

## Recursion-2

### ① Climb Stairs →

find no. of ways to reach  $n^{\text{th}}$  stair.

Two steps allowed →

① One stair at a time

② Two stairs at a time.

src

0<sup>th</sup> stair

(n-2)<sup>th</sup> stair

(n-1)<sup>th</sup> stair

n<sup>th</sup> stair

destination

Easy way is that to find out how we reached  $n^{\text{th}}$  stair,

two ways

from  $(n-1)^{\text{th}}$  stair →

from  $(n-2)^{\text{th}}$  stair

$$f(n) = f(n-1) + f(n-2)$$

no. of ways to reach  $n^{\text{th}}$  stair

no. of ways to reach  $(n-1)^{\text{th}}$  stair

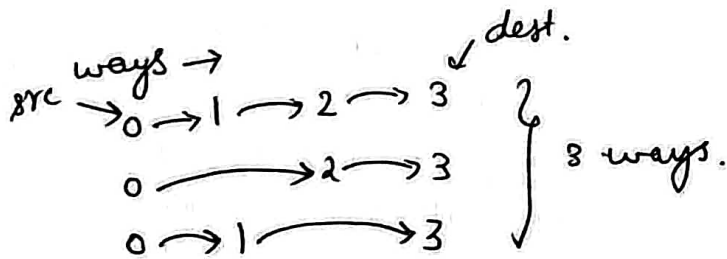
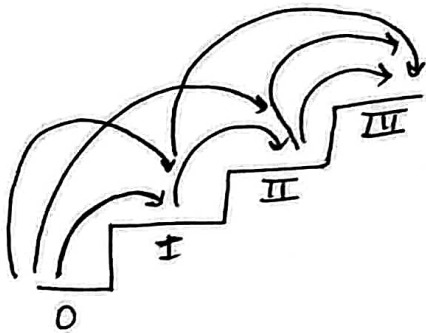
no. of ways to reach  $(n-2)^{\text{th}}$  stair

→ Base case → Termination condition → rukna kaha hai?

0<sup>th</sup> stair पर पहुँचने के कितने तरीके हैं? → 1 → not 0.

1<sup>st</sup> stair पर . . . . . → 1 (0 to 1)

⇒ `if(n==0 || n==1) return 1;` ← Base case.



0<sup>th</sup> stair → 1 way.

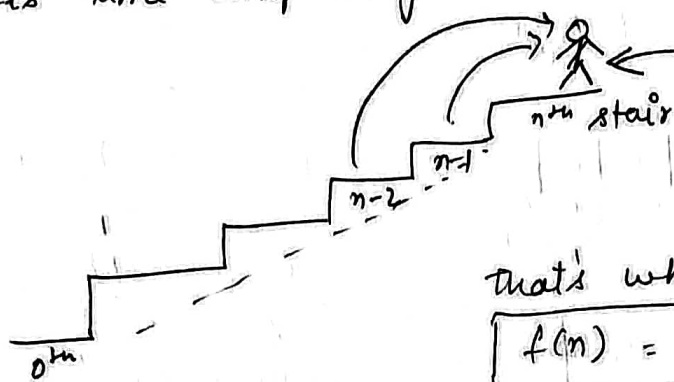
We are already standing at 0<sup>th</sup> stair, but if we say that we have 0 ways to reach 0<sup>th</sup> stair that means we can't reach 0<sup>th</sup> stair ever, but we are already there, so that's why we have 1 way to reach 0<sup>th</sup> stair.

lecture

```
int climbStairs(int n){
    if (n==0 || n==1) return 1; ← Base case
    return climbStairs(n-1) + climbStairs(n-2);
}
```

← R.R.

