Agenda

1) Revision quizzes
2) Power function **
3) Print Array
4) indices of an Array **
5) check palindrome

What is recursion ?

9 users have participated

A	A function that calls itself	89%
B	A function that calls another function	0%
C	A loop that repeats until a condition is met	0%
✓ D	A function that solves a problem by breaking it into smaller subproblems and calling itself	11%

What data structure is used for function call tracing in recursion?

8 users have participated

A	Queue	0%
B	Linked List	0%
✓ C	Stack	100%
D	Array	0%

Which of the following is the base case for calculating the factorial of a number using recursion?

6 users have participated

A	N == 2	0%
✓ B	N == 0	100%
C	N == 1	0%
D	N == -1	0%

What is the time and space complexity for calculating factorial using recurion ?

13 users have participated

✓ A	O(N), O(N)	62%
B	O(N), O(1)	38%
C	O(1), O(N)	0%
D	O(N), O(log(N))	0%

```
function factorial (n) {
    if (n==0) {
        return 1;
    }
    ans= n* factorial (n-1);
    return ans;
}
```

n=0 (base)
↑
n=1
↑
n=2
↑
n=3
↑
n=4
↑
n=5

n c

n=5

TC = tc of single func. * total no. of calls

= 1 * n = O(n)

SC = sc of single func * max. functions in call stack at any moment

= 1 * n = O(n)

To compute Fibonacci(N) you need answers of ?

1 user has participated

A	fibonacci(N - 1)	0%
B	Fibonacci(N - 2)	0%
✓ C	Fibonacci(N - 1) and Fibonacci(N - 2)	100%
D	none	0%

Q. Given a, n find a^n.

$a = 2 \quad n = 5 \quad ans = 2^5 = 32$

$a = 3 \quad n = 4 \quad ans = 3^4 = 81$

$pow(a,n) = pow(a,n-1) * a$

$a^n = a^{n-1} * a$

```
function pow(a, n) {
    if (n==0) {
        return 1;
    }

    ans = pow(a, n-1) * a;
    return ans;
}
```

Assumption: takes (a, n) and return a^n

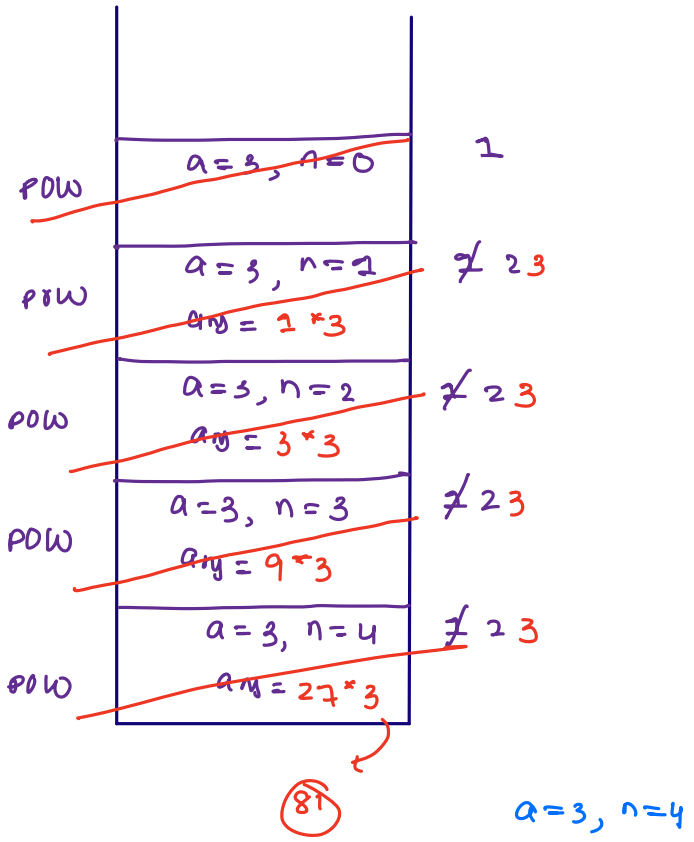
main logic:

```
pow(a,n) = pow(a, n-1) * a;
```

base case:

```
if(n==0) {
    return 1;
}
```

```
function pow(a, n) {
    if (n==0) {
        return 1;
    }
    ans = pow(a, n-1) * a;
    return ans;
}
```



TC: O(n)

SC: O(n)

