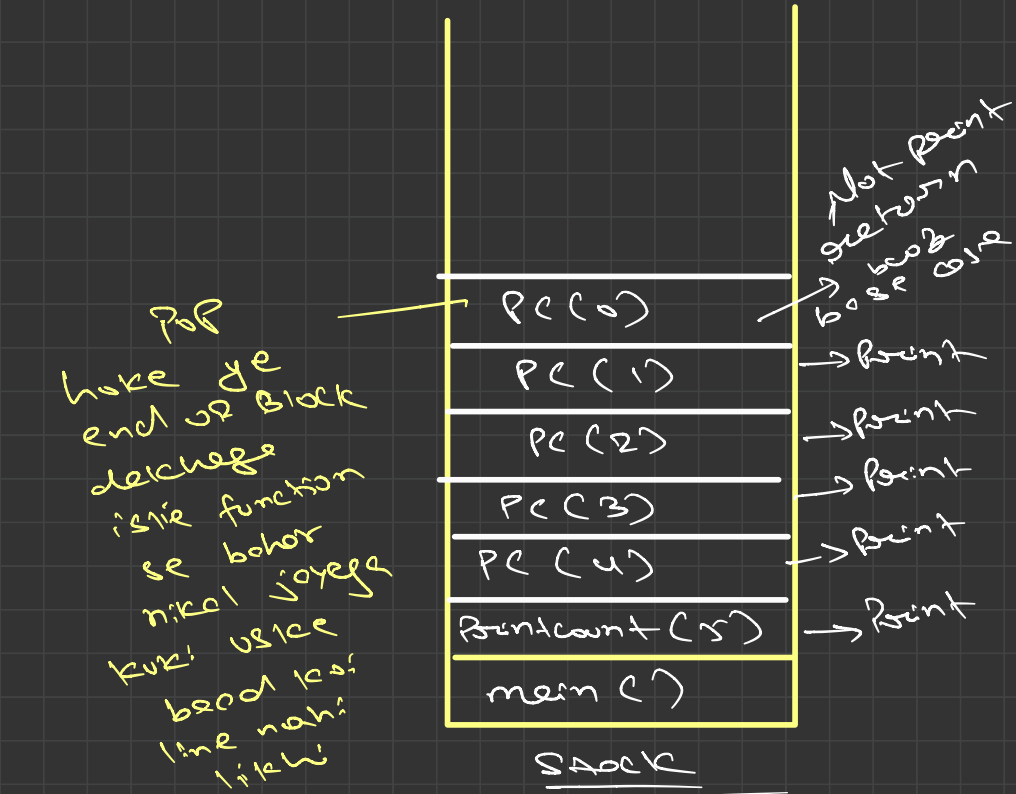
Counting

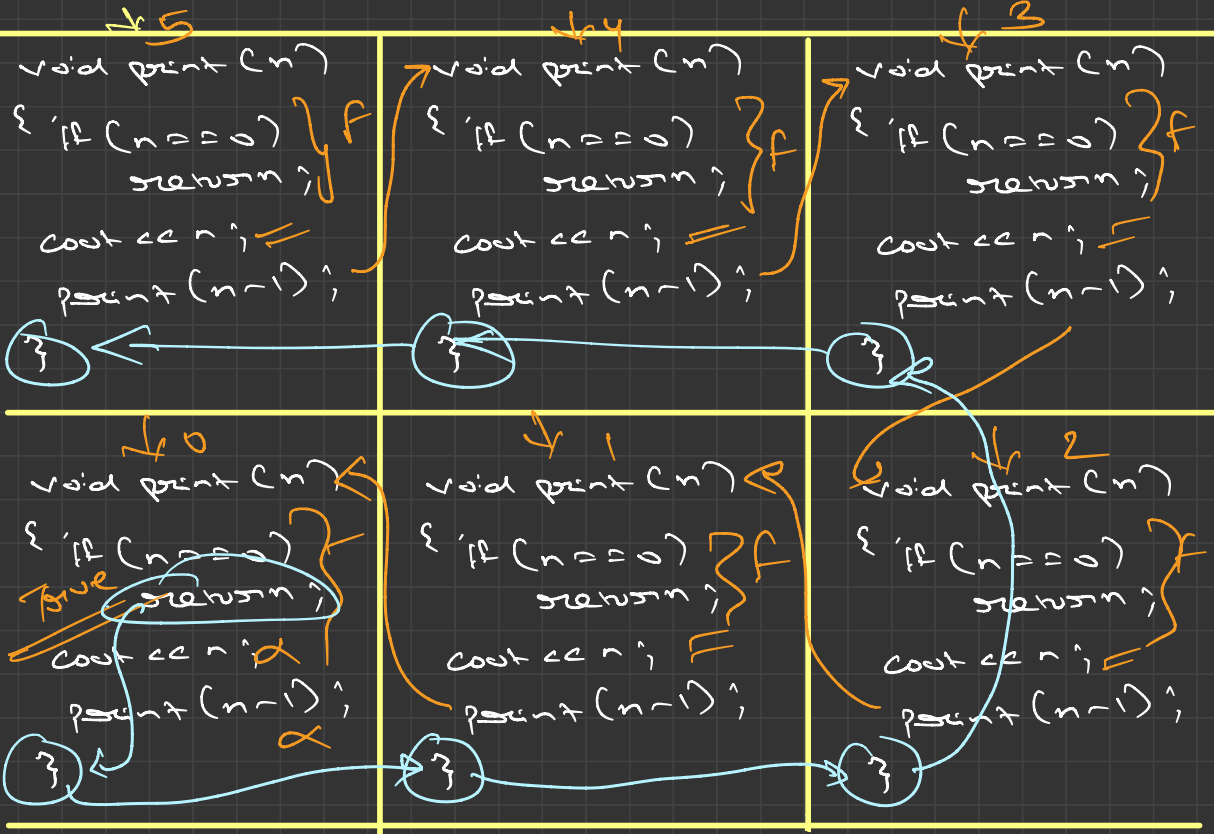
```
int main() {
int n;
cin >> n;
printcount(n);
}
```

```
void printcount(int n){
// Base case
if(n==0)
return;
// processing
cout << n;
// recursive relation
printcount(n-1);
}
```

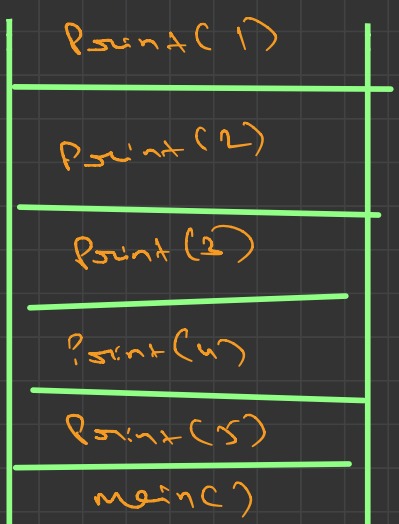
→ end of program

Recursive call stack




5
4
3
0
1
2
void print(n)
{ if (n == 0)
return ;
cout << n ;
print(n-1) ;
}
f
True
α

O/P 5, 4, 3, 2, 1


Print(1)
Print(2)
Print(3)
Print(4)
Print(5)
main()

