Dt 1st Jan 2023
Pg

Module - Recursion

Thinking Process

✗ # Traditional Definition → A fⁿ calling itself

✓ # Logical Definition →

When the solⁿ to a bigger problem depends on solⁿ to smaller instances of the same problem

1) Factorial Problem

$n! = n * (n-1) * (n-2) * \dots 1$

$n! = n * (n-1)!$

$$\boxed{fact(n) = n * fact(n-1)}$$

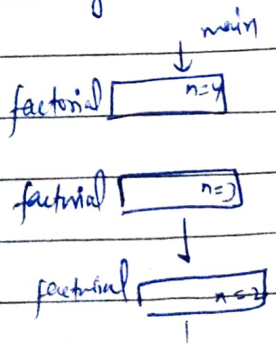
Code

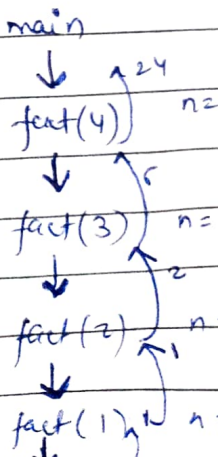
```
int factorial(int n){ if(n<=0) return 1;
                      //n==1
    int ans = factorial(n-1);
    return n*ans;
}
int main(){
    cout << factorial(4) << endl;
}
```

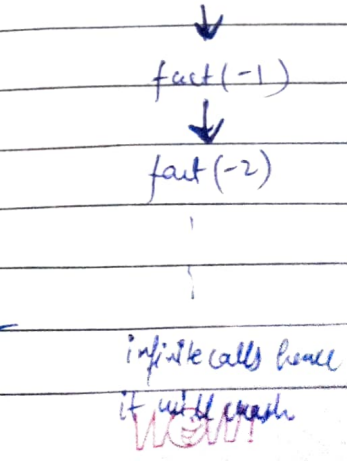
If base case not mentined then Segmentation Fault arises in following scenarios -

→ If array size is 20 and we try to access a 5th element.

→ Not have enough memory.

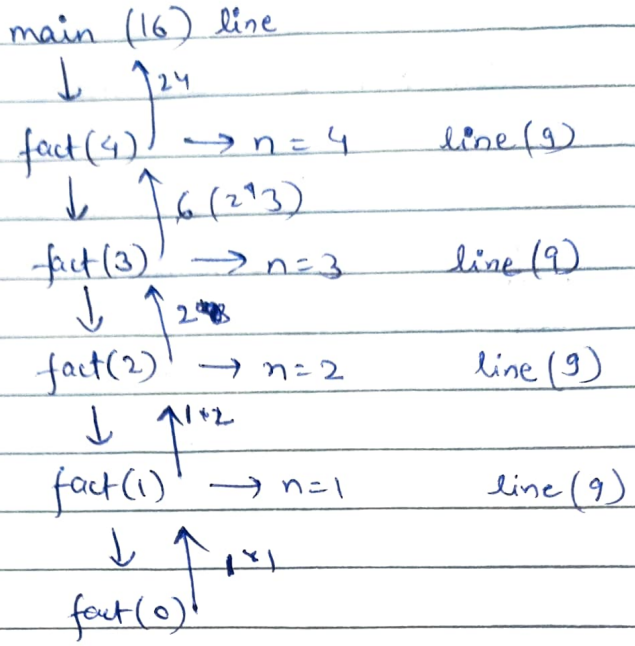




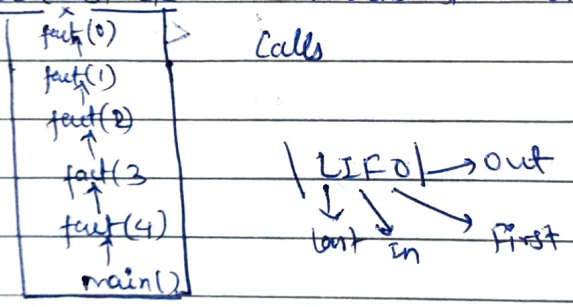


→ Write stopping condition

#



Recursion Call Stack → maintains the order of different Recursive Calls



They are going out of the call stack in the reverse order of their calling.

