Week ⇒ 6

Recursion

⇒ What is Recursion

When a function calls itself directly or indirectly.

```
void solve() {
    solve();
}
```

function calling itself

Desi language

Jab ek bigger problem ka sol^n depend karta ho choti and same type ki problem pr.
To wha pe recursion hogi.

eg. To find value of 2^5.
Bigger problem = 2^5

Smaller problem = 2^4

$$\boxed{2^5 = 2 \times 2^4}$$

In general

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

Scanned by TapScanner

CamScanner

eg. Factorial

Bigger problem = 5!

choti problem = 5×4!

$5! = 5 \times 4!$

BP SP

eg. Counting (Reverse)

n=5

↳ 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)

Bigger Problem Smaller problem

Thus.

```
void cal(){
    cal();
}
```

function calling itself. thus it is recursion.

Problem statement

↓

Aisa relation or formula jisme

↓

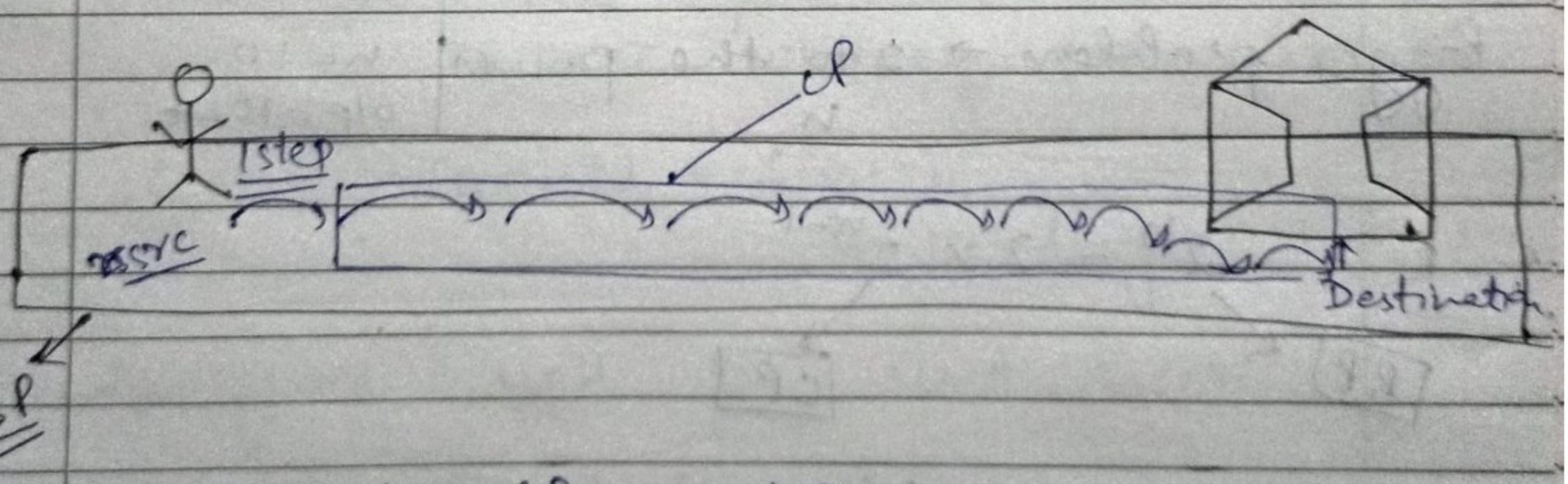
Bigger problem → choti problem pe depend kare

↓

Thus waha recursion use hoga.

Analogy:

Go src to destination



BP = 1 + CP

src to destination = 1 step + 9 steps

Thus here recursion will be used.

Scanned by TapScanner

eg. fibonacci Series

0 1 2 3 5 8 13 21

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

$$f(n) = \underbrace{f(n-1) + f(n-2)}_{CP}$$

BP

Recursion will be used as BP depends on CP.

eg. Problem statement

↳ I/p ⇒ n

↳ o/p ⇒ print 2^n

n = 3
o/p = 8

n = 6
o/p = 64

n = 10
o/p = 1024.

Bigger problem → 2 to the power n.

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

BP CP

Recursion will be used as BP depends on CP.

