Space Complexity

For classic data structure N×M matrix, the space taken is O(N×M)

Auxiliary Space → O(N²)

Recursion

Recursion → Method calling itself is known as recursion.

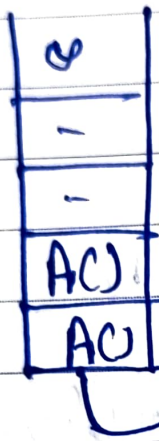
```
A() {          →  A();
  ==                 ↓
  ==                 A();
  ==                 ↓
  A();               A();
}                    ↓
                     A();
                     ↓
                     A();
                     ↓
                     ∞
```

[Recursive Code takes more stack space than non-recursive]

As there is no base condition.

Date: __/__/____

If there is no base condition then function calls itself infinitely & if it calls infinite times then we can say there is chance of stack overflow.



→ Then there will be stack overflow.

Recursion types

- Tail
- Non-Tail
- Indirect
- Nested

① Tail Recursion

When recursion is at last position in method then it is known as tail Recursion.

Benifit → Equivalent can be written in non-recursive form.

```
AC() {
  —
  —
  —
  —
AC();
}
```

Example :-
```
Method (int n) {
    if (n == 0)
        return;
    else
        print (n);
    return Method (n-1);
```

Dry Run :-

TR(3) → 3
↓
TR(2) → 2 → Print
↓
TR(1) → 1
↓
TR(0) → Break Condition

② Non-Tail Recursion

```
Me (int n) {
    if (n ≤ 1) return;
    A(n-2);
    print (n);
    A(n-1);
    print (n-2);
}
```

Date: ___/___/___

③ Indirect Recursion

```
A (int n) {
    if(n<=1) return;
    B(n-2)
    print(n);
    B(n-1)
    print(n-2)
}
```

```
B (int n) {
    if (n<=1) return
    print(n-3)
    A(n-1)
    A(n-2)
}
```

④ Nested Recursion

$$A(m,n) \begin{cases} n+1, & \text{if } m=0 \\ A(m-1,1) & \text{if } n=0 \\ A(m-1, A(m, n-1)), & \text{otherwise} \end{cases}$$

* Principal of Mathematical Induction

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Let, equation is true for $n=k$.

Prove that eq. holds true for $n = k+1$

Prove for $n = 1$

So, we assume $n = k$ and prove for $n = k+1$ as well as for $n = 1$ so our eq. is true for all n Natural Numbers.

To Prove $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Assume $n = k$

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

Prove for $n = k+1$

$$\sum_{i=1}^{k+1} i = \frac{(k+1)(k+1+1)}{2} = \frac{(k+1)(k+2)}{2}$$

LHS

$$\sum_{i=1}^{k+1} i = \underbrace{1+2+3+4+\ldots+k}+(k+1)$$

$$\sum_{i=1}^{k} i + (k+1)$$

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

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

$$\frac{(k+1)(k+2)}{2} = \text{R.H.S.}$$

We proved for $n=k$ & $n=k+1$

for $n=1$

LHS $\sum_{i=1}^{1} i = 1$

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

LHS = RHS

Date: ___/___/___

Reverse Natural Numbers Printing

PD(5) = 5 4 3 2 1

Faith PD(4) = 4 3 2 1

→ High level thinking

- Establish the expectations
- Establish the faith
- Link expectation & faith.

Expectations = n
n-1
n-2
n-3
n-4
⋮
2
1

Faith
PD(n-1) = n-1
n-2
⋮
2
1

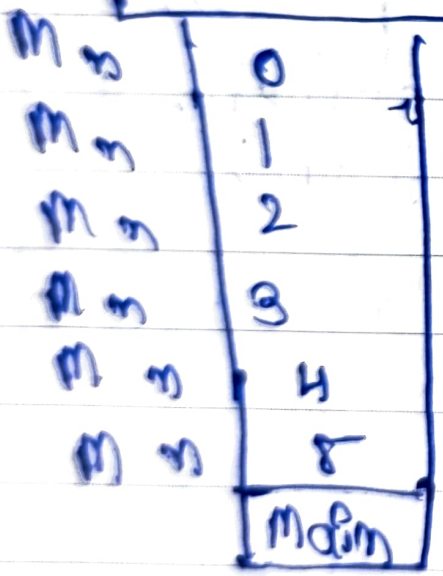
Link both

PD(n) = n
PD(n-1)
1
(1)

On equation ① we printed n by ourself & pass (n-1) to P(n-1) & assuming that it will do it.

① Print Numbers →

```
Public void method (int n)
    print method (n-1);   ——— ①
    SOUT (n)              ——— ②
```



Base Condition (BC)

As soon as base condition hits line ② will execute & then this will be wipe out from stack.

Same will work for all line

* Whenever we use recursion our code divides into two parts part one & part two.

Date: ___/___/___

Code above recursion call is part ① & code below it will be part ②
Part ① fills the stack memory & part ② wipes it out.

② Factorial →

```
public int factorial (int n)
    fn1 = factorial (n-1); - ①
    fn = n * ~~fact~~ fn1 - ②
    return fn; - ③
```