Relation b/w Recursion & Principle of Mathematical Induction

PMI

f(n) is true for ∀n

① Base Case Prove F(0) or F(1) is true

② Induction Hypothesis → Assume F(k) is true

③ Induction Step → using ② prove F(k+1) is true.

Dt_____
Pg_____

$\Sigma n = n(n+1)/2$

① **Base Case** → $F(0) \rightarrow \Sigma 0 = 0(1)/2 = 0$

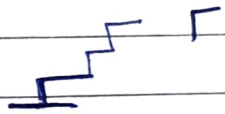
$F(1) \rightarrow \Sigma 1 = 1 = \frac{1(2)}{2} = 1$

② **Induction Hypothesis** → $\Sigma k = \frac{k(k+1)}{2}$

③ **Induction Step** → $\Sigma k+1 = \frac{(k+1)(k+2)}{2}$

$k+1 + \Sigma k = \frac{(k+1)2}{2} + \frac{k(k+1)}{2}$

RHS $= \frac{(k+2)(k+1)}{2}$



We tell base and how to take step so we can take 50 steps.

$K=0$ Proved

I.H → $K \rightarrow 0$ F(0) is true

↓

I.S → $K \leftarrow 1$ F(1) is true

Base Case
Recursive Call
Small Calculation
} Their order can be different.

Power

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

$x = 2$ x^n
$n = 3$ 2^3

ans = 8 ↑ main $2*2*2 = 8$
↓ pow(2,3)
4 ↑ pow(2,2)
ans = 4 ↓
2 ↑ pow(2,1)
ans = 2 ↓
ans = 1 ↑ pow(2,0) WOW!

① Popular Approach for Solving Problems

→ Easy to think and understand

→ Code is shorter

→ Uses Call Stack to store recursive calls

Dt

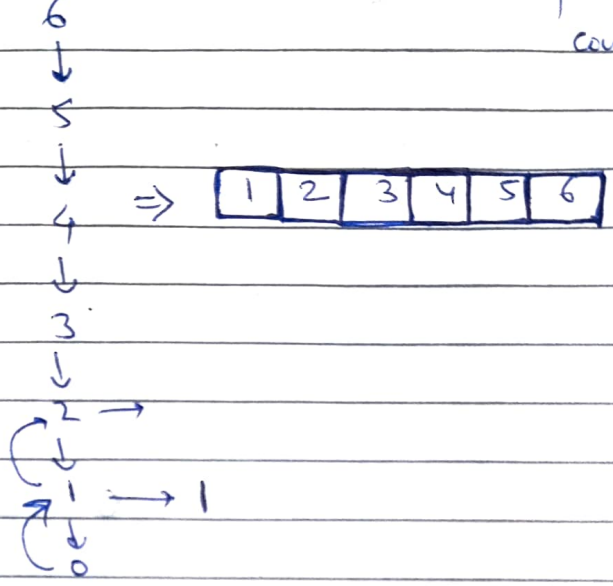
Pg

(2) Print Numbers From 1 to N

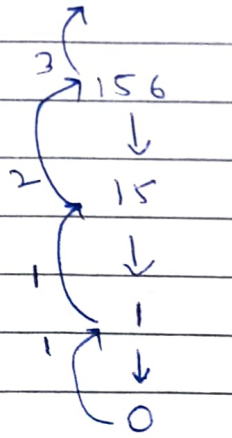
Base case → if(n<=0) return;

printNumbers(n-1);

cout<<n<<endl;

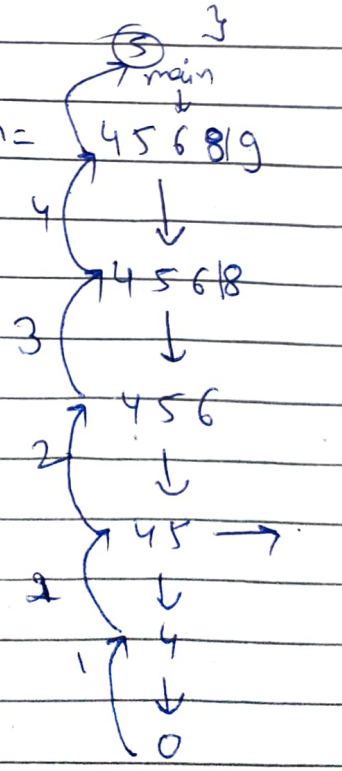


(3) Number of Digits




```
int count(int n){
    if(n<10) return 1;
    int ans = count(n/10);
    return 1+ans;
}
```