Optimisation:

$$a^n = a^{n-1} * a \quad \text{(prev. logic)}$$

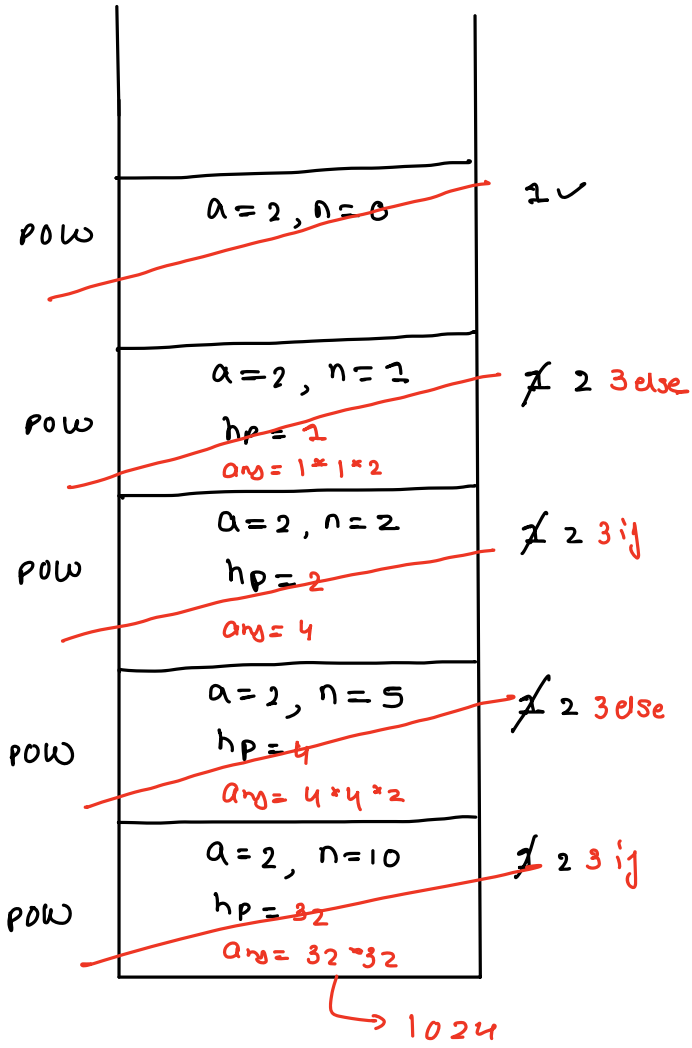
$$a^n = \begin{cases} a^{n/2} * a^{n/2} & \text{(when n is even)} \\ a^{n/2} * a^{n/2} * a & \text{(when n is odd)} \end{cases}$$

```
function pow(a, n) {
    if (n == 0) {
        return 1;
    }
    hp = pow(a, n/2);

    if (n % 2 == 0) {
        ans = hp * hp;
        return ans;
    }
    else {
        ans = hp * hp * a;
        return ans;
    }
}
```

```
function pow(a, n) {
    if (n==0) {
1       return 1;
    }
2   hp = pow(a, n/2);

    if (n%2 == 0) {
        ans = hp * hp;
        return ans;
    }
3   else {
        ans = hp * hp * a;
        return ans;
    }
}
```



a=2, n=10

TC: O(log n)

SC: O(log n)

Smart power

→ one more approach

```
Junction pow(a,n) {
    if(n==0) {
        return 1;
    }

    if(n%2==0) {
        ans = pow(a,n/2) * pow(a,n/2);
        return ans;
    }
    else {
        ans = pow(a,n/2) * pow(a,n/2) * a;
        return ans;
    }
}
```

Fake smart power

TC: O(n)

