Types of recursion

(1) **Tail recursion :** (Every call at run time)

Time → O(n)
Space → O(n) } (A)

```
void any (int n) {
    if(n>0) {
        cout << n ;
        any (n-1);   → recursive call at the end
                       of condition
                         ↳ Tail recursion
    }
}
any (3);
```

↳ space : 4 stacks will be created (n+1) (recursion uses stack)

(A) So it is better to modify a tail recursion into a loop as the loop will have space complexity as O(1).

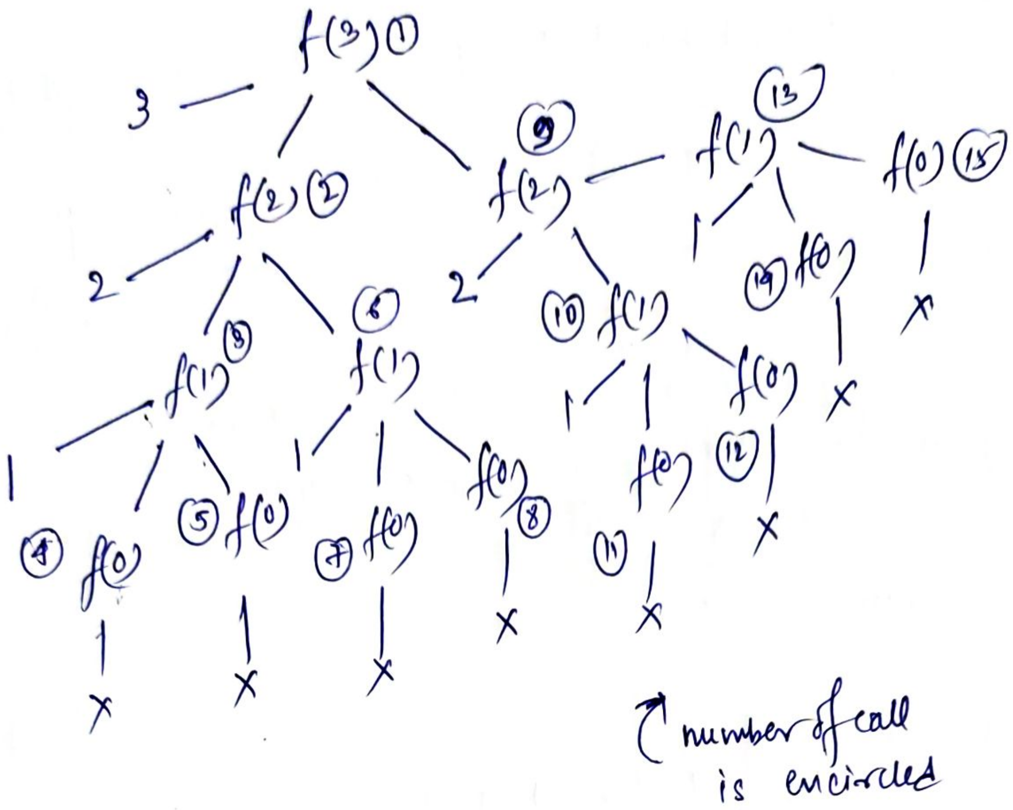
(2) **Head recursion !** (Every call at returning time)

```
void any (int n) {
    if(n>0) {
        any (n-1);   → recursive call
                       at the start
                       of condition
                         ↳ Head
                           recursion
    }
}
```

(3) Tree recursion

```
void something (int n) {
        if(n>0) {
            cout<<n;
            something (n-1);   ] → Calling ~~the~~ the
            something (n-1);   ]   recursive funcn
                                   more than two
        }                          times is c/a
}                                  tree recursion
```

Something () ≈ f ()



