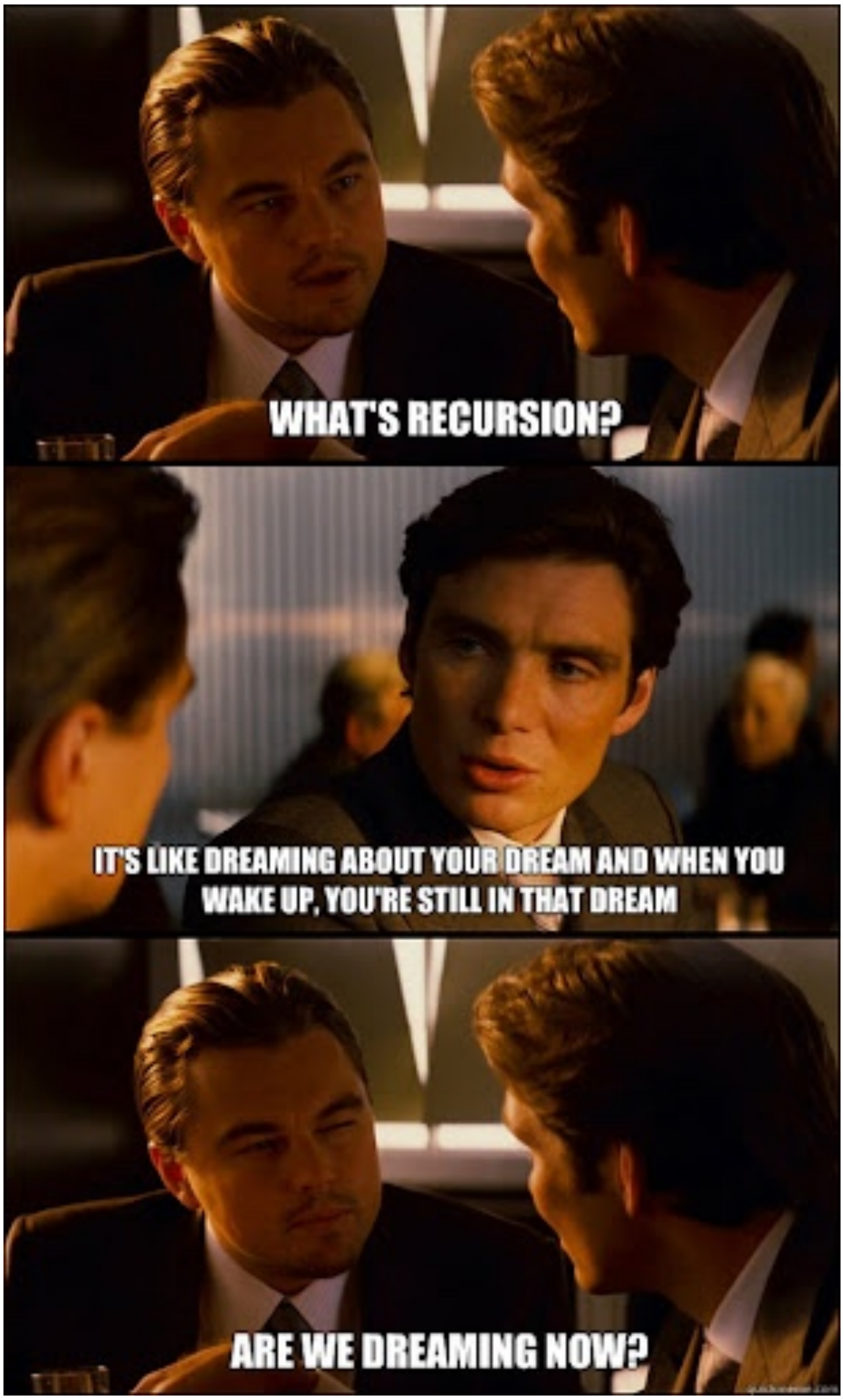
Agenda

1. Power Function
2. Print array
3. All Indices
4. Check Palindrome


WHAT'S RECURSION?
IT'S LIKE DREAMING ABOUT YOUR DREAM AND WHEN YOU WAKE UP, YOU'RE STILL IN THAT DREAM
ARE WE DREAMING NOW?