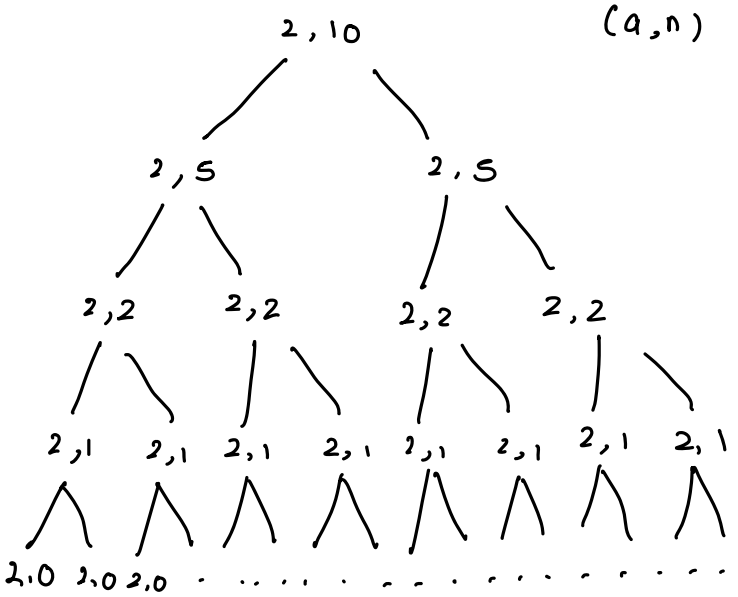
SC: O(log2n)

```
function pow(a,n) {
    if(n==0) {
        return 1;
    }

    if(n%2==0) {
        ans = pow(a,n/2) * pow(a,n/2);
        return ans;
    }
    else {
        ans = pow(a,n/2) * pow(a,n/2) * a;
        return ans;
    }
}
```

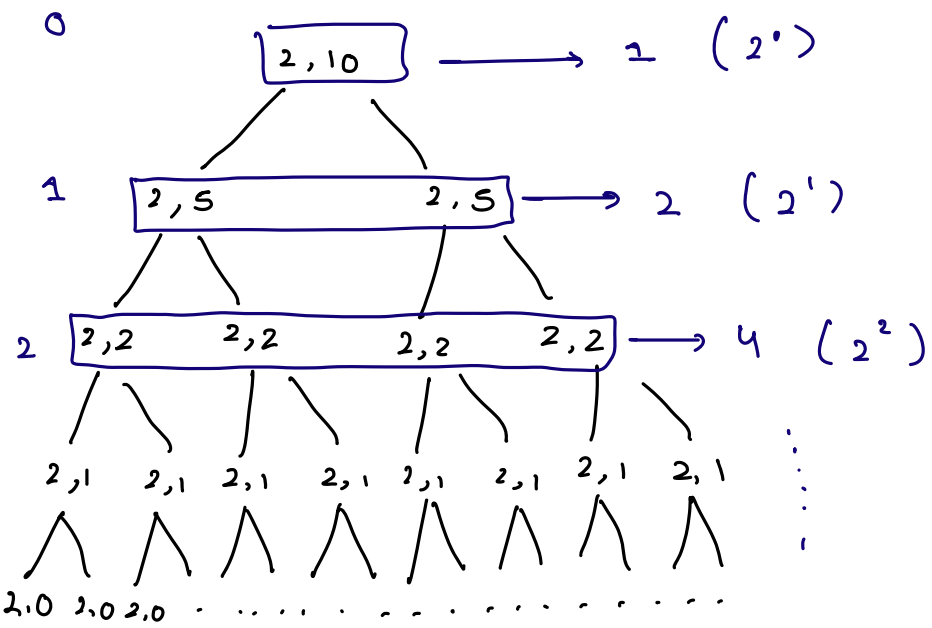
a = 2 , n = 10

(a,n)



a = 2 , n = 10

(a,n)



$\rightarrow 1 \ (2^0)$

$\rightarrow 2 \ (2^1)$

$\rightarrow 4 \ (2^2)$

last level $\rightarrow \log_2 n$

total calls $= 2^0 + 2^1 + 2^2 + \ldots\ldots + 2^{\log_2 n}$

$= 1 + 2 + 2^2 + \ldots\ldots + 2^{\log_2 n}$ $\quad (a=1, r=2, t=\log_2 n)$

$= \dfrac{1\,(2^{\log_2 n} - 1)}{1}$ $\quad (2^{\log_2 n} = n)$

$$= \frac{1\,(n-1)}{2}$$

$$= n$$

Q. Given A[], print it using recursion.

A = [10 20 30 40 50]
0 1 2 3 4

display(A, i) ⇒ print(A[i]) + display(A, i+1)

```
function display(A[], i) {
    if (i == A.length) {
        return;
    }

    print(A[i]);
    display(A, i+1);
}

function solve(A) {
    display(A, 0);
}
```