① ② ③



WOW!

calls Pg

base → if(n<=0) return;
printNumbers(n-1);
cout << n << endl;

3	4	5	6

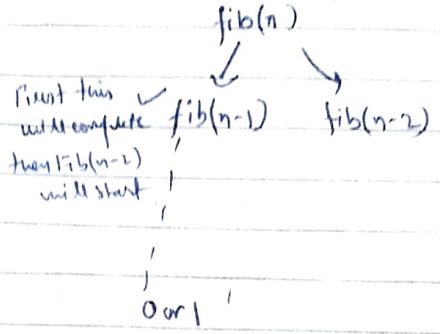
(int n){
(n<10) return 1;
ans = count(n/10);
return 1+ans;

g

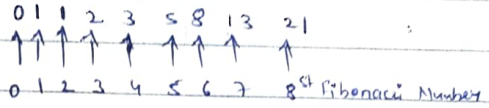
→

Dt
Pg

Both Recursive calls don't happen simultaneously.

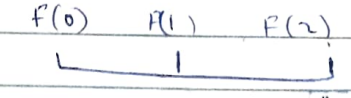


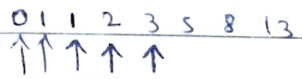
Fibonacci → Fib(n) = Fib(n-1) + Fib(n-2)



Extension of PMI for all numbers less than n.

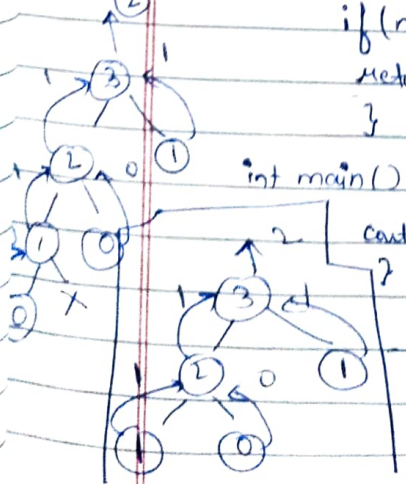
1. Base case → Prove F(0) or F(1) is true
2. Induction Hypothesis → Assume that F(i) is true for ∀ i<=k
3. Induction Step → Prove that F(k+1) is true using step 2

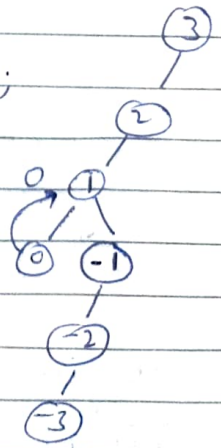




# int fibonacci(int n){
if(n==0||n==1) return n;
return fibonacci(n-1) + fibonacci(n-2);
}

int main(){
cout << fibonacci(4) << endl; → 3
}





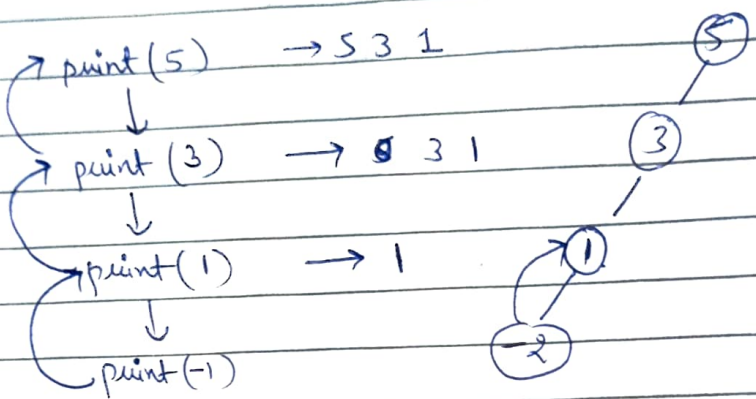
WOW!