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

Scanned by TapScanner

CamScanner

⇒ Problem Statement

↳ i/p → n

↳ o/p → n!

Bigger problem → n!

Smaller problem → (n-1)!

$$\boxed{f(n) \rightarrow n \times f(n-1)}$$

Recursion will be used here.

⇒ Problem statement

↳ i/p → n

↳ o/p → reverse counting from n → 1

Bigger problem → f(n) where f(n) means reverse couting from n to 1.

f(n) → n(print) + f(n-1)

↳ printing n-1 to 1.

Recursion will be used here.

f(5) → print(5) then f(4)

4 + f(3)

3 + f(2)

2 + f(1) → 1 + f(0)

Scanned by TapScanner

CamScanner

Code is samajte hain

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

Bigger problem depending on smaller problem

```
int main() {
    int n;
    cin>>n;
    int ans= factorial (n);
    cout << "Answer is :" << ans << endl;
    return 0;
}
```

Components in Recursive code

Mandatory

(1) Base code ⇒ Ye batata hai ki code to stop kaha karna hai.

(2) Recursive relation ⇒ Call recursive
eg. $f(n) = n \times f(n-1)$;

(3) Processing → Optional.

Scanned by TapScanner

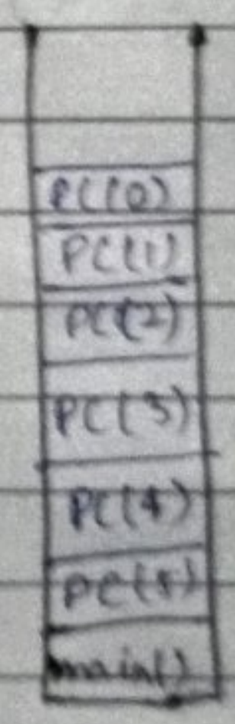
CamScanner

Couting code

```
void printcounting (int n) {
    if (n==0) {
        return;
    }
    cout << n << endl;
    printcounting (n-1);
}
int main() {
    int n;
    cin>> n;
    int ~~ans~~ = printcounting (n);
    cout << ~~ans~~, endl;
    return 0;
}
```

```
Void printcounting (int n)
{
    //B.C
    if (n==0)
        return;

    //processing
    cout<<n<<" ";

    //RR
    printcounting (n-1);
}
```

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



Stack
↓
Last in first out
(LIFO)

O/P 5, 4, 3, 2.

Call Stack

call aayi

print(5)

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

print(4)

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

print(3)

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

print(0)

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

print(1)

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

print(2)

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

Wapas jao

main()
↓
P(5)
↓
P(4)
↓
P(3)
↓
P(2)
↓
P(1)
↓
P(0)

This is called Recursion tree.