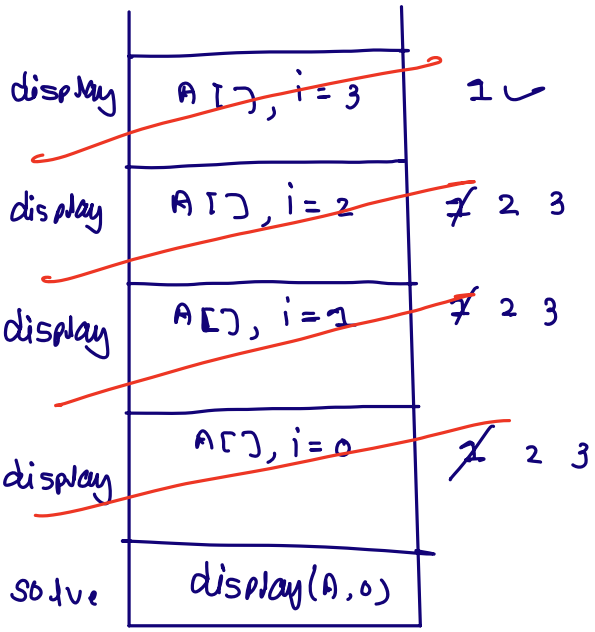
display(A, i) → prints everything from i to last

→ print(A[i]) + display(A, i+1)

```
function display (A[], i) {
    if (i == A.length) {
        return;
    }
    print(A[i]);
    display(A, i+1);
}

function solve(A) {
    display(A,0);
}
```

A = | 10 | 20 | 30 |
0 1 2



10 20 30

Q. Given A[] and B, find all indices of B in A[].

A = [4 5 3 1 5 4 5]
0 1 2 3 4 5 6

B = 5

ans[] = [1 4 6]
0 1 2

```
A = [1, 2, 3, 1, 1]
    0  1  2  3  4
B = 1
```

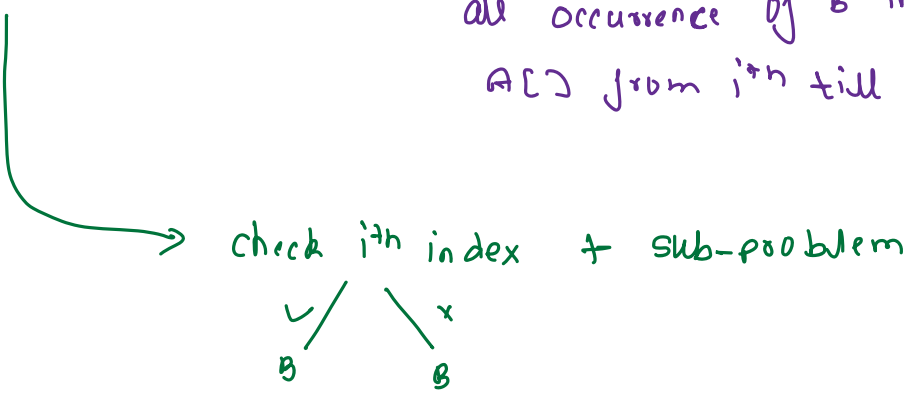
ans[] = [0, 3, 4]
0 1 2

note : Imagine the situatic of ~~static~~ array (in context ans)

A = [4 5 3 1 5 4 5]
0 1 2 3 4 5 6

B = 5

Assumption → occur(A, B, i, count) => returning ans arr containing all occurrence of B in A[] from i^{th} till last



```
function occur(A[], B, i, count) {
    if (i == A.length) {
        return ans[count];
    }
    if (A[i] == B) {
        ans = occur(A, B, i+1, count+1);
        ans[count] = i;
        return ans;
    }
    else {
        ans = occur(A, B, i+1, count);
        return ans;
    }
}

function solve(A[], B) {
    return occur(A, B, 0, 0);
}
```

TC: O(n)

SC: O(n)