• Power Function

Given a and n, find a^n using recursion. (N >= 0)

$a=3, \quad n=4$ $\qquad 3^4 = 3\times3\times3\times3 = \underline{\underline{81}}$

Idea 1

$$a^N = \underbrace{a \times a \times a \times a \times \text{— ——} \times a \times a}_{N \text{ times.}}$$

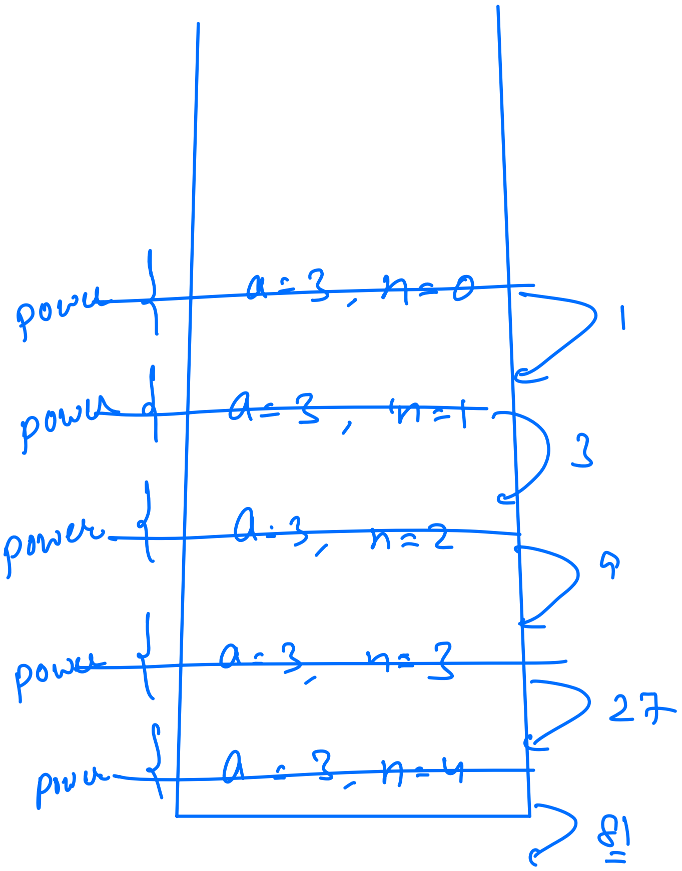
$$\boxed{a^N = a^{N-1} \times a}$$

power(a, N) = power(a, N-1) × a

```
int power( int a, int n){
    if(n == 0) {return 1}
    return power( a, n-1) x a ;
}
```

Idea 1 - Dry Run

```
int power( int a, int n){
|     if(n==0) {return 1}
|     return power( a, n-1) x a ;
}
```



$$\boxed{\begin{array}{l} T.C \rightarrow O(N) \\ S.C \rightarrow O(N) \end{array}}$$

$2^{16} = 2^{15} \times 2$

$$\left.\begin{array}{l} 2^{16} = 2^{8} \times 2^{8} \\ 3^{64} = 3^{32} \times 3^{32} \end{array}\right\} \quad a^{N} = a^{N/2} \times a^{N/2}$$

if N is even

$$\left.\begin{array}{l} 2^{17} = 2^{8} \times 2^{8} \times 2 \\ 3^{15} = 3^{7} \times 3^{7} \times 3 \end{array}\right\} \quad a^{N} = a^{N/2} \times a^{N/2} \times a$$