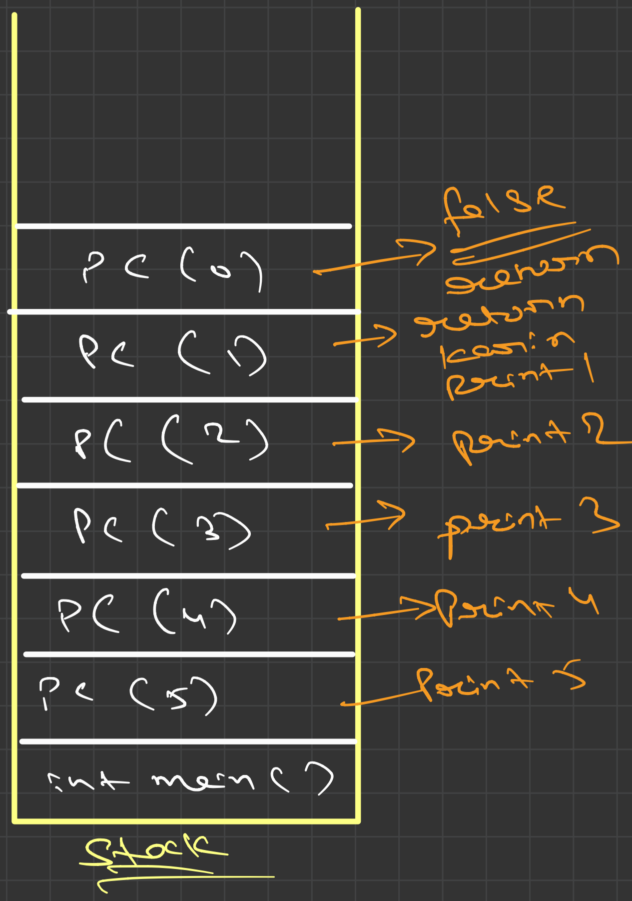
Counting

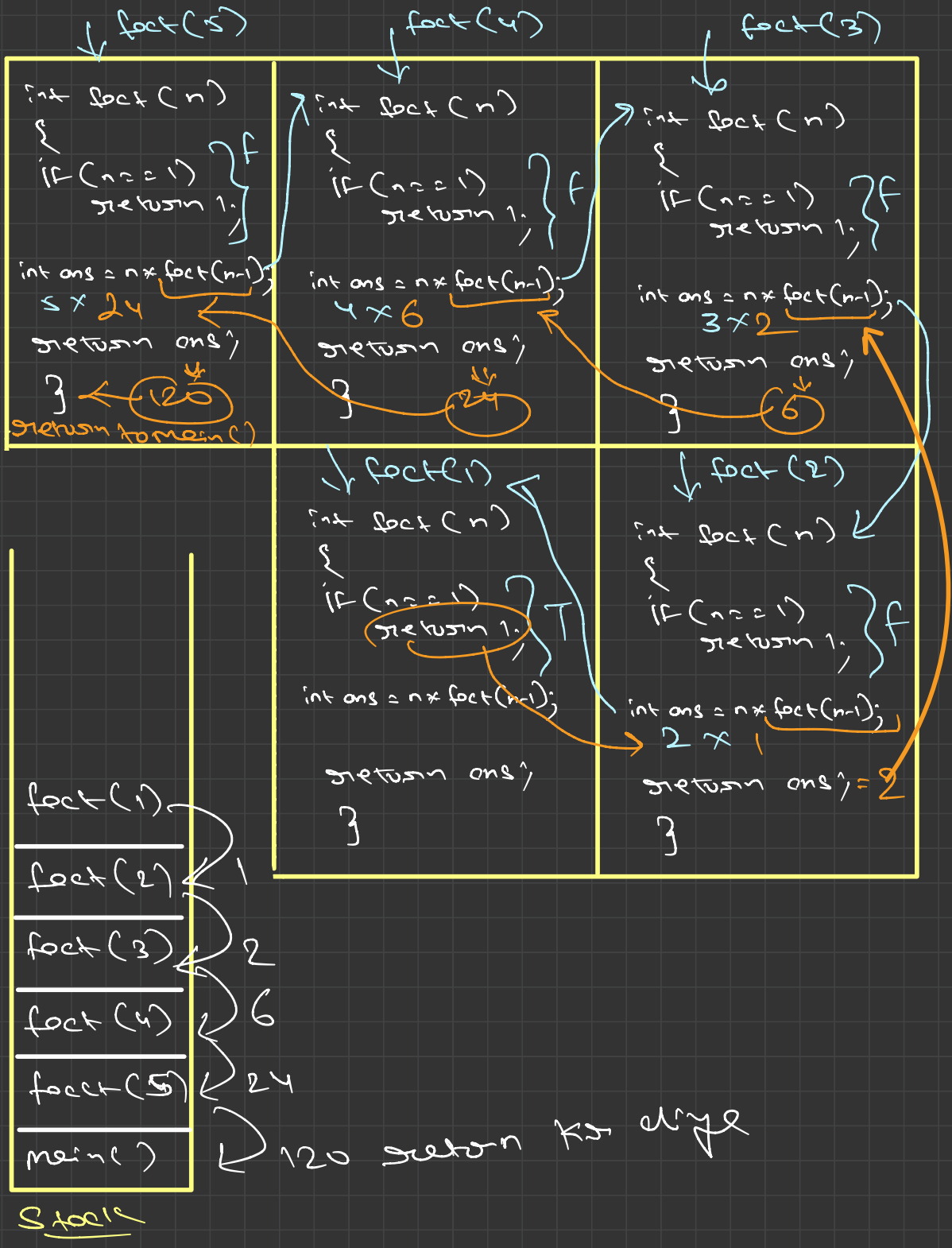
```cpp
int main() {
int n;
cin >> n;
printcount(n);
}
```

```cpp
void printcount(int n){
  // Base case
  if (n==0)
    return;
  // recursive relation
  printcount(n-1);
  // processing
  cout << n;
}
```

