Chapter-3

Recursion -1

- Introduction to Recursion

→ $n! = n \times n-1 \times n-2 \times n-3 \times \cdots \times 1$

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

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

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

→ i.e, cutting a bigger Problem into many Small Problems.

CODE

```
#include<iostream>
using namespace std;

int factorial(int n) {
    int SmallOutPut = factorial(n-1);
    return n* smallOutPut;
}

int main() {
    int n;
    Cin >>n;
    int outPut = factorial(n);
    Cout << outPut << endl;
}
```

→ it gives outPut as error: Segmentation fault

→ Segmentation fault Occurs when—

1) we create array of 20 elements, & we are trying to access 25, 30-- or larger elements.

2) when we want memory but there is no more memory available. (This is what happen here)

— Dry Run

all the 'n's are different. when function call for n=4 it waits for output, but no output is there it create another 'n'=3, then again we for output. in continue process at some point no more memory is available.

main
↓
fact(4) n=4
↓
fact(3) n=3
↓
fact(2) n=2
↓
fact(1) n=1
↓
fact(0) n=0
↓
fact(-1) n=-1
⋮ ∞

→ Lets visualize it:
between line 4> & 5>, add this line
Cout << n << endl;

→ Solution, Create it in such way that it auto stop at '0'.

CODE ▷
```
   int factorial(int n) {
2>     if(n==0)
3>     return 1;
4>   // Cout << n << endl;
5>     int SmallOutPut = factorial(n-1);
6>     return n* SmallOutPut;
7> }
```

```
8> int main() {
9>     int n;
10>    cin >> n;
11>    cout << factorial(n) << endl;
12> }
```

4 → factorial(n)

— <u>Dry Run</u>

main (11) line

SO=6 ↑ $\boxed{24}$	fact(4)	n=4	line 5
SO=2 ↑6	fact(3)	n=3	line 5
SO=1 ↑2	fact(2)	n=2	line 5
SmallOutPut(SO)=1 ↑1 ←	fact(1)	n=1	line 5
↑1	fact(0)		

→ As we Come back from fact(0) to fact(4) all the memory created will be auto deleted.

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

bottle

Recursion & PMI

→ Recursion works on the basis of "Principal of Mathamatical Induction".

PMI

$F(n)$ is True $\forall n$

1) Base: Prove $F(0)$ or $F(1)$ is True

2) Induction HyPothesis: $\boxed{\text{Assume}}$ that $F(K)$ is True

3) Induction Step: using step ② Prove that $F(K+1)$ is True

→ Ex $\quad \sum n = \frac{n(n+1)}{2}$

Base Case $\quad F(0) \quad \sum 0 = 0$ L.H.S

R.H.S $\quad \frac{n(n+1)}{2} = 0$ R.H.S

$F(1)\ \sum 1 = 1$ L.H.S

R.H.S $\quad \frac{n(n+1)}{2} = \frac{1 \times 2}{2} = 1$ R.H.S

Induction HyPothesis: $\quad \sum K = \frac{K(K+1)}{2}$

Induction Step: To Prove, $\quad \sum K+1 = \frac{(K+1)(K+2)}{2}$

$$K+1 + \sum K = \frac{(K+1)2}{2} + \frac{K(K+1)}{2}$$

$$= \left(\frac{K+1}{2}\right)(K+2)$$

- Fibonacci Number (0, 1, 1, 2, 3, 5, 8, 13 . . . . .)

- Program to find 'n^th' fibonacci Number

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

→ before that — lets see the Extended form of PMI

Base Case : Prove F(0) or F(1) is True

② IH : Assume f(i) is True $\forall i \leq k$

③ IS : Use ② to Prove f(k+1) is True

→ Now, How Can we use f(i) here,
So, we Proved

F(0) F(1) F(2) F(3) - - - - - -

Using both F(1) & F(0) we Can Prove F(2)

Using F(0)(1)(2) we Can Prove F(3)

CODE

```
int fib(int n) {
    if (n==0) {
        return 0;
    }
    int SmallOutPut1 = fib(n-1);
    int SmallOutPut2 = fib(n-2);
    return SmallOutPut1 + SmallOutPut2;
}
```

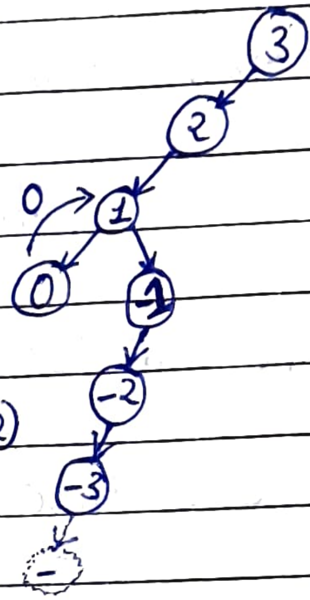
```
int main() {
    Cout << fib(3) << endl;
}
```

OutPut : Segmentation fault

— Lets See why we get Segmentation fault this time.

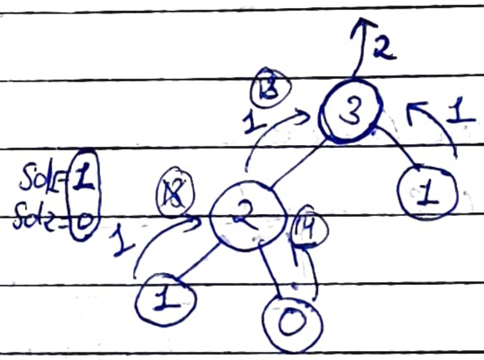
Code Starts with n=3,
from fib(n-1), it Calls for 2
& Similarly 2 Calls for 1
& 1 calls for 0
from Return 0, we get 0
0 will Send (0 output) back to 1

— now 1 moves to next line & calls (n-2)
which is (-1) [1-2=-1]
now, -1 Calls for -2, then -2 for -3
& so on, Hence Result in Segmentation fault



CODE

```
int fib (int n) {
    if (n==0) {
        return 0;
    }
    if (n==1) {
        return 1;
    }
    int SmallOutPut1 = fib(n-1);
    int SmallOutPut2 = fib(n-3);
    return SmallOutPut1 + SmallOutPut2;
}
int main() {
    Cout << fib(3) << endl;
}
```



NOTE => Using induction HyPothesis first write your Code, then only go for dry Run. Never ever think & figure while writing code, Else you might end messed up.

Recursion & Arrays

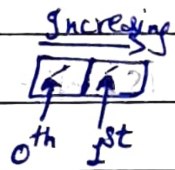
Quest Check weather Array is sorted or not using Recurssion?

Sol^n Lets Just start with an Array , int size

Initially we know,
Array & its Size

Starting with base Case,
Array of Size 0 or 1 is always Sorted
Hence, we Return true.

Now, Array of 2



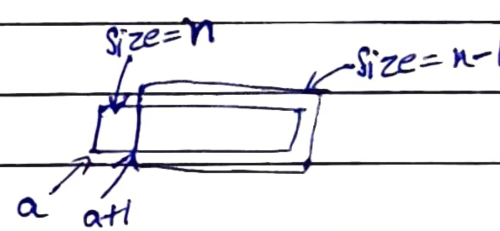
Now, we check
$a[0] > a[1] \longrightarrow$ $\boxed{\text{Return } \underline{false}}$
If true, Hence, Not Sorted

Now, where we are Currently

we need to check this much Part more

i.e, we had done initial step, now automation using recurssion had to done [chote array Ko check Karna hai ab]

for Recurssion on Smaller arrays



Hence, we apply Same function on (a+1, n-1)