9/10/23

Week -> 7

Recursion ->

Time & Space Complexity Recursive Solⁿ

Rewind -> iterative

```
for(i=0; i<n; i++){
        cout << array[i];
}
```

T(c) => Time taken as a fⁿ of input 'n'

$1 \rightarrow 1^2 = 1$

$2 \rightarrow 2^2 = 4$

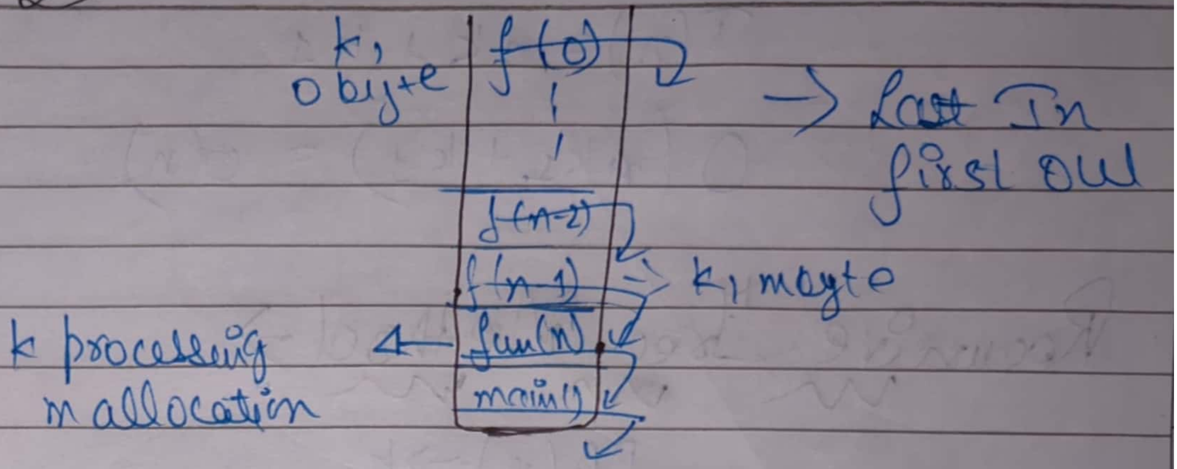
$3 \rightarrow 3^2 = 9$

Recursion ->

```
main(){
    fun(n);
}
```

```
fun(){
    if(n==0) return
    //Processing
    fun(n-1);
    int a, b, c;
}
```

9/10/23



$f(n), \underbrace{f(n-1), f(n-2)}, f(n-3)$ - —

↳ These are Instances

Print Array (Linear traversal of an Array)

```
void printArray(int a[], int n){
        if(n==0) return;
        cout << *a << " "; → k work
        printArray(a+1, n-1);
    }
```

$f(n) = k + f(n-1)$

① ⇒ $T(n) = k + T(\cancel{n-1})$

$T(\cancel{n-1}) = k + T(\cancel{n-2})$

$T(\cancel{n-2}) = k + \cancel{T}(n-3)$

|

|

$T(\cancel{1}) = k + T(\cancel{0})$

$+ \quad \cancel{T(0)} = k_1$

Formula based Method

$T(n) = nk + k_1$

$O(nk_1 + k_1) = O(n)$

Recursive Tree Method ->

PA(n) --> k

PA(n-1) --> k

PA(n-2) --> k

n-3 --> k

n=2 --> k

n=0 --> k_1

=> $T(n) = nk + k_1$

$T(n) = O(nk)$

$\boxed{TC = O(n)}$

Space Complexity => O.S is making stack for your Recursive program

PA(0) -> m,

+

n space

1

2

:

n-2 PA(n-2) -> m

n-1 PA(n-1) -> m

n PA(n) -> m

main()

$O(n+1) = O(n)$

9/10/23

$n = 3$

PA(0) → 1
PA(1) → 2
PA(2) → 3 → 3 + (1) Ignored   O(3)
PA(3) → 4
main

Factorial Rec

```
int fact (int n) {
            if (n==1)
                return 1;

        return n* fact (n-1);
}
```