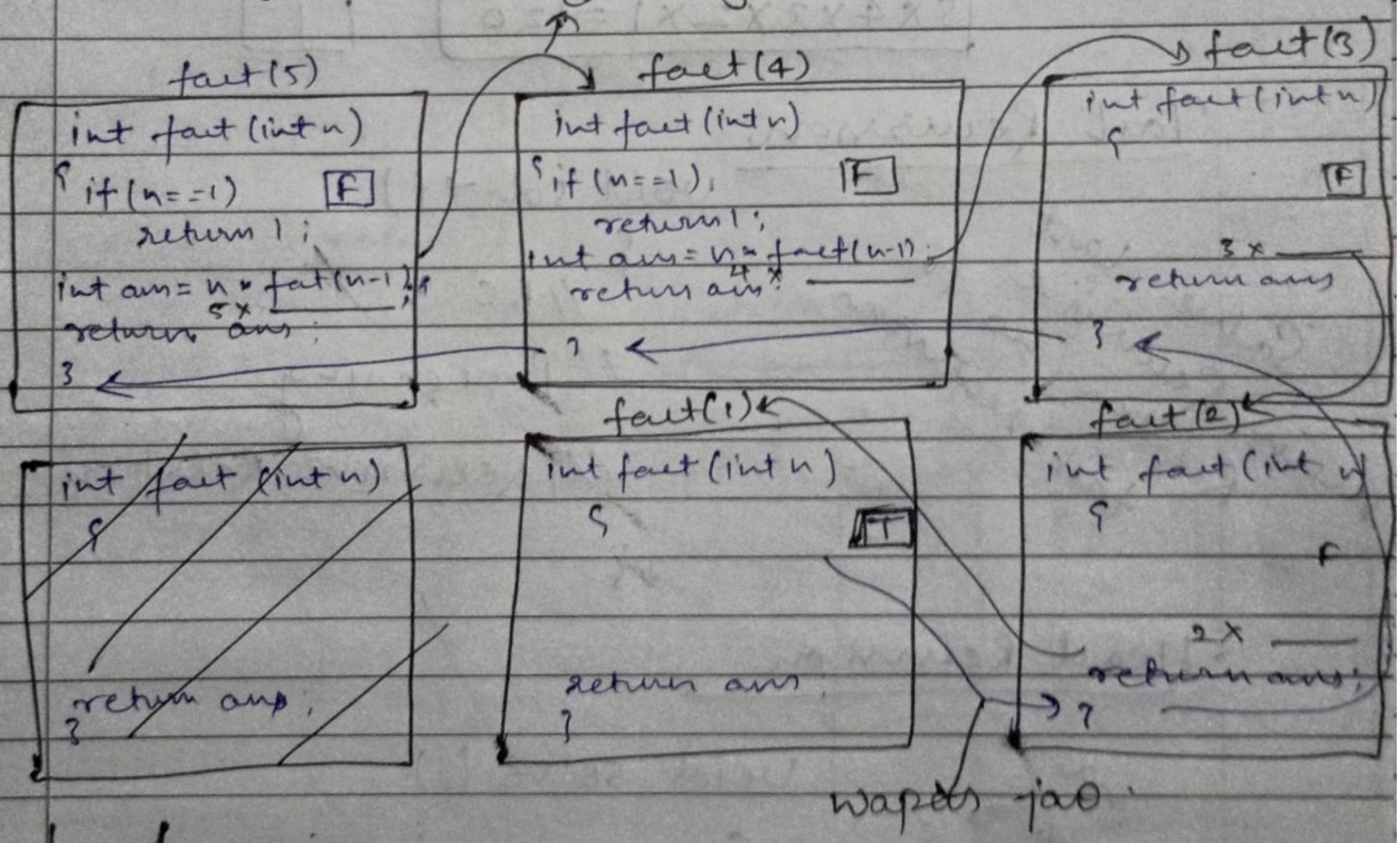
Scanned by TapScanner

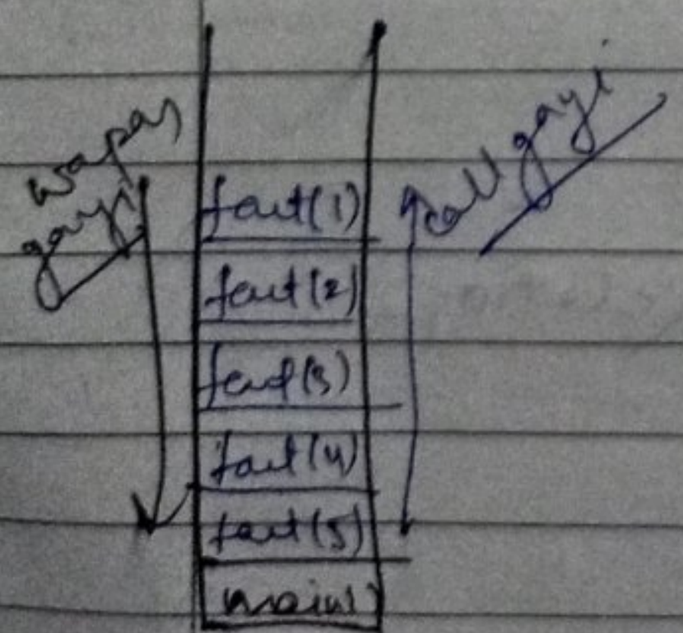
CamScanner

PAGE No
DATE

Factorial

```
int fact (int n)
{
    //BC
    if (n==1)
        return 1;

    int ans = n * fact(n-1);
    return ans;
}
```





Scanned by TapScanner
CamScanner

fact(5)

↓

5 x fact(4)

↓

4 x fact(3)

↓

3 x fact(2)

↓

2 x fact(1)

↓

5x4x3x2x1 = 120 | 1

Tail Recursion

Solve last main Recursive Relation to tail recursion.

```
void solve()
{
    //BC
    // Processing
    // Recursive Relation
}
```

Head Recursion

Let me solve the recursion

```
void solve()
{
    //BC
    //Recursive Relation
    // Processing
}
```

Scanned by TapScanner

CamScanner

eg.

