Today's Questions

1. Power 3rd method
2. Print Zig Zag
3. Tower of Hanoi
4. Display Array
5. Display array Reverse
6. Max array

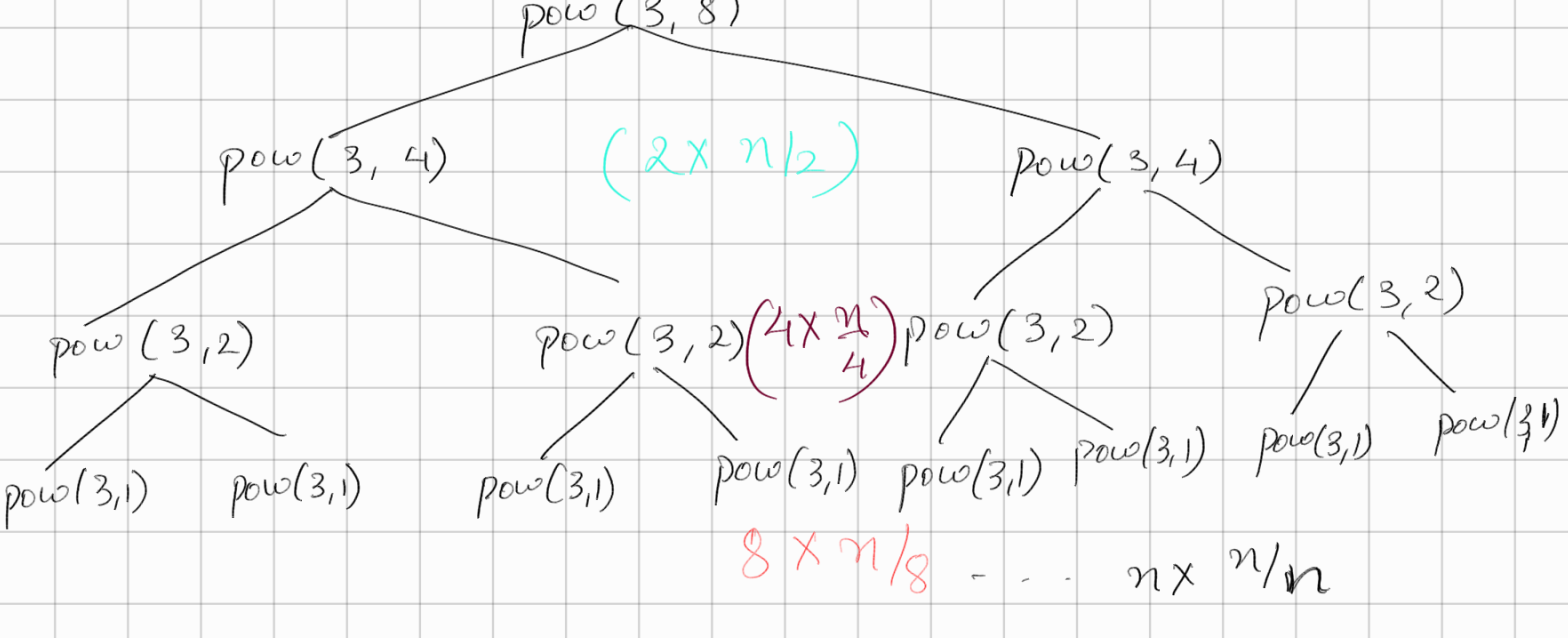
(1) ⭐ Power function

```
pow(x, n){
    if (n==0) return 1;

    int ans = pow(x, n/2) x pow(x, n/2)

    if( n%2==1)
        ans *= x;
    return ans;
}
```

Recursion Tree (for x=3, n=8)



But $\log_2(8)$ is 3,

Here, total number of calls are $2^0 + 2^1 + 2^2 + 2^3 + 2^4 \cdots$

$$\text{GP sum} = \frac{2^5 - 1}{2-1} = \frac{2^5 - 1}{1}$$

$\therefore$ TC: $O(2^5)$ for $n = 8$ (approx: $2^4 \Rightarrow = n$)

$\therefore$ Here $O(n)$ complexity, but that makes the logarithmic logic obsolete. So, for logarithmic power, we only need to call once only

(2) Print Zigzag

for $n=3$: O/P : 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3.

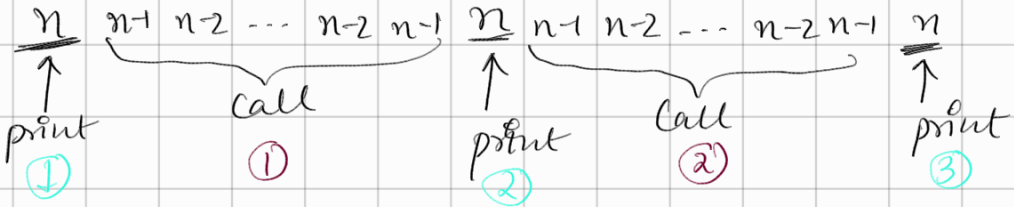
Expectation: pzz(n) : n n-1 n-2 ... n-2 n-1 n n-1 n-2 ... n-2 n-1 n