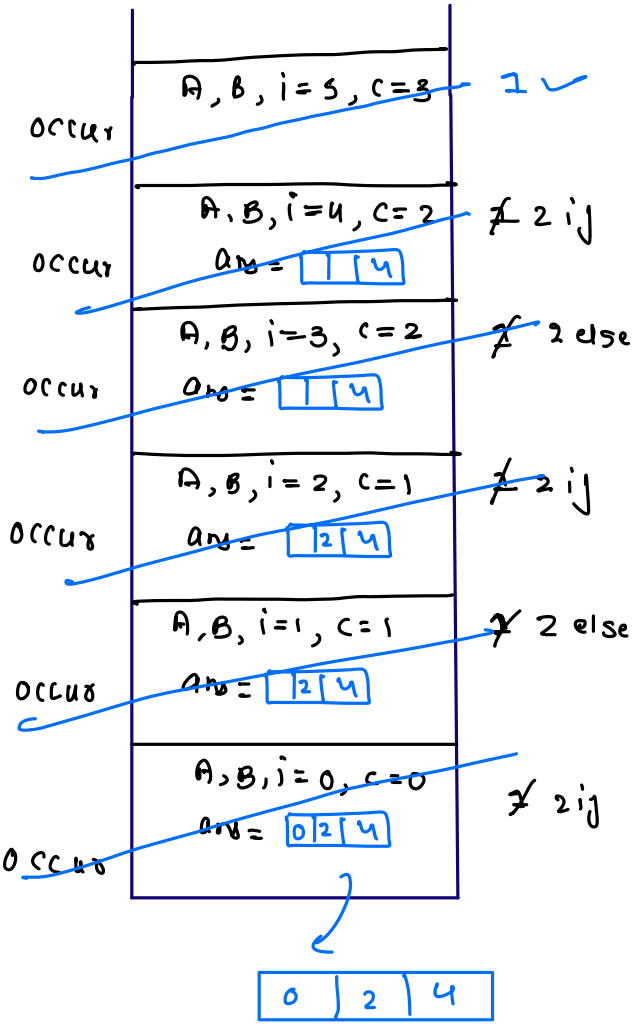
```
function occur(A[], B, i, count) {
    if (i == A.length) {
        return ans[count];
    }
    if (A[i] == B) {
        ans = occur(A, B, i+1, count+1);
        ans[count] = i;
        return ans;
    } else {
        ans = occur(A, B, i+1, count);
        return ans;
    }
}
```

1

2

A = | 4 | 3 | 4 | 0 | 4 |
0 1 2 3 4

B = 4



HW: one more dry-run

Q. Given a string, write recursive code to check if it is palindromic or not.

str = racecar true

str = tomtom false

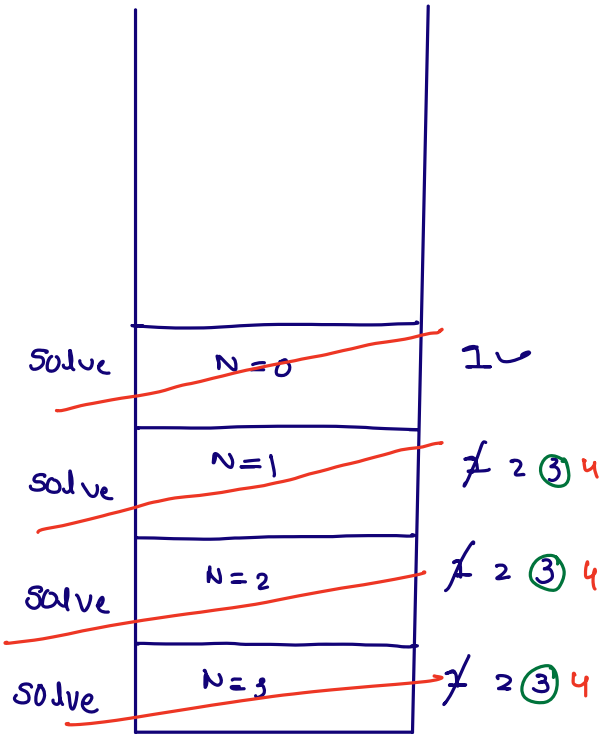
```
function pal ( str, l, r ) {
    if ( l >= r ) {
        return true;
    }

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

exercise: create call stack diagram

```
function solve ( str ) {
    return pal (str, 0, n-1);
}
```

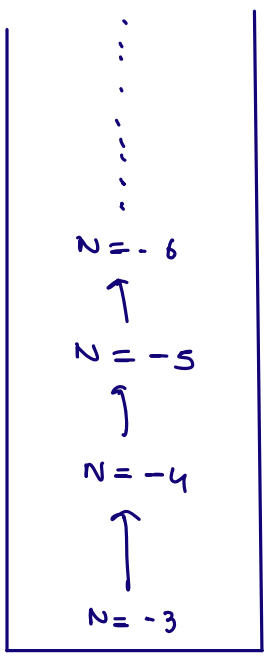
```
void solve(int N){
    if(N == 0)
        return;
    print(N);
    solve(N-1);
    print(N);
}
```



3 2 1 1 2 3

N = -3

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



→ Stack overflow

(infinite recursion)