Tail Recursion

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

↓

5 4 3 2 1

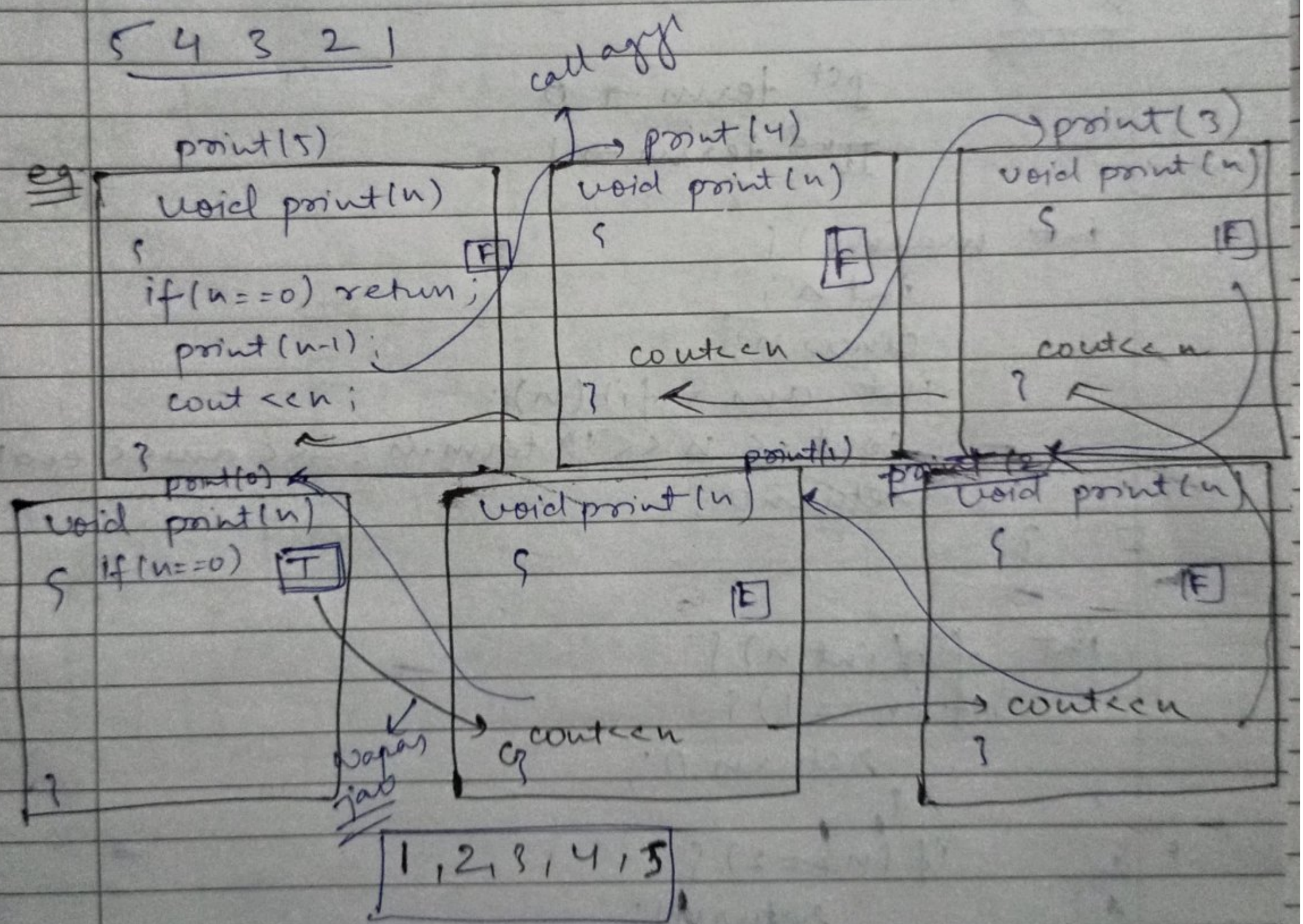
Head Recursion

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

↓

1 2 3 4 5

eg.