Dt
Pg

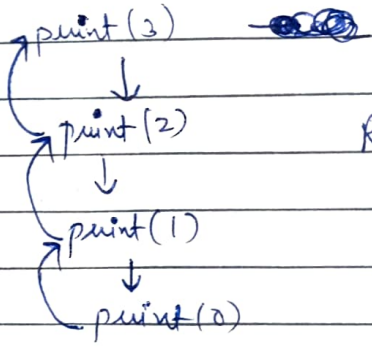
When no base case exists then infinite recursive calls and memory overflow problem occurs hence runtime error occurs

Check

#



#



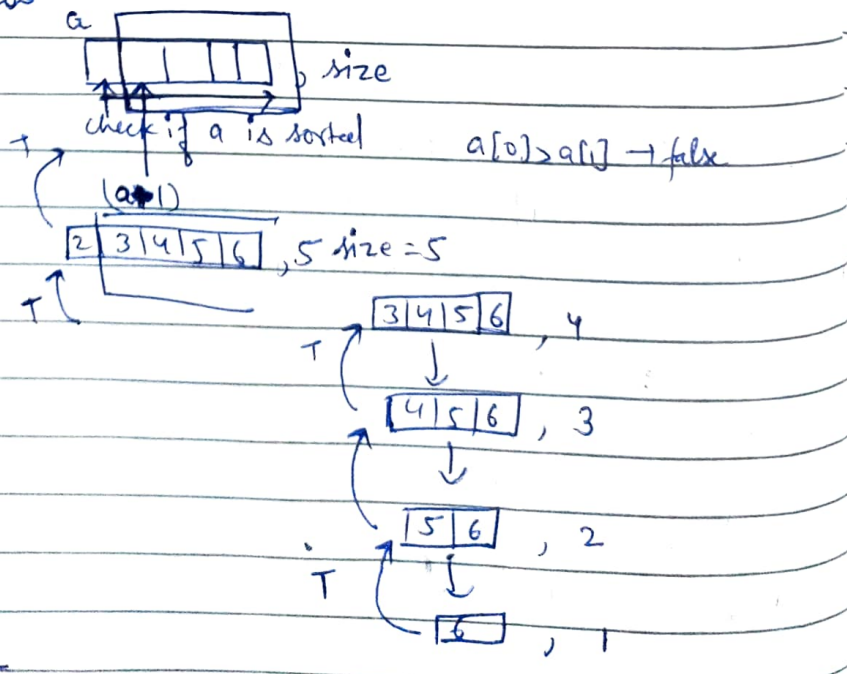
Runtime error as it is
print(n--);
[n will pass & not (n-1)]

#

#

first

Recursion and Arrays

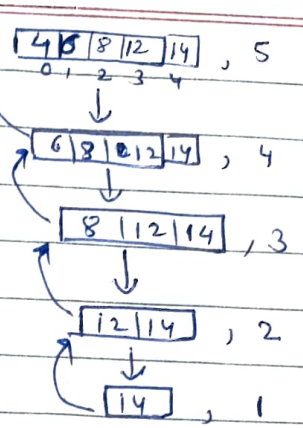


WOW!

Dt
Pg

Check if array is sorted or not

Array → | 4 | 6 | 8 | 12 | 14 |
0 1 2 3 4




```
if (size == 0 || size == 1) return true;
if (arr[0] > arr[1]) return false;
return fun(arr+1, size-1);
```

Array → | 9 | 1 | 8 | 7 | , Sum = 25
0 1 2 3

```
if (size <= 0) return 0;
return Sum(a+1, size-1) + arr[0];
```

Array → | 9 | 8 | 10 | , 8

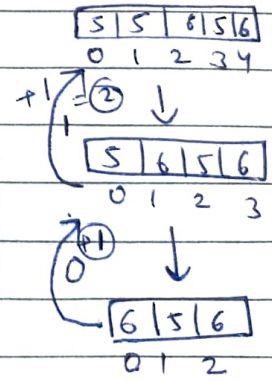
```
if (size <= 0) return false;
if (arr[0] == x) return true;
return ans(arr+1, size-1, x);
```

First Index

array → | 5 | 5 | 6 | 5 | 6 | , size = 5, x = 5, ans = 0
0 1 2 3 4
10 → -1 (not exist)

→ Base case
→ Recursive call
→ Small Calculation

```
if (size <= 0) return -1;
if (arr[0] == x) return 0;
int ans = Rans(arr+1, size-1, x);
return 1+ans; }
```



WOW!