$f(n) = n * F(n-1)$

→ processing

$T(n) = k + \cancel{T(n-1)}$

$\cancel{T(n-1)} = k + \cancel{T(n-2)}$

$\cancel{T(n-2)} = k + F(n-3)$

⋮   ⋮

$+$

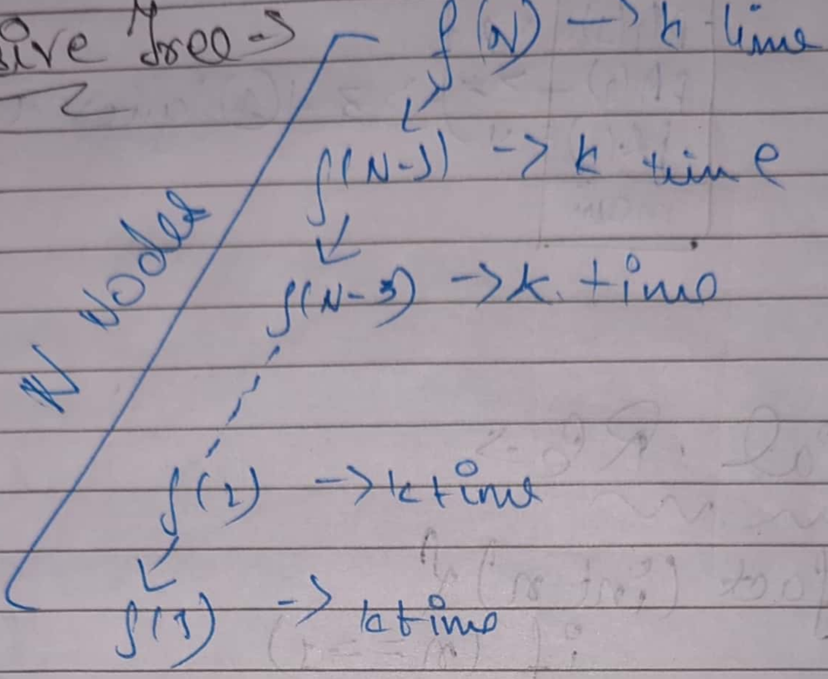
$T(1) = k$

$T(n) = nk$

$O(Tn) = O(nk) \Rightarrow TC = O(n)$

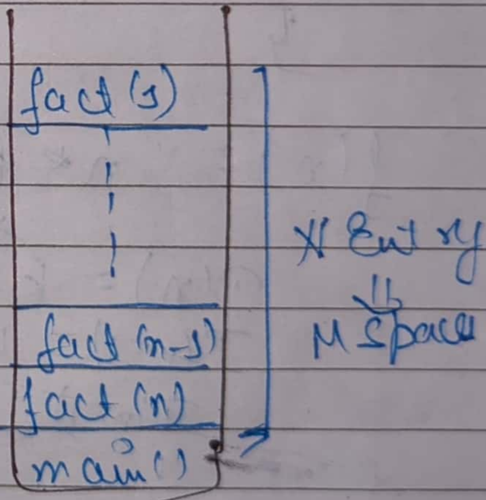
9/10/23

Recursive Tree ->



Space Complexity of Fact

$O(n*m) \Longleftarrow$

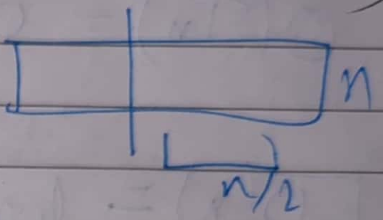


Binary Search RE ->

```
int Bs( int a[], int k, int start , int end){
        if (start > end)
            return -1;
    int mid = start + (end- start)/2;
```

9/10/23

```
If (a[mid]==k)
        return mid;
else if (k > a[mid])
{
    return bsr(a,k,mid+1,end);
}
else {
        return bsr(a,k, ~~st~~start ,mid-1)
}
```