if N is odd

Idea 2

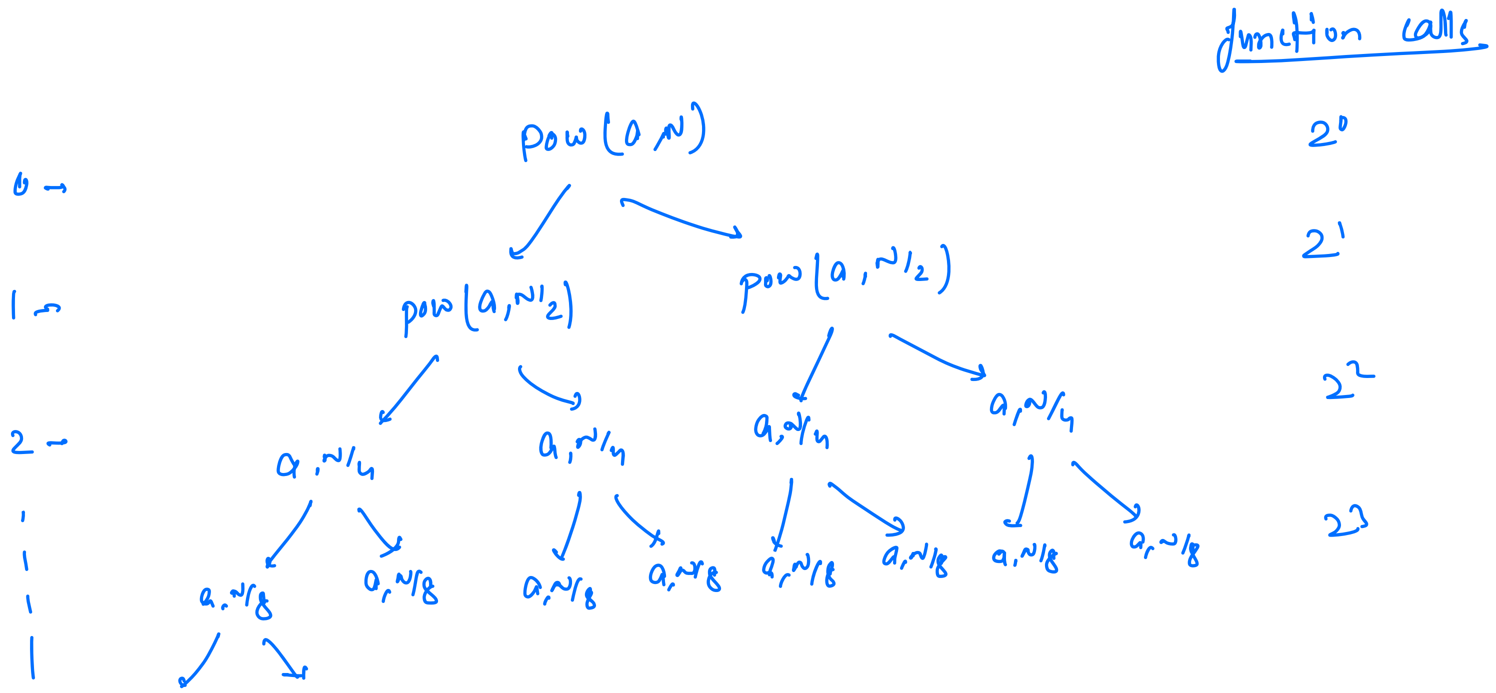
if $N \% 2 == 0$, $\text{power}(a,N) = \text{power}(a,N/2) \times \text{power}(a,N/2)$

if $N \% 2 == 1$, $\text{power}(a,N) = \text{power}(a,N/2) \times \text{power}(a,N/2) \times a$

```
int power ( int a, int N){
    if( N==0) { return 1 }
    if( N %2 == 0){
        return power(a,N/2) x power(a,N/2);
    }
    else{
        return power(a,N/2) x power(a,N/2) xa ;
    }
}
```

T.C → O(N)

S.C → $O(\log_2 N)$



no. of levels → $\log_2 N$

total function calls = $\underbrace{1 + 2 + 2^2 + 2^3 + \_\_\_\_}_{\log_2 N \text{ terms}}$