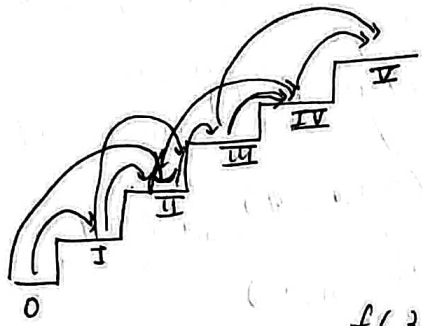
We will end up getting TLE.  
It's time complexity is exponential.



Yha tak mai  
Kaise aayi hungi?  
⇒ Ya to  $(n-1)^{th}$  stair  
se ya  $(n-2)^{th}$  stair se,

that's why

$$f(n) = f(n-1) + f(n-2)$$



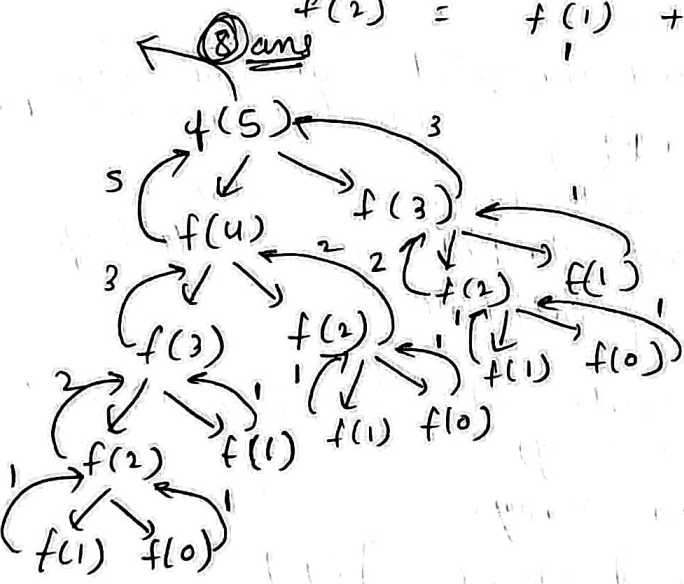
$$f(5) = f(4) + f(3) = 8$$

$$f(4) = f(3) + f(2) = 5$$

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

$$f(2) = f(1) + f(0) = 2$$

$$\begin{matrix} f(1) = 1 \\ f(0) = 1 \end{matrix}$$



This is same  
as fibonacci  
series.

Ques Array an → 

10	20	30	40	50
----	----	----	----	----

print the array.

Iterative method we know:→

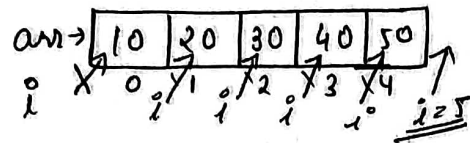
```
for (int i=0; i<n; i++){
    cout << arr[i];
}
```

stopping condition.  
Can we use it as  
a base case in  
recursion?

Recursive →

```
void print(int arr[], int n, int i){
```

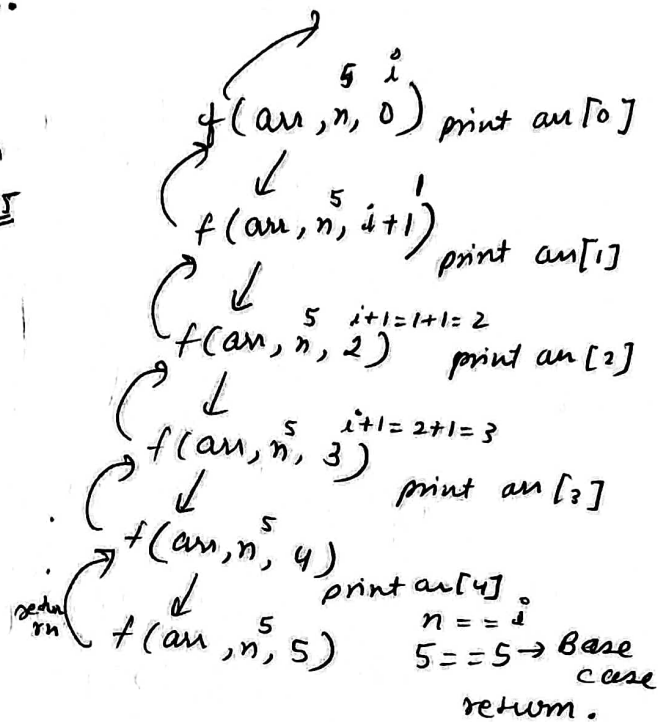
1 case solve Kardiya → if ( $i \leq n$ ) { ← base case  
 return; ← print the element in  $i^{\text{th}}$  index.  
 print(arr, n,  $i+1$ ); ← call for  $(i+1)^{\text{th}}$  index.  
 } Baki recursion sambhal lega.