→ end of program

return karne ke baad value print karege isliye increasing order mei counting print karege upto n;




fact(5)
fact(4)
fact(3)
int fact(n)
if(n==1) return 1;
int ans = n * fact(n-1);
5 x 24
return ans;
120
return to main()
4 x 6
24
3 x 2
6
fact(1)
fact(2)
2 x 1
2
fact(1)
fact(2)
fact(3)
fact(4)
fact(5)
main()
1
2
6
24
120 return ki style
Stack

Head & Tail recursion

Tail recursion

```
void solve() {
    // B.C.
    // Processing
    // R.R.
}
```

Same kaam hona like head recursion call

Head recursion

```
void solve() {
    // B.C.
    // R.R.
    // Processing
}
```

Same kaam hona se pehle recursion call

```
void print(int n) {
    if (n==0) {
        return;
    }
    cout << n;
    print(n-1);
}
```

print(n-1); → Tail recursion

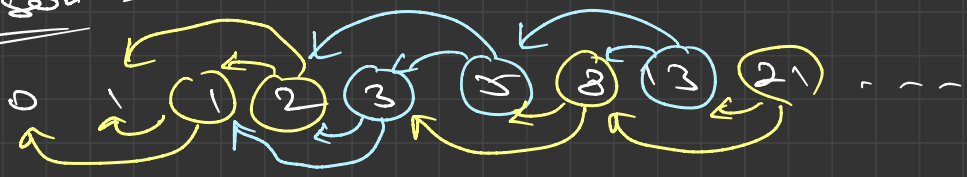
O/P – 5 4 3 2 1

```
void print(int n) {
    if (n==20) {
        return;
    }
    print(n-1);
    cout << n;
}
```

Head recursion

O/P – 1 2 3 4 5

Fibonacci series



nth term = $(n-1)^{th}$ term + $(n-2)^{th}$ term

$$\underbrace{f(n)}_{\text{B.P.}} = \underbrace{f(n-1)}_{\text{S.P.}} + \underbrace{f(n-2)}_{\text{S.P.}}$$

---

0 1 1 2 3 5 8 13 21 ...

