Recursion

- Any function which calls itself is called Recursive.
- Each time the function calls itself with a slightly simpler version of the original problem.
- Recursion step - A recursive method solves a problem by calling a copy of itself to work on a smaller problem.

- 2 types of cases in Recursive algo's
  (1) Base case - In a recursive func at some point it encounters a subtask that it can perform w/o calling itself. This case where the func does not recur is called Base case

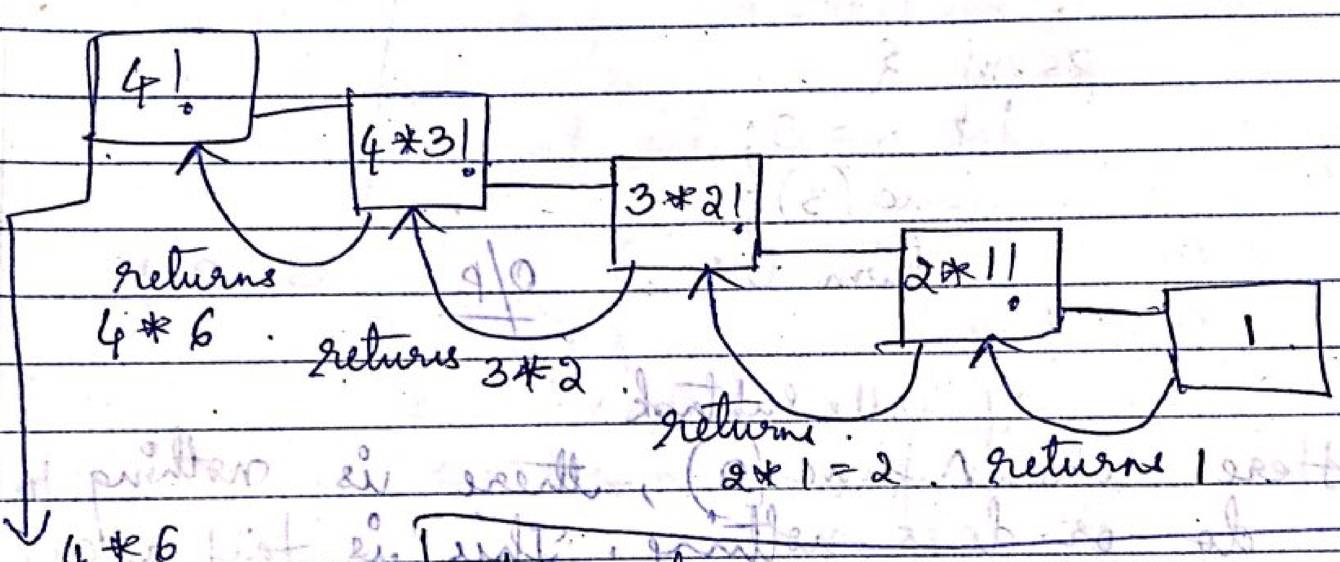
  (2) Recursive case - where the func calls itself to perform a subtask.

- ~~Re~~ RA can be implemented w/o ~~using~~ recursive func calls using Stack, but its not a good idea.

- ∞ recursion the program runs out of mem & results in stack overflow.

Eg/. Factorial => n! => 4!

```
public ~~[illegible]~~ int factorial (int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * ~~[illegible]~~ factorial(n-1);
    }
```



Approaches

1) Top down 2) Bottom up

3) half-half

eg/: binary search, Merge sort

[ie we apply recursion only on half of the data]

Types Recursion

1. Direct recursion

A func calls itself from within itself

This has 4 categories

a) Tail Recursion - If a recur func is calling itself & that recursive call is the last statement in the func

then it is known as tail recursion.
After that call the recursive function performs nothing.

TR can easily be converted to iterative manner.

Eg/.
```
void func (int n) {
      if(n>0) {
        sout (n);
        func(n-1);
      }.

psvm {
   int x = 3;
   func (3);
   return 0;
}
```

O/P   3 2 1

Here when it calls subtask func(0), there is nothing to do or does nothing. This is tail recur.

TC : O(n)
SC : O(n) => stacks takes up space.

The same program written using loop instead of tail recursion

```
Void func (int n) {          TC : O(n)
  while ~~if~~ (n>0) {       => SC : O(1).
      Sout (n);
      ~~while~~ n--
}
```

~~b) Head recursion of a recursive function calling itself & that recur call is the 1st~~

b) Head recursion

- recursive func calling itself
- and that rec call is the first statem. in that func
- The func. doesn't have to process or perform any operation at the time of calling & all operations are done at returning time.

```
Voidfun (int n) {
    if (n>0) {
        func(n-1);
        sout(n);
    }
}

psvm {
    func(3);
    return 0;
}.
```

TC : O(n)

SC : O(n)

O/P : 123

It is usually hard to convert head recursion into iterative (loop) structure.