Dt
Pg

Last Index of Number

arr → | 5 | 5 | 6 | 5 | 6 | , size = 5 , x = 5 , ans = 3
0 1 2 3 4 = 6 , ans = 4
= 10 , ans = -1

1) Base case
if (size <= 0) return -1;
2) Recursive Call → (arr+1, size-1, x) → return 1 + ans;
3) Small Calculation

arr → | 5 | 6 | 7 | 8 | size = 4, x = 5, ans = -1
↓
| 6 | 7 | 8 |

All Indices of Number

array → | 5 | 6 | 5 | 5 | 6 | size = 5, x = 5
0 1 2 3 4

output: | 0 | 2 | 3 | → 3

| 6 | 5 | 5 | 6 |
0 1 2 3 — 6 → | 1 | 4 | → 2 → 0

1) Base Case → 0
2) Recursive call → | 6 | 5 | 5 | 6 | → (arr+1, size-1, x, 1 (output)) = ans
3) Small Calculation → i) update each element of output array by 1.
ii) Check at index 0
↳ Yes
Shift the output array

| 1 | 2 | → 2

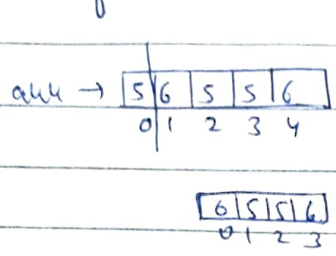
for (0 → 2) {
ans[i] = ans[i]+1
}

WOW!

Dt.

Pg.

All Indices of Number



1) Base Case → 0

2) Recursive Call → (arr+1, size-1, x, output) = ans

3) Small Calculation → Update each element of array (output) by 1

→ Check at Index 0

↓Yes

Shift the output array

Code Base

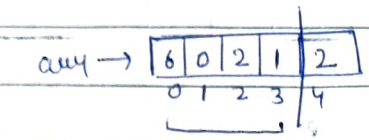
```
int AllIndices(int *arr, int n, int x, int *output){
    if(size<=0) return 0;
    int ans = AllIndices(arr+1, n-1, x, output);
    for(int i=0; i<ans; i++){
        output[i] = output[i]+1; }
    if(arr[0]==x){
        for(int i=ans-1; i>=0; i--){
            output[i+1] = output[i]; }
        output[0]=0;
        ans++; }
    return ans;
}
```

```
int main(){ int n; cin>>n; int *arr = new int[n]; for(int i=0;i<n;i++)cin>>arr[i];
    int *output = new int[n]; int x; cin>>x;
    int ans = AllIndices(arr, n, x, output);
    for(int i=0; i<ans; i++){ cout<<output[i]<<endl;
    }
}
```

WOW!

Dt
Pg.

All Indices of Number → 2nd Approach




```
int AllIndices (int *arr, int n, int x, int *output){
    if(n==0) return 0;
    int ans = AllIndices(arr, n-1, x, output);
    if(arr[size-1]==x){
        output[ans] = size-1;
        ans++;
    }
    return ans;
}
```

(n-1)

① Base Case → size==0 → 0

② Recursive call → (arr, size-1, x, output) → ans

③ Small Calculation → if(arr[size-1]==x){ output[ans] = size-1; ans++; }

return ans;

Here computer programs are fundamental application of mathematics. So first let's try to understand the mathematical logic behind recursion.

Multiplication (Recursive)

m = 3, (n = 5)

if(m==0||n==0) return 0;

3 + 3 + 3 + 3 + 3 → fun(3, 4) + 3.

WOW!

Dt
Pg

Count Zeroes

Base case

```
if(n<9){
if(n==0) return 1; else { return 0; }
int ans = fun(n/10);
int num = n%10;
if(num == 0) return ans+1;
else return ans;
```

Geometric Sum

$$ans = \underbrace{1 + \frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3}}_{} = 1.87500$$

Base → if k==0 return 1;
Recursion → fun(k-1);
Small calculation → ans + (1/(double)pow(2,k));

Sum of digits (Recursive)

27 + 2

9873 2 → 9873 → 29

Check Palindrome (strings)

MADAM

```
bool Palindrome(char arr[]){
    int start=0, end=strlen(arr)-1;
    return Palindrome(arr, start, end);
}

bool Palindrome(char arr[], int start, int end){
    if(start>=end) return true;
    if(arr[start] != arr[end]) return false;
    return Palindrome(arr, start+1, end-1);
}
```

WOW!