O/p → 10 20 30 40 50

→ code without index parameter →

```
void print(int arr[], int n){
  if (n == 0) return;
  cout << arr[0] << " ";
  print(arr, n);
}
```

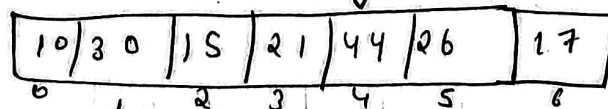


In the func writing post increment will give error.  
 (means  $i++$  in place of  $i+1$ .)

It is because again and again  $i$ 's value as 0 will be passed, and we are stucked in infinite loop.  
 pre increment ( $++i$ ) works fine.

H.W → What if we wrote cout after recursive relation?  
 (processing)

Ques → Maximum element in an array.



Iterative →

```
int maxi = INT_MIN;
for (int i = 0; i < n; i++) {
  if (arr[i] > maxi)
    maxi = arr[i];
}
return maxi;
```

→ For (int i = 0; i < n; i++) {  
 maxi = max(maxi, arr[i]);  
 }  
 return maxi;

```

void findMax (int n) (int arr[], int n, int i, int&maxi) {
    if (i >= n) → whole array is traversed
        return;
    // EK case solve krna hai
    if (arr[i] > maxi)
        maxi = arr[i];
    // baki recursion sambhal lega.
    findMax(arr, n, i+1, maxi);
}

```

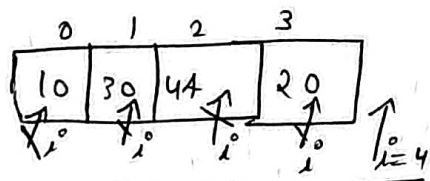
↑ pass by reference.

```

int main() {
    int arr[] = {10, 20, 30, 40, 50, 5, 15};
    int n = 8;
    int i = 0;
    int maxi = INT_MIN;
    findMax(arr, n, i, maxi);
    cout << "Max Element : " << maxi;
}