O/P → $\xrightarrow{3\ 2\ 1\ 1\ 2\ 1\ 1}$

Time complexity → $1 + 2 + 4 + 8 = 15$

$2^0 + 2^1 + 2^2 + 2^3$ ⟶

$\therefore\ 2^0 + 2^1 + 2^2 + \ldots 2^n = 2^{n+1} - 1$

So, $O(2^n)$

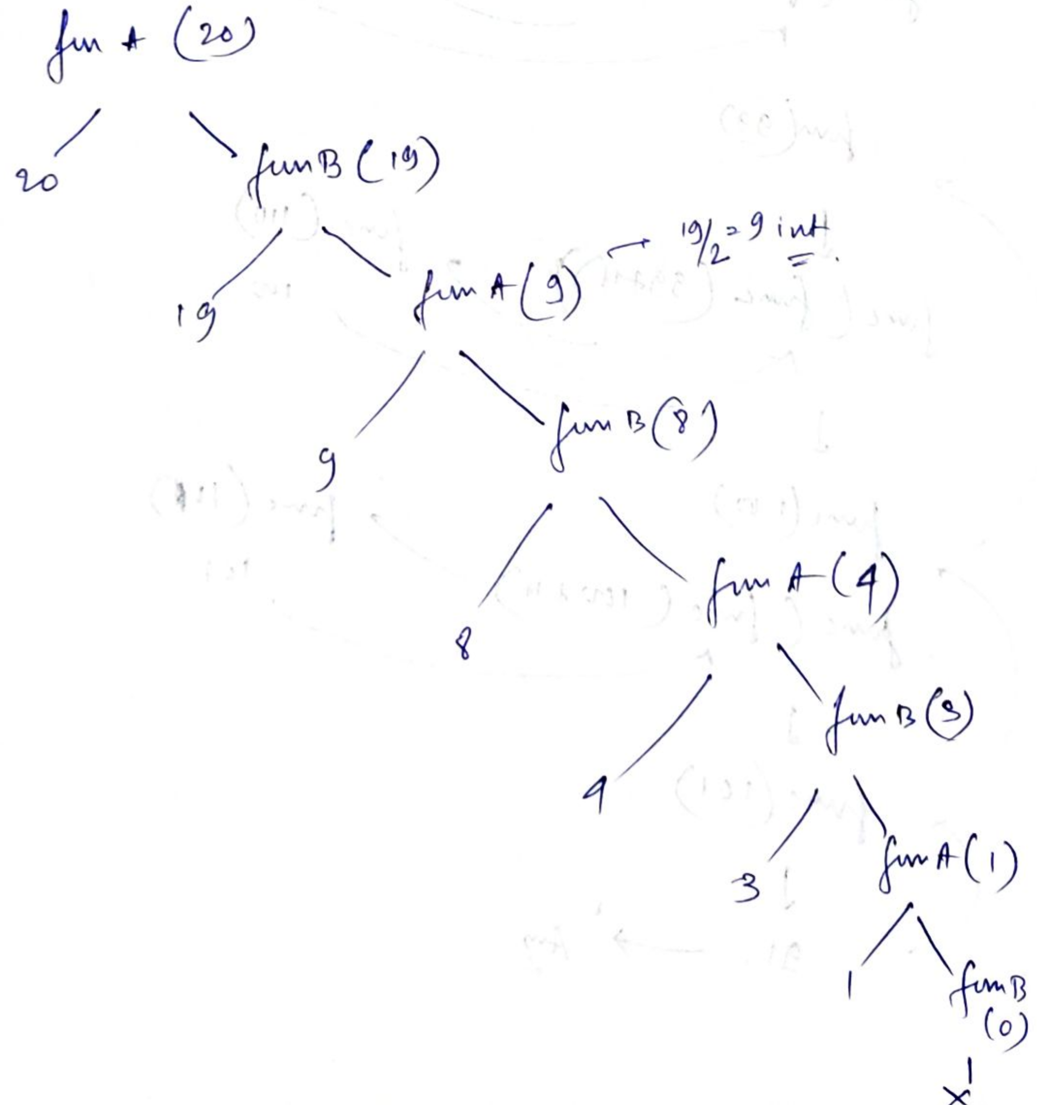
Space complexity → $O(n)$

• Indirect Recursion

```
void A(int n){
    if(<1){
        ==
        B(n-1);
    }
}
void B(int n){
    if(<1){
        ==
        A(n-3);
    }
}
```

Eg.

```
void funA(int n)
{   if(n>0)
    {   ~~[illegible]~~ cout << n;
            funB(n-1);
    }
}
void funB(int n)
{   if(n>1)
    {   cout << n;
        funA(n/2);
    }
}
```



• Nested Recursion

```
void func(int n)
{   if (4>)
    {
       ==
         func(func(n-1));
    }
}
```

Eg.

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

func(98)
```

func(98) > 91

↓

func(func(98+11)); → func(109)
99

func(99)

↓

func(func(99+11)) → func(110)
100

↓

func(100)

func(func(100+11) → func(111)
101

↓

func(101)

↓

91. ⟶ Ans