Today's agenda

- Pow(a, n)
- TC & SC of Recursion

Q) Given a and N, calculate a^n.

Ex:

a	n	
2	3	→ 8
3	5	→ 243

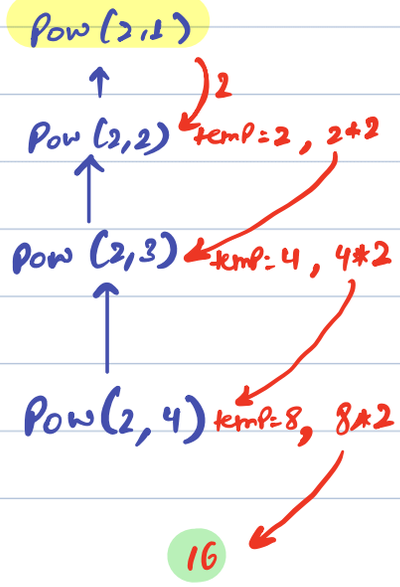
```
int Pow(int a, int n){
    if(n==1){return a;}

    int temp = Pow(a, n-1);
    return temp*a;

}
```

Faith: Given a and n, calculate and return a^n.

main logic:

base case:

n==0 → 1
or
n==1 → a



No. of function : 4 → n

T.C of one function : O(1)

overall T.C: O(1) * N : O(N)

// Give me something better than O(N).

① $a^n = a^{n-1} * a$

$2^5 \downarrow 2^4 \downarrow 2^2 \downarrow 2^2 \downarrow 2^1$

$a^n \longrightarrow a^{n/2}$ (temp)

↳ if n == even → $a^n = a^{n/2} * a^{n/2}$ (temp, temp)

↳ if n == odd → $a^n = a^{n/2} * a^{n/2} * a$

temp * temp
$a^{n/2} * a^{n/2} = a^n$

2^{16} (temp) → 2^8 (temp) → 2^4 (temp) → 2^2 (temp) → 2^1

temp: 256 → 256 * 256 = 65536

temp: 16 → 16 * 16 = 256

temp: 4 → 4 * 4 = 16

temp: 2 → 2 * 2 = 4

$2^{17} \rightarrow 256 * 256 * 2$

256

2^8

$a^n \longrightarrow a^{n/2}$ (temp)

↳ if n == even → $a^n = a^{n/2} * a^{n/2}$ (temp, temp)

↳ if n == odd → $a^n = a^{n/2} * a^{n/2} * a$

```
int Pow(int a, int n){
    if(n==1){return a;}
    int temp = Pow(a, n/2);
    if(n%2==0){return temp*temp;}
    else{return temp*temp*a}
}
```

T.C: O(logN)

Faith: Given a and n, calculate and return a^n.

main logic:

$a^n \longrightarrow a^{n/2}$ (temp)

↳ if n == even → $a^n = a^{n/2} * a^{n/2}$ (temp, temp)

↳ if n == odd → $a^n = a^{n/2} * a^{n/2} * a$

base case:

a: 2 n: -3

$\frac{1}{2^3}$

```
int Pow (int a, int n){
    if (n==1) { return a; }

    int temp = Pow(a, n/2);
    if (n%2==0){return temp* temp;}

     else{ return temp* temp*a }

}
```

a=2 N=37

↓

N=18

↓

N=9

↓

N=4

↓

N=2

↓

N=1

Overall T.C: (T.C of 1 function) * (Total no. of function)

(T.C of 1 function) → O(1)

(Total no. of function) → logN

Ug O(logN)

→ $a^n = a^{n/2} * a^{n/2}$

$a^n = a^{n/3} * a^{n/3} * a^{n/3}$

TC of recursive code

Overall T.C: (T.C of 1 function) * (Total no. of function)

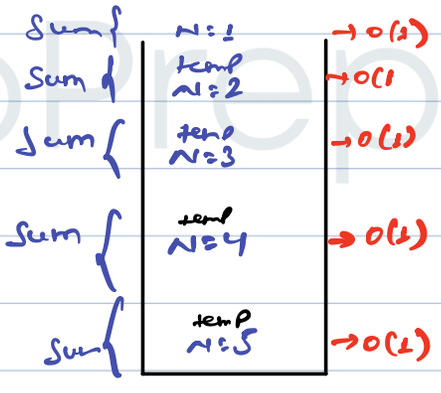
O(1)

```
int Sum (int N){
    if (N==1){ return 1; }
    int temp= Sum (N-1);
    return temp+N;
}
```

T.C of 1 function : O(1)

No. of function : N

Total T.C : O(N)



S.C: O(N)

↳ if you delete a space that is already considered in your space complexity and create new space in place of previous space, you are not increasing your space complexity.