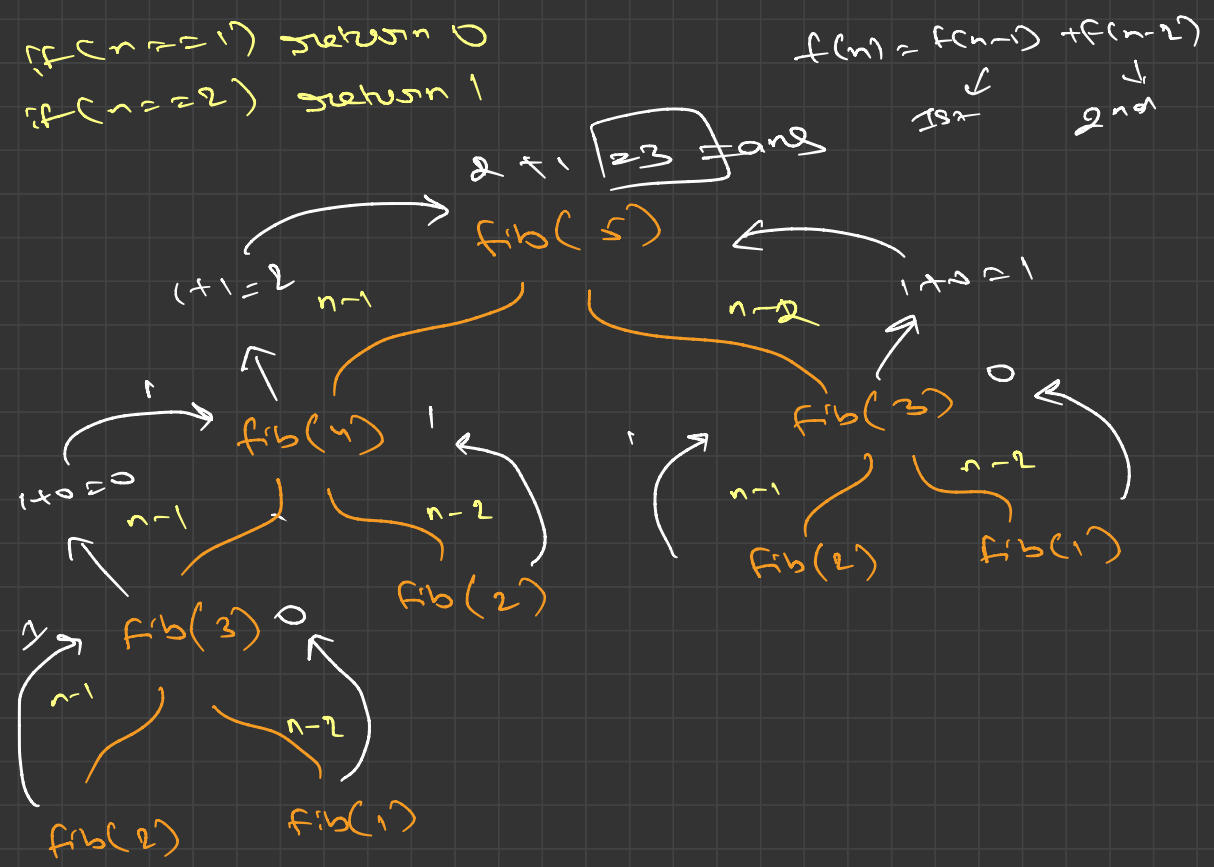
1st term → 1st term

2nd term → 2nd term

1st term = 0
2nd term = 1

→ Isko algo se handle karo

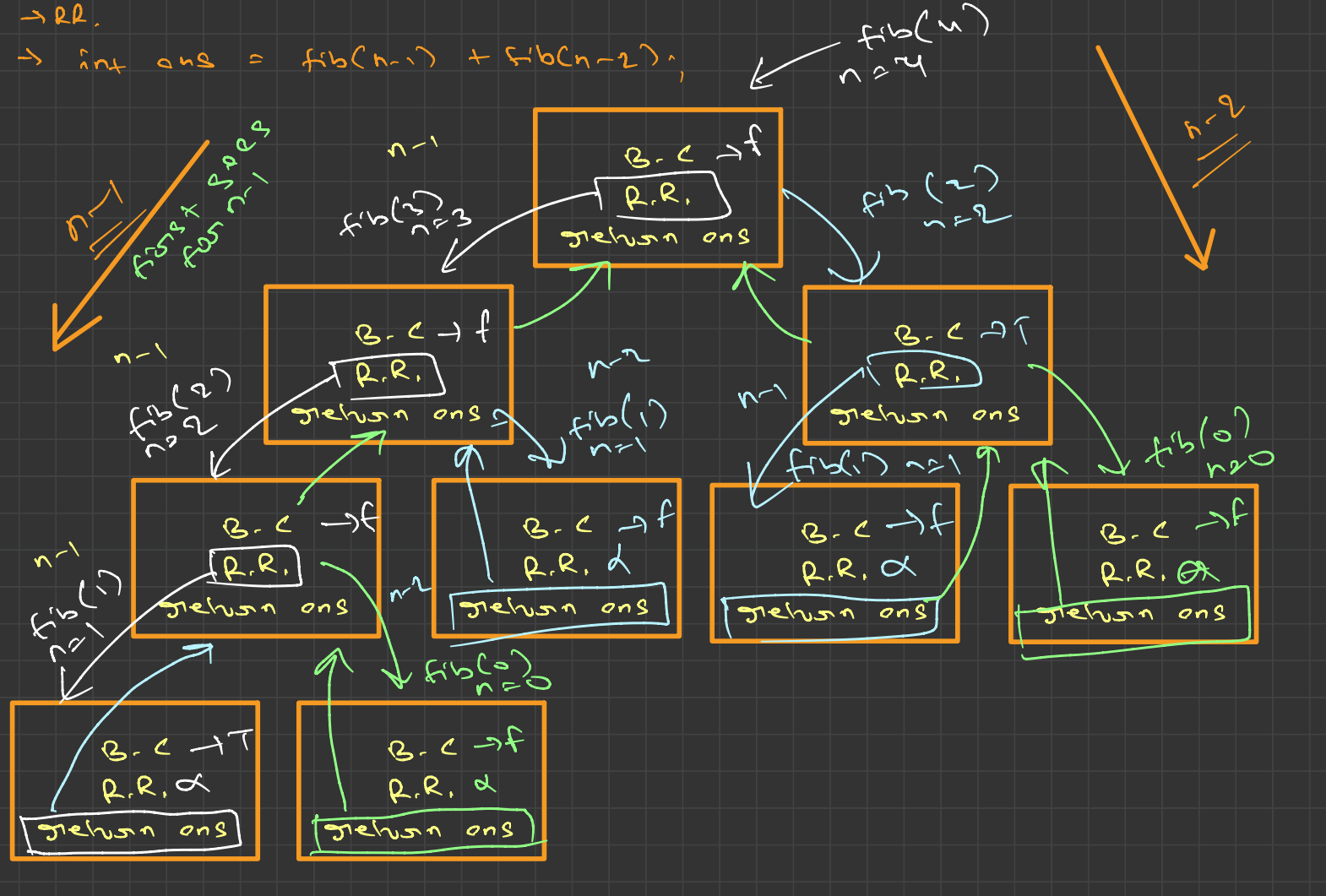
```
int fib(int n){
    if(n==0){
        return 0;
    }
    if(n==1){
        return 1;
    }
    int ans = fib(n-1) + fib(n-2);
    return ans;
}
```