$$= \frac{1 \cdot (2^{\log_2 N} - 1)}{(2-1)}$$

$a = 1$

$r = 2$

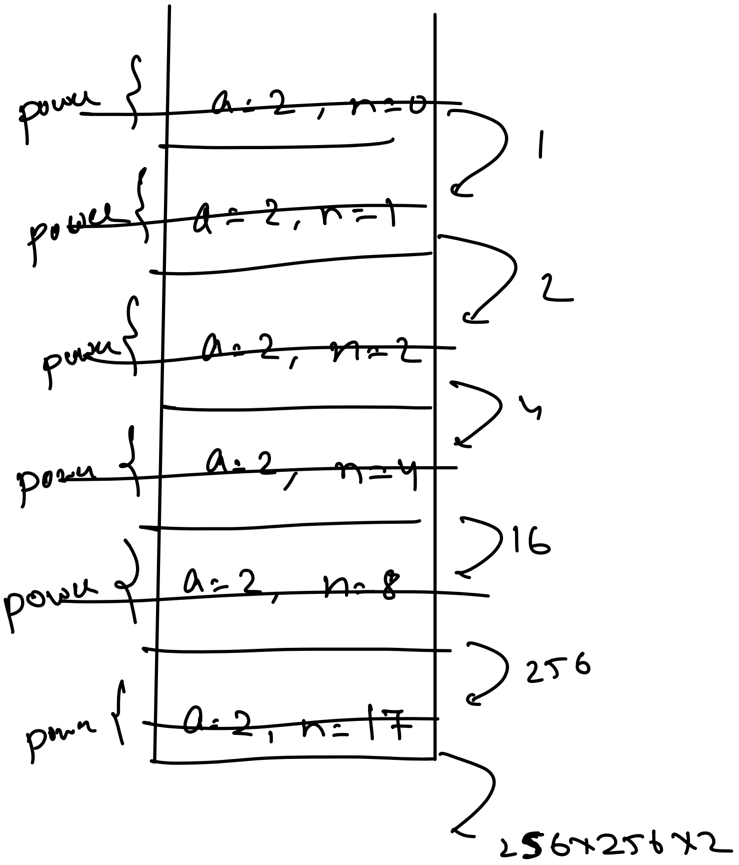
no. of terms = $\log_2 N$

$= (N-1)$

Optimised Approach

```
int power ( int a, int N){
|   if( N==0) { return 1}
|   int p = power( a, N/2);
|   if( N%2 ==0){
|   {     return p x p;
|   }
|   else
|   {     return p x p x a;
|   }
}
```

$$\begin{bmatrix} T.C \rightarrow O(\log_2 N) \\ S.C \rightarrow O(\log_2 N) \end{bmatrix}$$



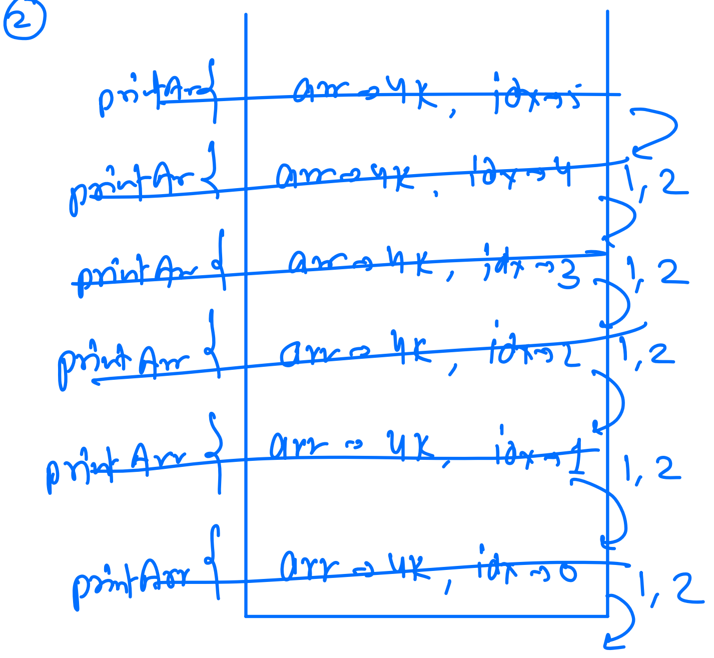
- ## Print array using recursion

Given an array of size N, print all its elements.

```
        4K
arr[] → [ 5 , 2 , 7 , 3 , 4 ]
          0   1   2   3   4
```

```
void   printArr( int[] arr , int idx ){
       if( idx == arr.length){ return}
       print( arr[idx]);                  ——— ①
       printArr( arr, idx + 1 );          ——— ②
}
```

T.C → O(N)
S.C → O(N)