$F(n) = k + F(n/2)$

$\Rightarrow \quad T(n) = k + \cancel{T(n/2)}$

$\cancel{T(n/2)} = k + \cancel{T(n/4)}$

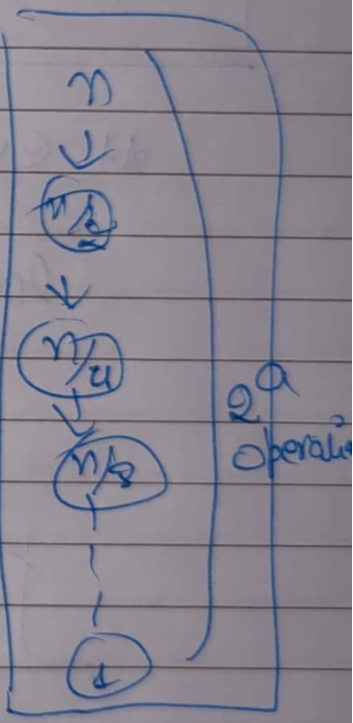
$\cancel{T(n/4)} = k + \cancel{T(n/8)}$

$\vdots$

$\cancel{T(2)} = k + \cancel{T(1)}$

$+ \quad \cancel{T(1)} = k$

$$\underline{T(n) = a*k}$$



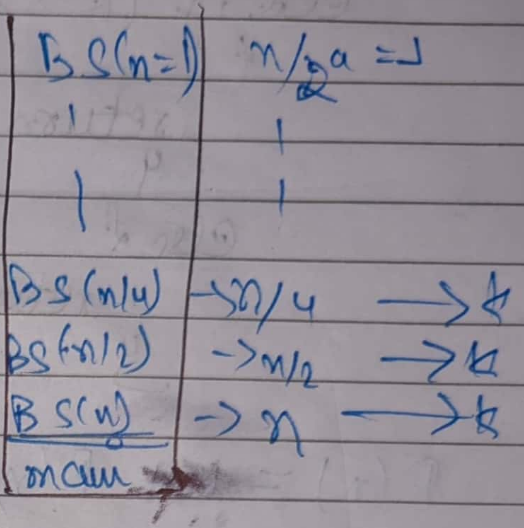
$$\boxed{T(n) = \log(n)}$$

2^a operations from n to 1

$\frac{n}{2^a} = 1 \Rightarrow a = \log n$

9/10/23

Space Complexity ->



$a = \log n$

$O(T_n) = O(k \log n)$

↳ space

$\boxed{O(n) = \log n}$ → space

At every stack we are using k space

logn -> Times iterate karna hai

$O(T_n) = O(\underset{\text{const so, remove it}}{k} \log n)$

$\boxed{TC = O(\log n)}$

Fibonacci Series RE ->

Very bad code to do fibonacci ←

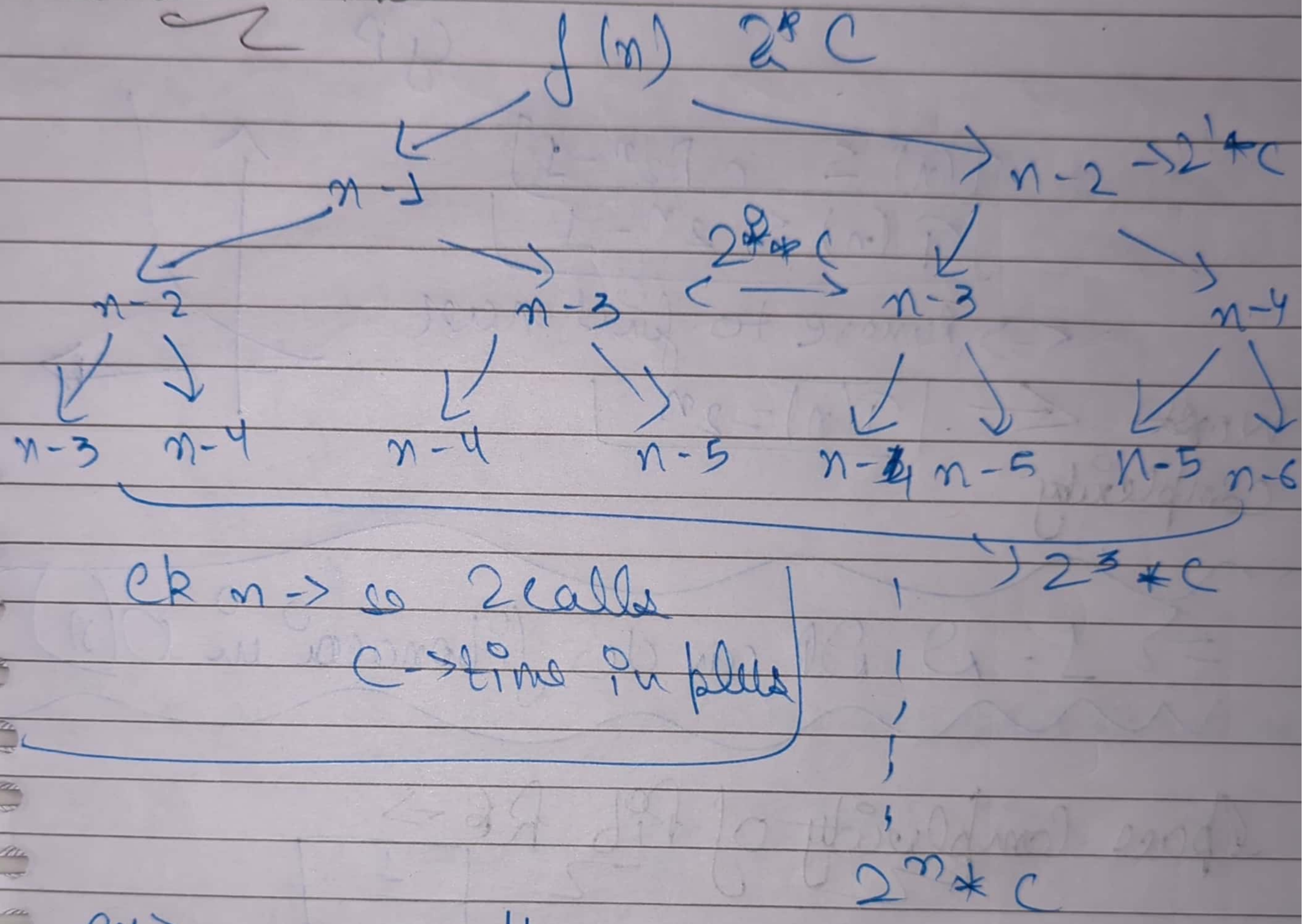
```
int fib (int n) {
    if (n = = 0 || n = = 1) return n;

  return fib(n-1) + fib(n-2);
}
```