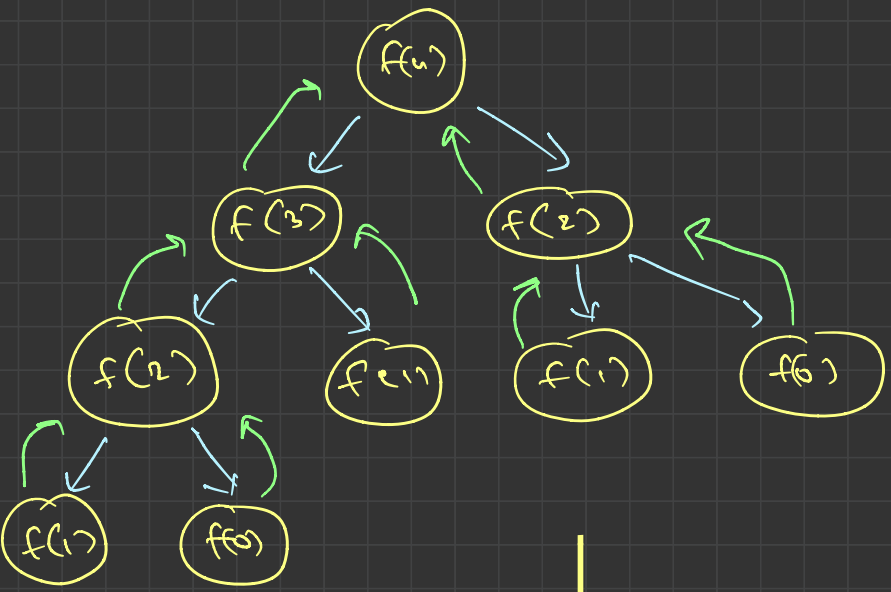

if(n==1) return 0
if(n==2) return 1
f(n) = f(n-1) + f(n-2)
1st
2nd
2+1 =3 ans
fib(5)
1+0=1
1+1=2
n-1
n-2
fib(4)
fib(3)
1+0=0
fib(2)
fib(1)
fib(3)
fib(2)
fib(1)

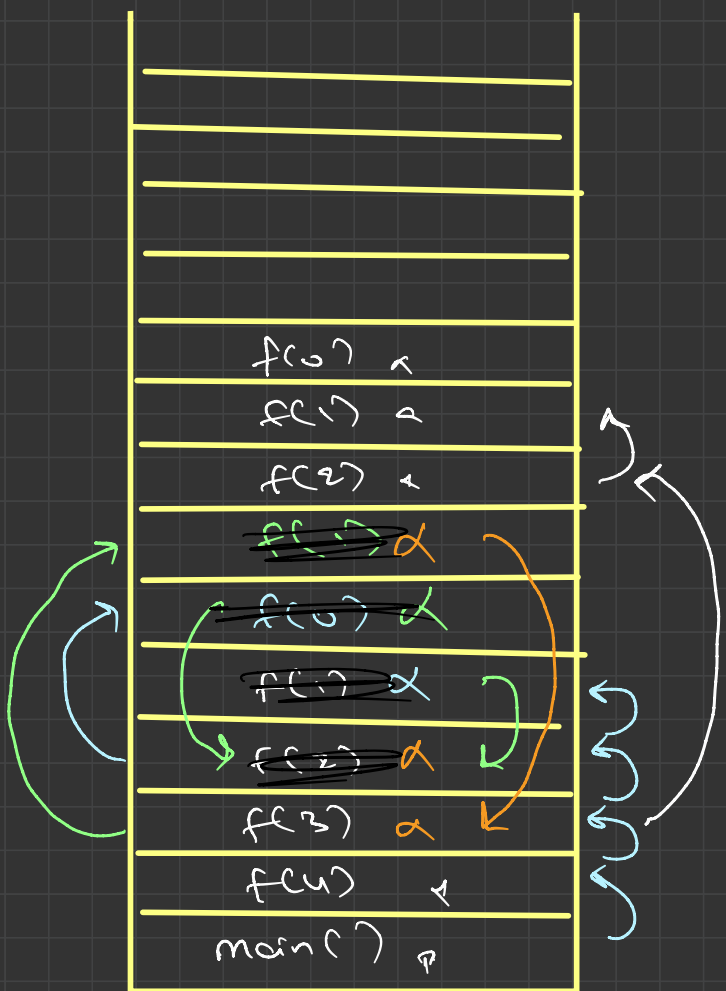
→ BC → if (n==0) return 0; if(n==1) return 1;

→ RR.

→ int ans = fib(n-1) + fib(n-2);




f(4)
f(3)
f(2)
f(2)
f(1)
f(1)
f(0)
f(1)
f(0)


f(0)
f(1)
f(2)
f(3)
f(4)
main()