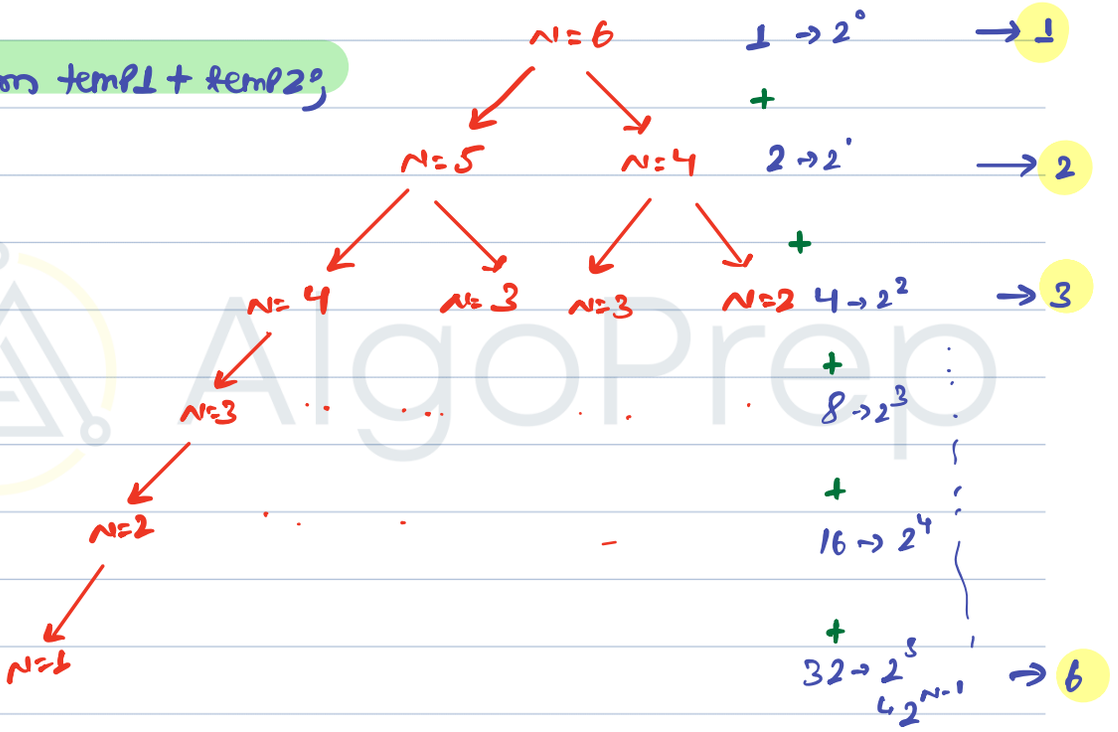
→ T.C of fibonacci

```
int fib (int N){
    if(N==0||N==1){return N;}
    int temp1 = fib(N-1);
    int temp2 = fib(N-2);
    return temp1 + temp2;
}
```

T.C of 1 function : O(1)

No. of function : 2^N ⇒ $O(2^N)$

Total T.C :



S.C:

$2^0 + 2^1 + 2^2 + \dots + 2^{N-1}$ = Sum of G.P: $a * \frac{(r^n - 1)}{r-1}$

$= 1 * \frac{2^N - 1}{2-1}$

$= 2^N - 1$

→ S.C: O(N)

Total S.C: (Space complexity of one function) * max no. of levels.

No. of Calls = x ⇒ No. of functions: x^n

Count of levels = n

Break till 9:42 Pm

Space Complexity:

↳ How much Space are you using.

iterative: Variables + $O(1)$, Array + n size + $O(n)$, String + n size + $O(n)$

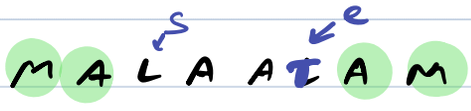
recursive: variables, Array, String, Call Stack

note: Input or output won't be considered in Space Complexity (Big O Notation).