9/10/23

Date (101)

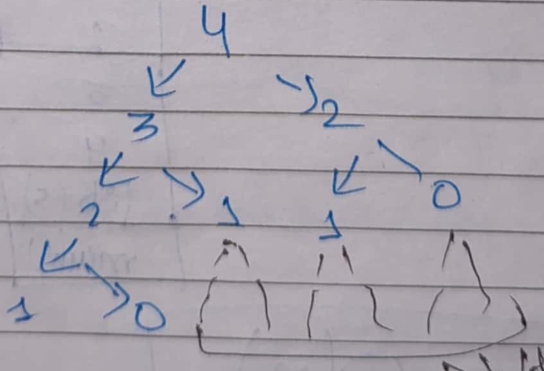
0	1	1	2	3	~~5~~	8
↓	↓	↓	↓	↓	↓	↓
0th	1st	2nd	3rd	4th	5th	6th

Recursive tree ->



ek n -> so 2 calls
c -> time in plus

ex>



-> Hypothetically more to find upper case

$T(n) \le 2^0 + 2^1 + 2^2 + 2^3$

② -> Isliy

9/10/23

Date (102)

$T(n) \le 2^0 * c + 2^1 * c + 2^2 * c --- + --- + 2^{n-1} * c$

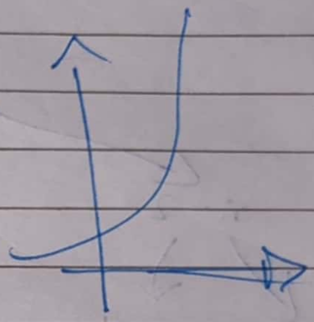
$T(n) \le c\left(\underbrace{2^0 + 2^1 + ---- + 2^{n-1}}_{GP}\right)$

$T(n) \le c[2^n - 1]$

$\boxed{T(n) \le 2^n - 1}$

<-> Remove to find max

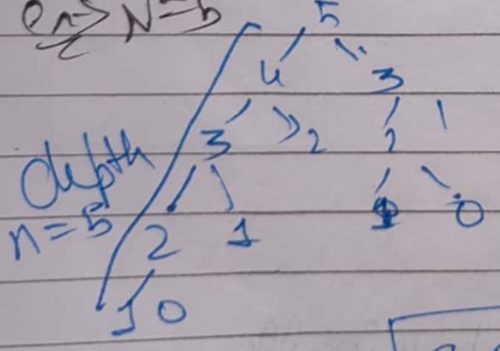
Worst Complexity ⟸ $\boxed{T(n) = 2^n}$

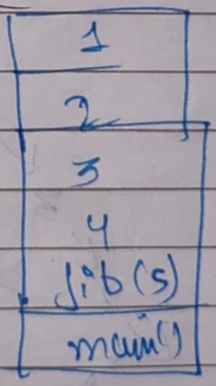


=> 1-D DP can do fibonacci in O(n)

Space Complexity of Fib RE ->

ex-> N=5





$\boxed{SC = O(n)}$ -> depth