```

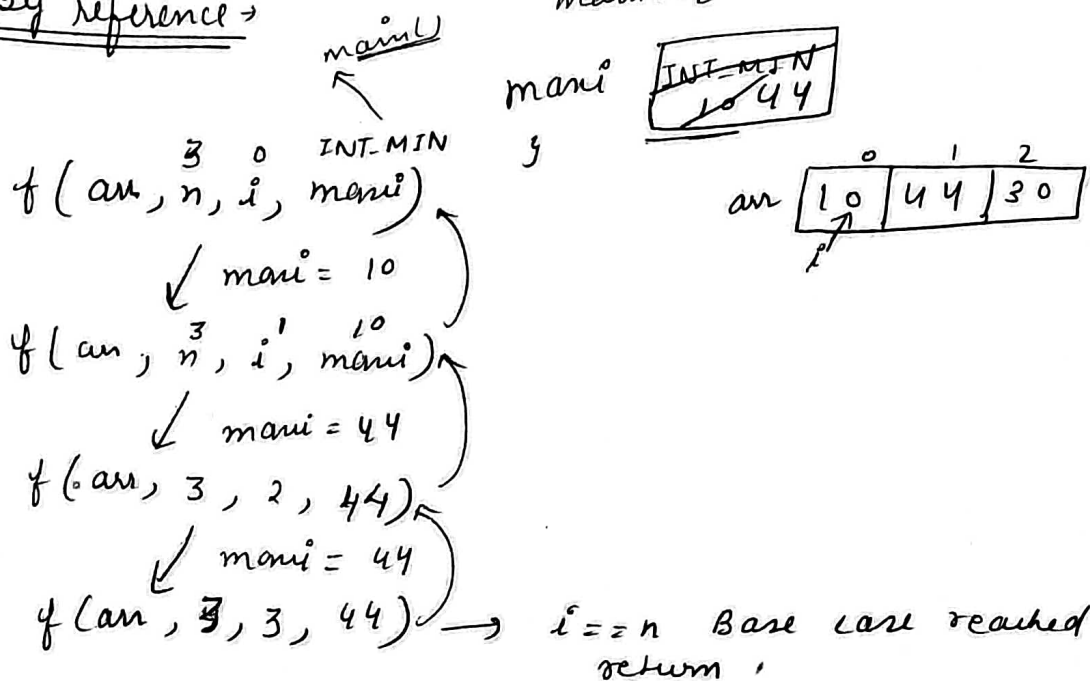
without passing reference variable



$f(arr, 4, 0, INT\_MIN)$   $n == i \rightarrow F$   
 $\downarrow$   $maxi = 10$   $\because 10 > INT\_MIN \rightarrow T$    
 $f(arr, 4, 1, 10)$   $n == i \rightarrow F$   
 $\downarrow$   $maxi = 30$   $\because 30 > 10 \rightarrow T$    
 $f(arr, 4, 2, 30)$   $n == i \rightarrow F$   
 $\downarrow$   $maxi = 44$   $\because 44 > 30 \rightarrow T$    
 $f(arr, 4, 3, 44)$   $n == i \rightarrow F$   
 $\downarrow$   $maxi = 44$   $\because 20 > 44 \rightarrow F$    
 $f(arr, 4, 4, 44)$   $n == i \rightarrow T$  Base case reached.  
 $i == 4$  return

Here each func call has its own maxi and when function returns, the scope of that maxi ends. At last we have INT\_MIN (in main() func.) as maxi, so that's why we are getting wrong ans.

① By reference →



Ques- Find minimum in the array.

```
void findMin (int an[], int n, int i, int &mini) {
    if (i == n)
        return;
    mini = min(mini, an[i]);
    findMin (an, n, i+1, mini);
}
```

Ques- i/p → string str = "lovebabbar"  
key = 'r'

o/p → 'r' is present or not? (using recursion).

l o v e b a b b a r

We will check for first character, baaki recursion sambhal lega.

```
bool checkKey (string str, int i, int n, char key) {
    // base case.
```

```
    if (i >= n)
        return false;
```

```
    // Ek case solve karo.
```

```
    if (str[i] == key)
        return true;
```

```
    // baaki recursion sambhal lega.
    return checkKey (str, i+1, n, key);
}
```

we can send the variable by reference. (Sometimes it's removes TLE).

→ This i+1 can give error (because i is passed as by reference)  
So we will make an var, and pass that.  
int newi = i+1;  
return checkKey (str, newi, n, key);

→ If we have to print the index.  
Then (we have to make minor changes)

```
int checkkey( - - - - - ) {
    if (i >= n)
        return -1;
    if (str[i] == key)
        return i;
    - - - - -
}
```

→ And if we have print all the occurrence of a character.  
Then,

```
void checkkey( - - - - - ) {
    if (i >= n)
        return;
    if (str[i] == key)
        cout << "Found at: " << i << endl;
    - - - - -
}
```

→ Now what if we don't have to print the indexes  
but we have store them into a datastructure, then,  
void checkkey(string& str, int& n, int& i, <sup>char</sup>& key, ~~int~~ vector<int>& ans) {

Note → Whenever we want to store something  
in a datastructure we have to pass  
that datastructure as pass by reference.

if (i >= n) return; // base case.

if (str[i] == key) {  
 ans.push\_back(i); } // Ek case hum solve  
karenge.