~~printArr{ arr → 4K, idx → 5~~
~~printArr{ arr → 4K, idx → 4~~ 1, 2
~~printArr{ arr → 4K, idx → 3~~ 1, 2
~~printArr{ arr → 4K, idx → 2~~ 1, 2
~~printArr{ arr → 4K, idx → 1~~ 1, 2
~~printArr{ arr → 4K, idx → 0~~ 1, 2

o/p → 5 , 2, 7 , 3, 4

find max in array →

```
int findmax( int [] arr, int idx){
    if( idx == arr.length -1) { return arr[idx] }
    return Max(arr[idx] , findmax( arr, idx+1))
}
```

find sum of all elements of an array →

```
int sum ( int [] arr, int idx){
    if ( idx == arr.length) {return 0}
    return arr[idx] + sum( arr, idx+1);
}
```

All Indices of Array

Given an array A of size N and target integer B, all all indices where B is present in the array.

arr[ ] → [4, 5, 3, 1, 5, 4, 5]
0 1 2 3 4 5 6

arr[] → [1,2,3,1,1] , B=1
0 1 2 3 4

B = 5

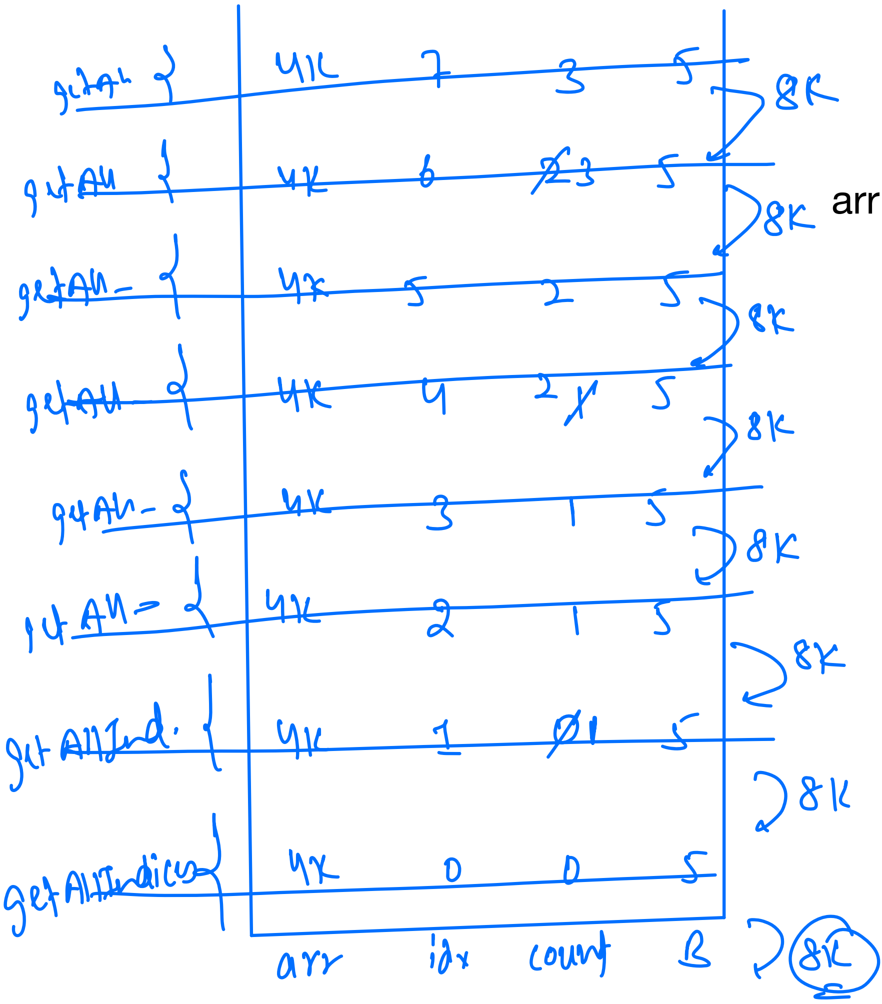
o/p → [0,3,4]

Output - [1, 4, 6]

```
int[]   getAllIndices ( int[] arr, idx, count, B){
    if( idx == arr.length) { int[] res = new int[count], return res }

  if( arr[idx] == B){ count++}

     int[] res = getAllIndices( arr, idx+1, count, B);

   if( arr[idx] == B){
          res[count-1] = idx ;
   }

   return res;
}
```

T.C → O(N)
S.C → O(N)