Scanned by TapScanner

CamScanner

Fibonacci Series

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

$$\boxed{f(n) = f(n-1) + f(n-2)}$$

Bare case → Usse hi apply karego jisse piche 2 values pehle se ho

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

Ist term → 0

IInd term → 1

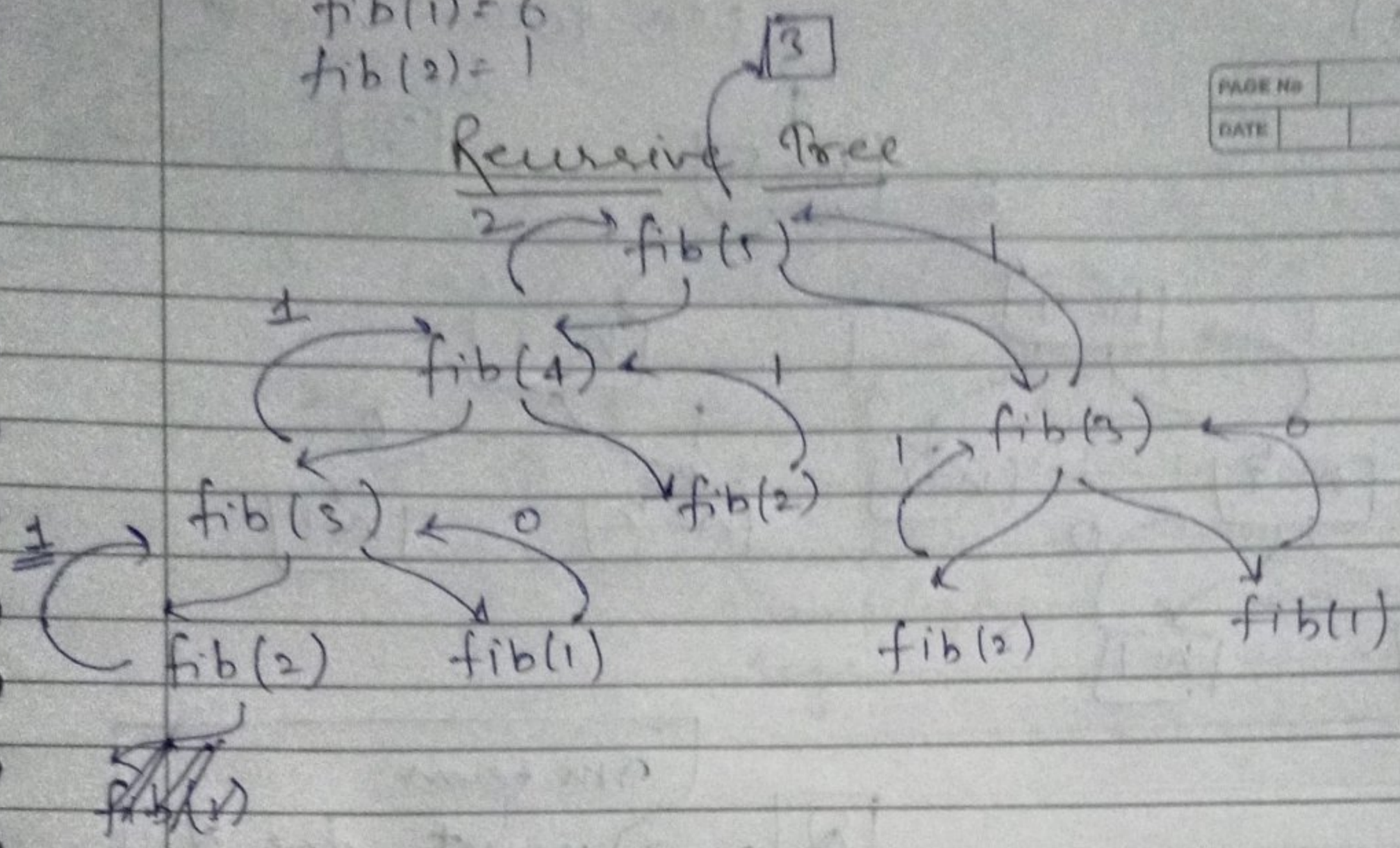
```
int main(){
    int n;
    cin>> n;
    int ans = fib(n);
    cout << n << th term is : " << ans << endl.
    return 0;
}

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