{ int newi = i + 1;  
 checkkey(str, n, ~~newi~~++, key, newi, key, ans);  
 ← Baaki recursion sambhal lega.

→ If we want to count the occurrence of a character  
then we will just pass a counter variable which will be  
increased by 1 if the target (key) is found.

```

    checkKey
void checkKey (string &str, int &n, int &i, int &count, char key) {
    if (i >= n) return;
    if (str[i] == key)
        count++;
    int newi = i+1;
    checkKey (str, n, newi, count, key);
}

```

Ques → i/p → 647  
o/p → print all digit of this no.

loop →

```

647 % 10 → 7
↓ / 10
64 % 10 → 4
↓ / 10
6 % 10 → 6
↓ / 10
0 → stop.

```

↳ Base case in recursion

```

void printdigit (int n) {
    if (n == 0)
        return;

    int digit = n % 10; } 1 case main solve karungi
    cout << digit << " ";
    int newValueofn = n / 10;
    printdigit (newValueofn); } ← baaki recursion sambhal lega.
}

```

i/p → 647

o/p → 7 4 6 (reverse order)

To print in original order.

```

void printdigit (int n) {
    - -

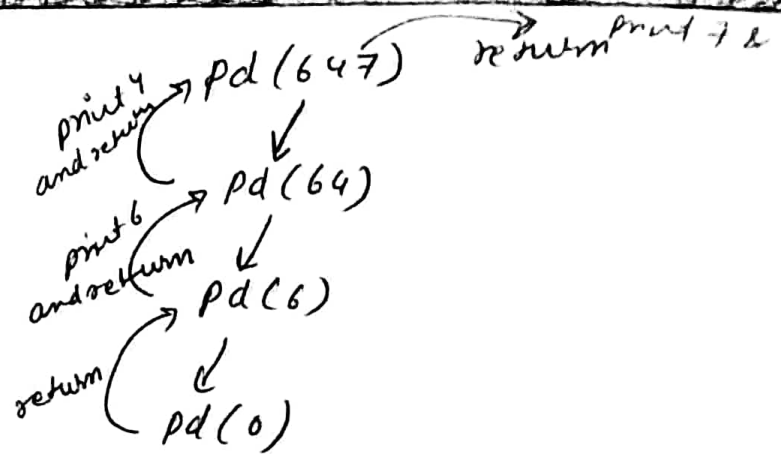
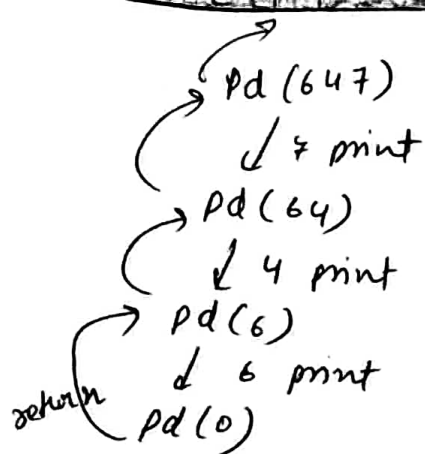
```

```

    int newValueofn = n / 10;
    printdigit (newValueofn); printdigit (n % 10);
    int digit = n % 10;
    cout << digit << " ";
}

```





This code doesn't work for  $n=0$   
 so we will handle this case explicitly.

```

if (n == 0)
    cout << 0 << endl;
  
```

① i/p  $n = 0647$   
 o/p  $\rightarrow 4 \ 2 \ 3$

It is converting 0647 into 423. why? Find out.