- ***Dry Run***



	arr	idx	count	B	
getAll {	4K	7	3	5	8K
getAll {	4K	6	~~2~~ 3	5	8K
getAll {	4K	5	2	5	8K
getAll {	4K	4	2 ~~1~~	5	8K
getAll {	4K	3	1	5	8K
getAll {	4K	2	1	5	8K
getAllInd. {	4K	1	~~0~~ 1	5	8K
getAllIndices {	4K	0	0	5	8K

arr[ ] → $\overset{4K}{[4, 5, 3, 1, 5, 4, 5]}$

0 1 2 3 4 5 6

res → $\overset{8K}{}$

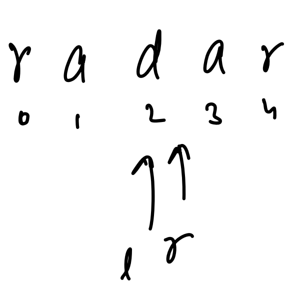
0	1	2
~~0~~ 1	~~0~~ 4	~~0~~ 6

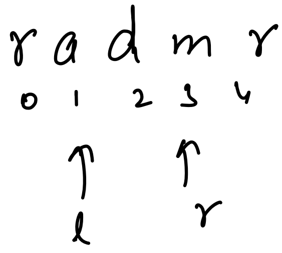
• Check Palindrome

Given a String, write recursive function to check if it is a palindrome.

Example -

1. "radar" ans: true
2. "area" ans: false

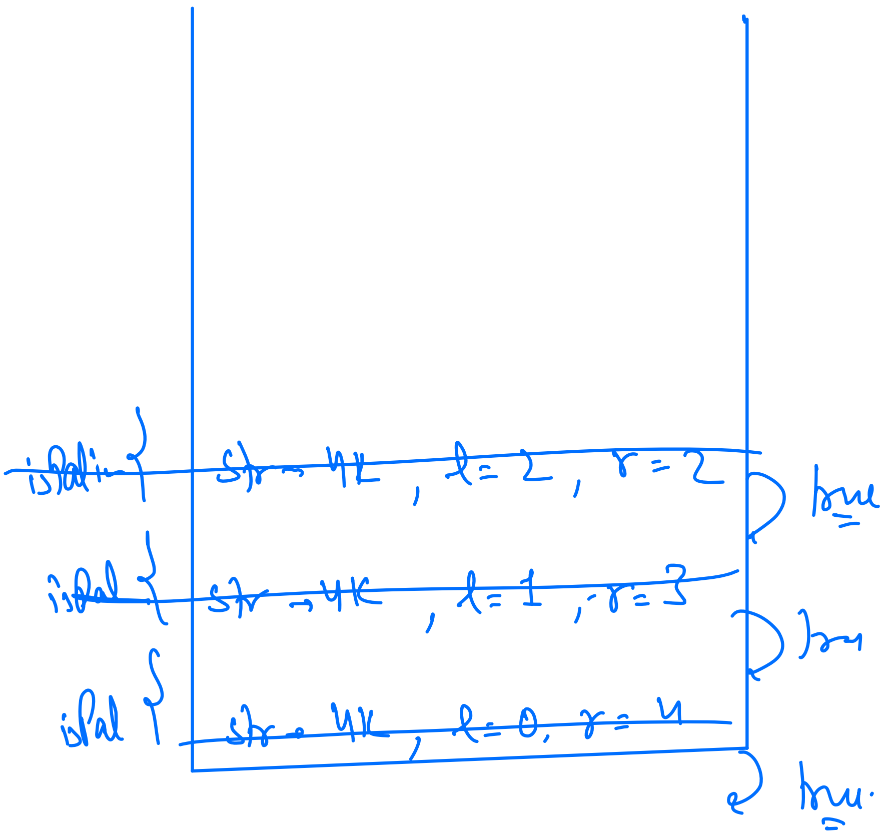




```
boolean isPalindrome ( String str, int l , int r){
        if( l ≥ r) { return true }

      if( str[l] != str[r]) {
              return false;
      }
      return isPalindrom ( str, l+1, r-1);
}
```