Convert head recursion into loop for comparison

```
Void func (int n) {
    ~~while~~
    ~~while (n > 0)~~
        ~~func(n-1);~~
  int i = 1
  while (i <= n) {
      sout (i);
      i++;
  }
```

```
PSVM {
    func(3);
}
```

TC : O(n).
SC : O(1).

Linear
~~Tree~~ Recursion ⇒

- RF ~~calls~~ that calls itself only once is called as Linear recursion.

d) Tree Recursion —

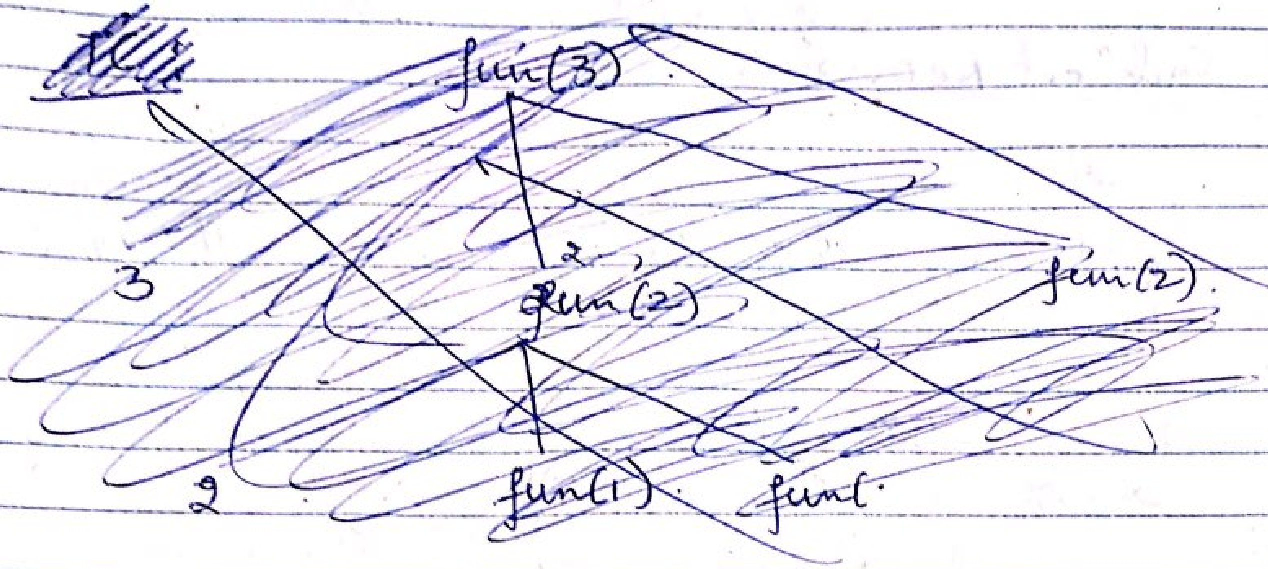
- ~~RF~~ calls itself for more than one time

Tree Recursion.

```
Void func (int n) {
    if (n > 0) {
        sout(n);
        func(n-1);
        func (n-1);
    }
// calls itself twice.
```

Linear recur.

```
Void func(int n) {
    if (n > 0) {
        sout(n);
        func(n-1);
    }
// calls itself once
```



TC : O(2^n).

SC : O(n).

e) Nested Recursion :-

In this, a ~~f~~ recursive function will pass the parameter as a recursive call. That means "recursion inside recursion".

eg.

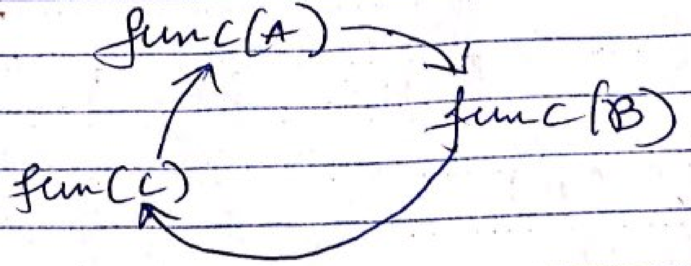
```
int func(int n) {
        if (n > 100) {
            return n-10;
        }
    return func(func(n + 11));
}
```

```
PSVM {
    int r = func(95);
}.
```

o/p : 91

2) Indirect Recursion -

There may be more than one function and they are calling one another in a circular manner.



Eg/.

```
void funcA(int A) {
        if (n > 0) {
            funcB (n-1);
        }
}


Void funcB (int n) {
        if (n>1) {
            funcA (n/2);
        }
}


PSVM {
     funcA (20);
}
```