Scanned by TapScanner

CamScanner

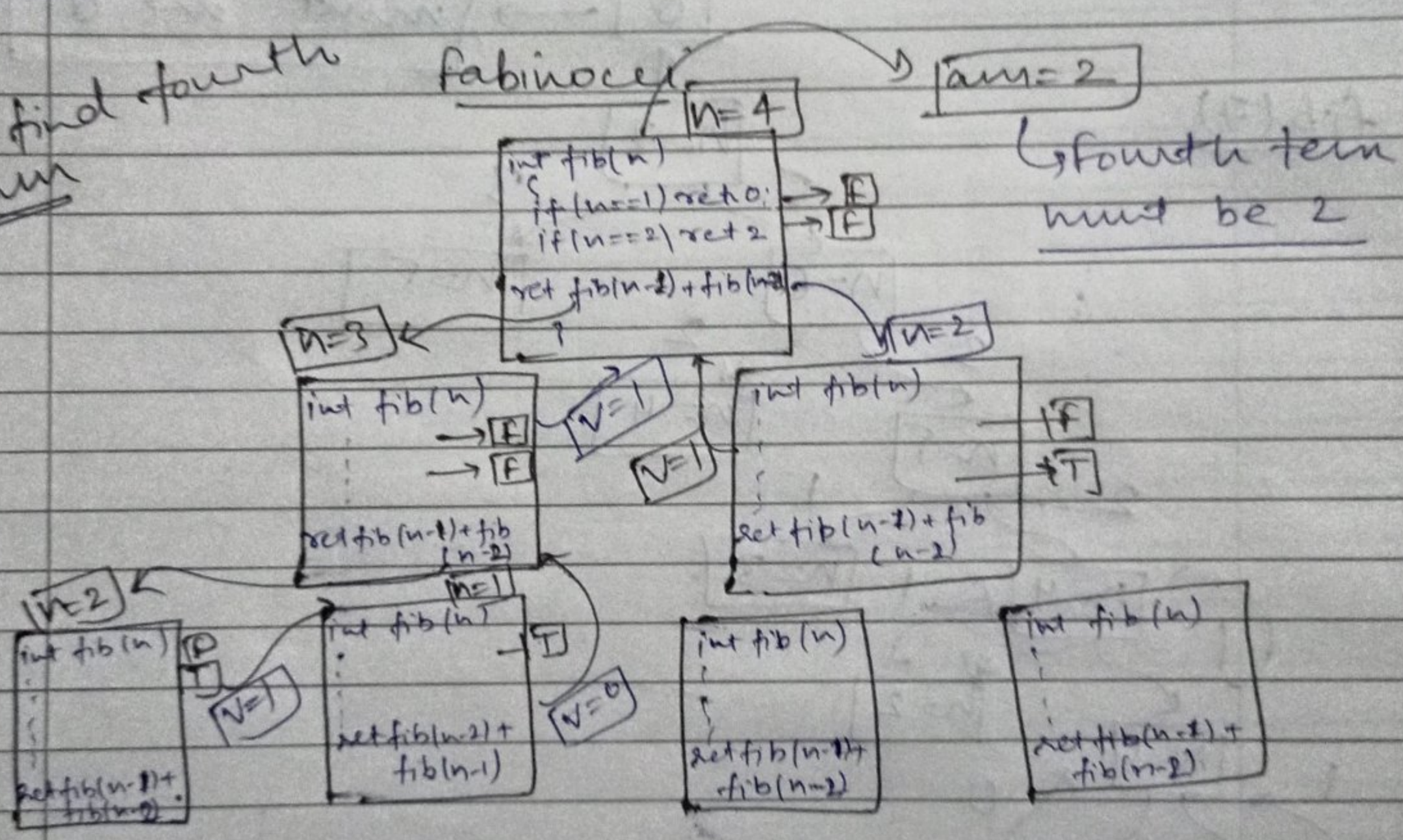
fib(1) = 0
fib(2) = 1

Recursive Tree



To find fourth term

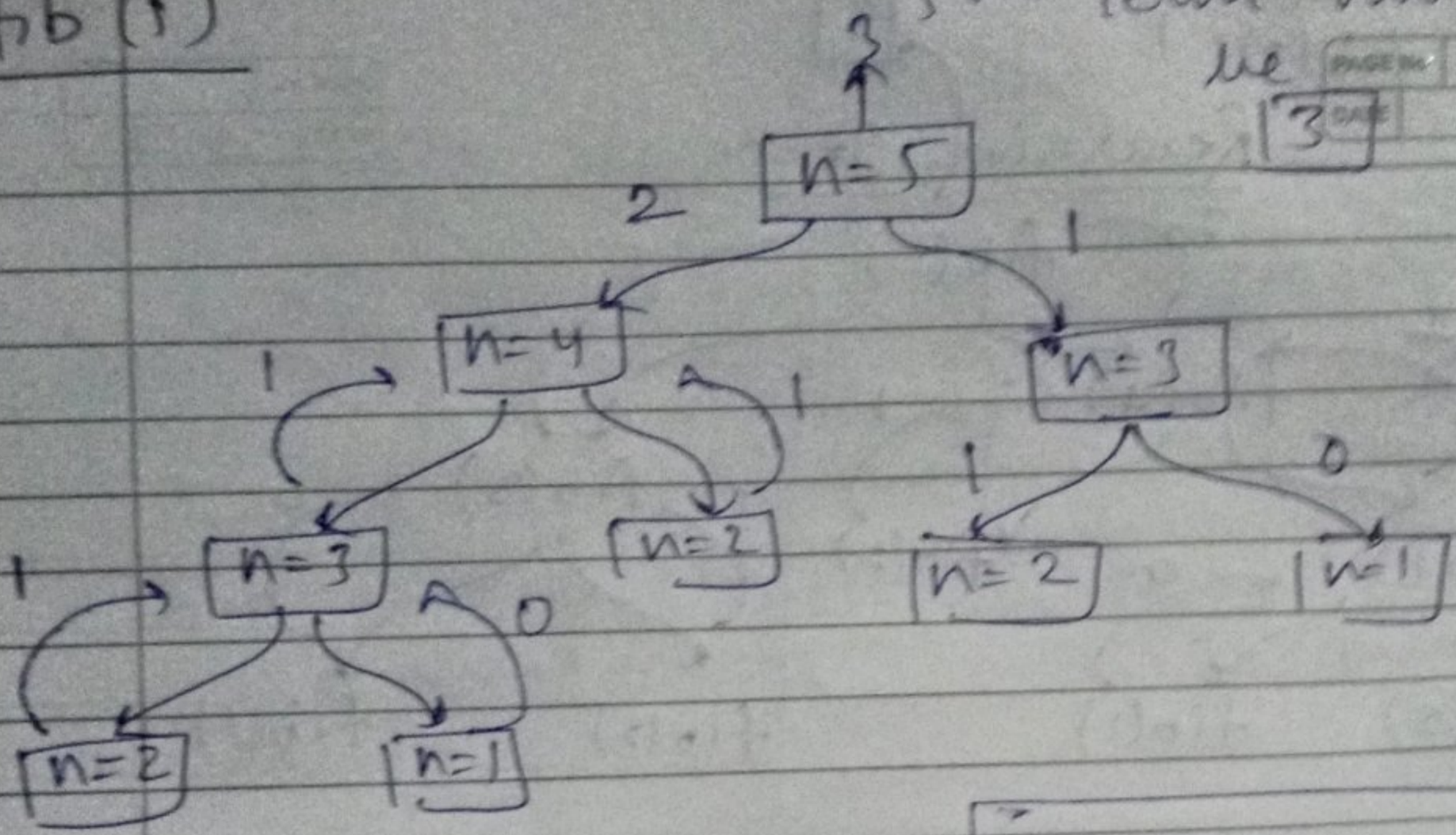
fabinocci



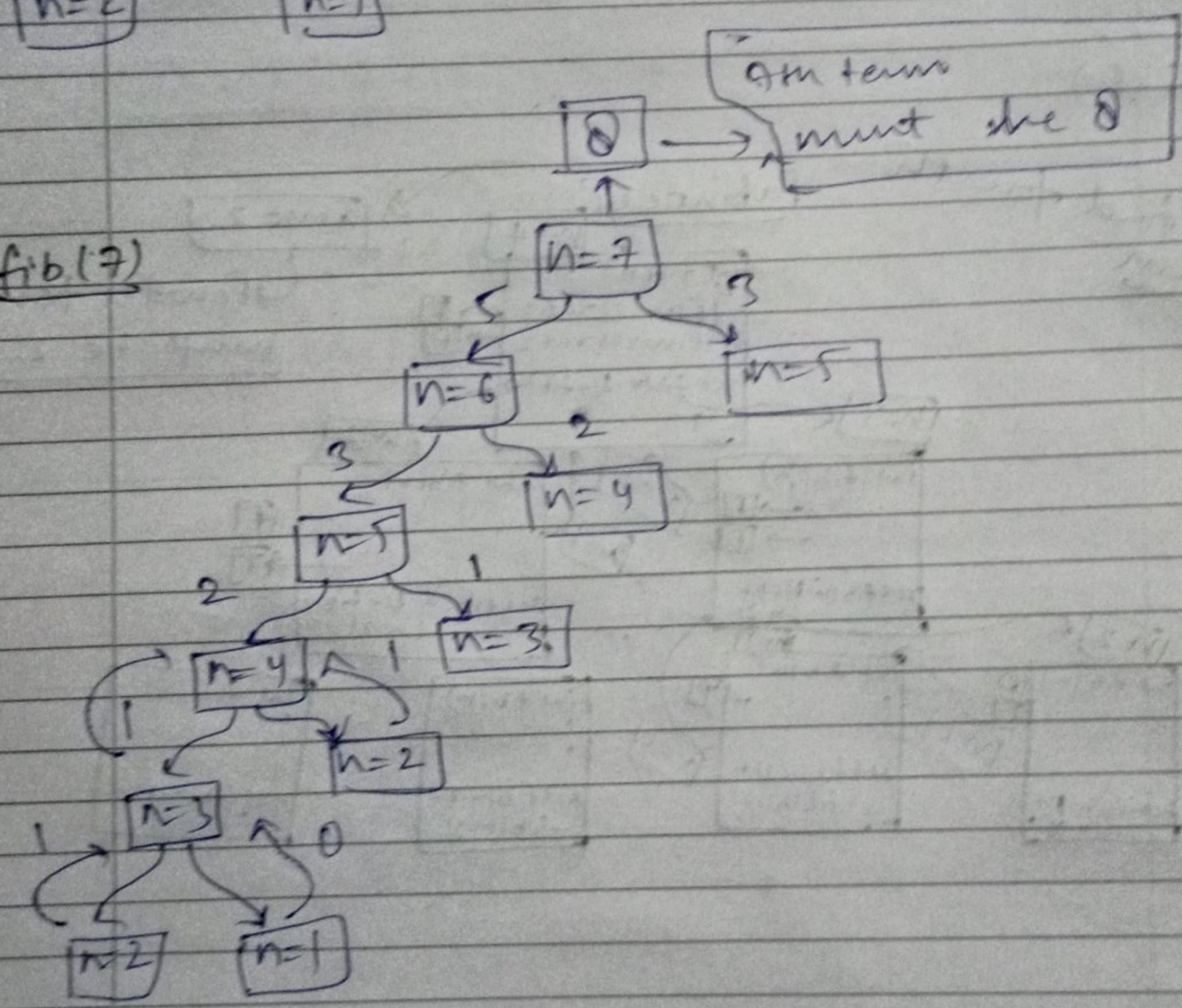
Scanned by TapScanner

CamScanner

fib(5)



fib(7)



Scanned by TapScanner
CamScanner

Magical Line ⇒ Ek case solve kardo
baaki Recursion sambhal lega.

$5! = 5 \times 4!$ { → how, → why, → where }

1 case solve karo — Recursion sambhal lega

Scanned by TapScanner

CamScanner