Q) Given an array (character), check if it is Palindrome or not?
↳ recursion

Ex: MALAYALAM → true

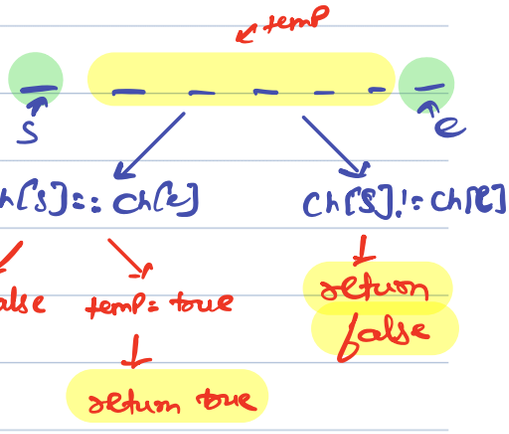
Ex: a a b b a → false




```
boolean isPalindrom (char ch[], int s, int e){
    if (s == e) { return true;
    if (ch[s] == ch[e]){
        boolean temp = isPalindrom (ch, s+1, e-1);
        return temp;
    }
    else { return false; }
}
```

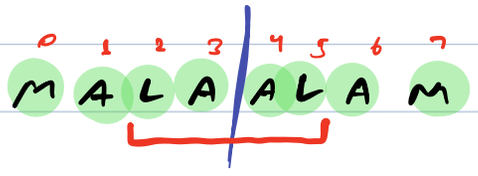
Faith: check and return whether Ch array is Palindrome between s to e.

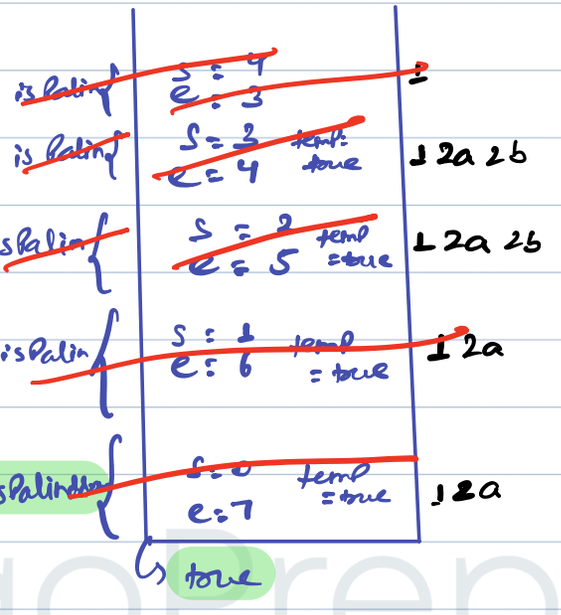
Main logic:

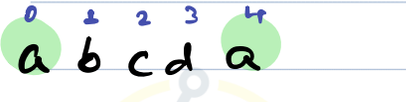


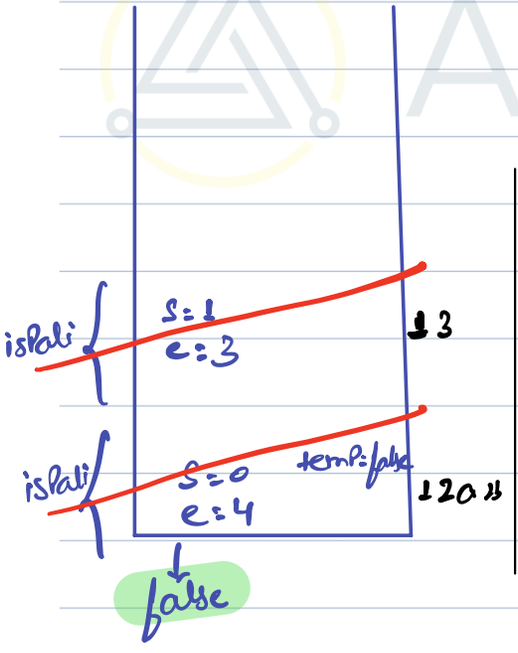
base case:

```
boolean isPalindrom (char ch[], int s, int e){
1  if (s==e || s>e) { return true; }
2  if (ch[s] == ch[e]){
a    boolean temp = isPalindrom (ch, s+1, e-1);
b    return temp;
3  } else { return false; }
}
```










```
boolean isPalindrom (char ch[], int s, int e){
1  if (s==e || s>e) { return true; }
2  if (ch[s] == ch[e]){
a    boolean temp = isPalindrom (ch, s+1, e-1);
b    return temp;
3  } else { return false; }
}
```

T.C: $O(1) * N/2 \approx O(N/2) \approx O(N)$

S.C: $O(1) * N/2 \approx O(N/2) \approx O(N)$