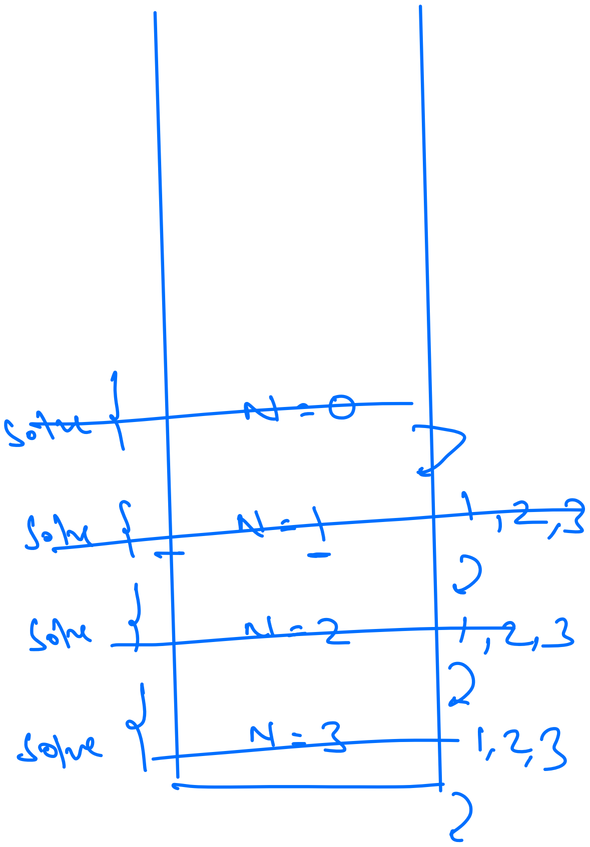
T.C → O(N)
S.C → O(N)

str → a b c b a
0 1 2 3 4


isPalin{ ~~str → 4K , l = 2 , r = 2~~
true
isPal{ ~~str → 4K , l = 1 , r = 3~~
true
isPal{ ~~str → 4K , l = 0, r = 4~~
true.

```
void solve(int N){
    if(N == 0)
        return;
    print(N);      ①
    solve(N-1);    ②
    print(N);      ③
}
```

o/p 3 ,2 ,1 ,1 ,2 ,3



```
void solve(int N){
    if(N == 0)
        return;
    print(N);
    solve(N-1);
}
```

N = -3

⇒ Error → stack overflow.

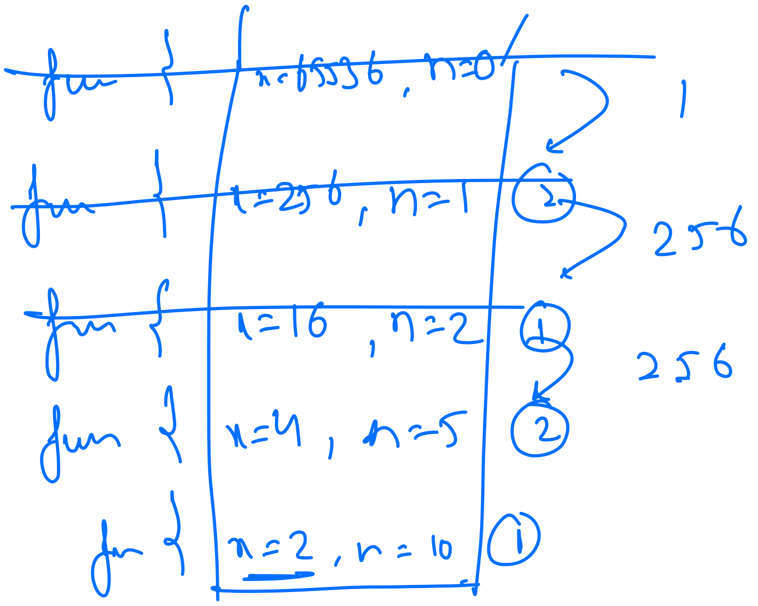
→ Tower of Hanoi

→ problem on recursion

⇓

Problem Solving Session. → optional.

```
              2        10
int fun( x , n ){
    if( n == 0 ){ return 1}

  else if ( n%2 == 0){
    |     return fun(x*x, n/2);  —— ①
    }
    else{
    |     return   x * fun(x*x,(n-1)/2);  —— ②
    }
}
```



$\| N \geq 0$