Faith: pzz(n-1) : _ n-1 n-2 ... n-2 n-1 _ n-1 n-2 ... n-2 n-1 _

Work: Fill blank spaces above with n

no. of blanks = 3, So we need to print 3 times & other stuff blw blanks repeats twice blw 3 spaces for all n values

So, we need to print 3 times & call 2 times alternatively starting with printn



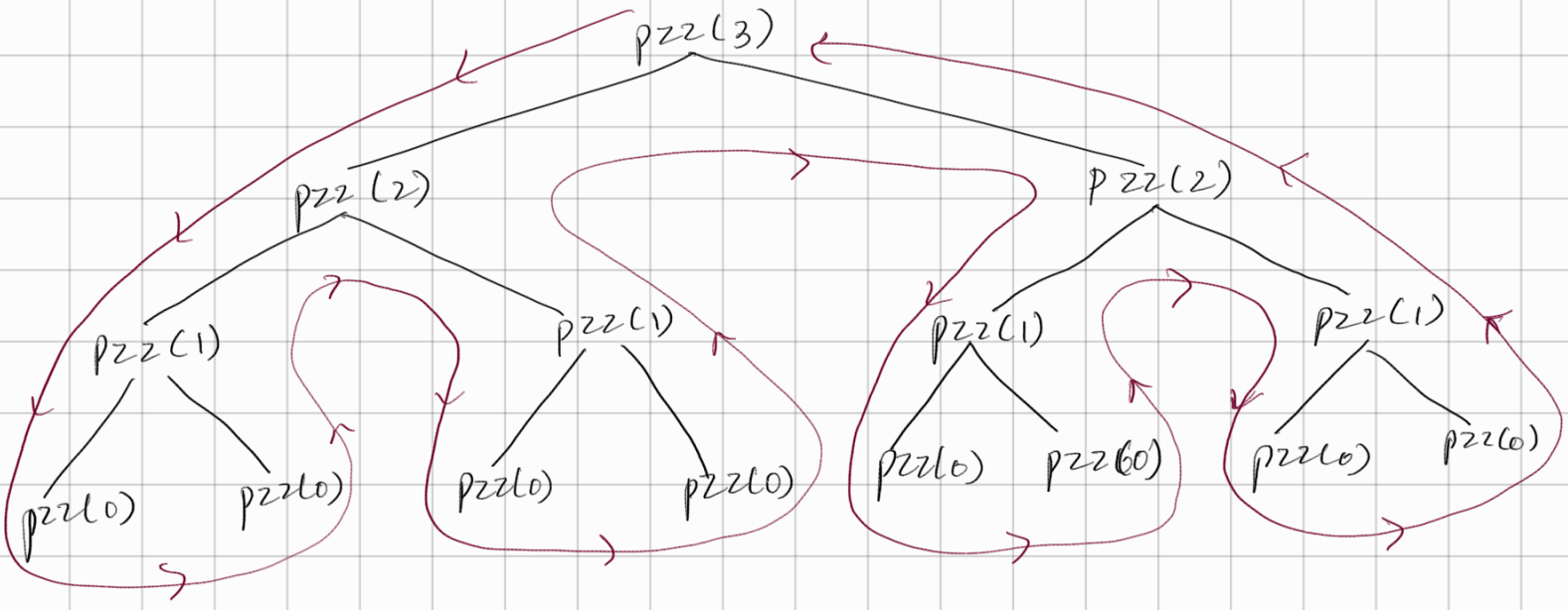
∴ CODE:

```
if (n==0) return;

①   print(n);
①   pzz(n-1);
②   print(n);
②   pzz(n-1);
③   print(n);
```

✱ Base case pzz(0) is hit (4X2) = 8 times → pzz(n-1) / pzz(n-1)

Recursion Tree



Recursion Tree works on Depth First Search concept

Recursion TC formula: => (Calls)^height + preorder x height + postorder x height

$$TC = \left(\text{no. of calls}\right)^{\text{height}} + \left(\text{preorder work} + \text{postorder work}\right) \times \text{height}$$

③ Display array

a) Expectation: dispArr(0) → prints array

b) Faith: dispArr(1) → prints array[1:n-1]

c) Work: print(arr[i])

∴ CODE:

```
if( idx >= arr.length)
    return;
print(arr[idx]);
dispArr(arr, idx+1);
```

④ Display array in Reverse

Similar to ③, but post order work

∴ CODE

```
if( idx >= length)
    return;
dispArr(arr, idx+1);
print( arr[idx]);
```

⑤ Maximum of an Array

a) Expectation: max(arr, 0) : returns max of [0 → n) elements

b) Faith: max(arr, 1) : returns max of [1 → n) elements

c) Work: Compare current element & return

return Math.max(arr[idx], maxArray(arr, idx+1));

∴ CODE:

```
if(idx >= arr.length) return Integer.MIN_VALUE;
return Math.max( arr[idx], maxArray(arr, idx+1));
```

arr = { 3 2 5 7 -2 1 }

idx 0 1 2 3 4 5

max(arr, 6)		returns Integer.MIN_VALUE
max(arr, 5)	4K arr 5	returns 1
max(arr, 4)	4K arr 4 idx	returns 1
max(arr, 3)	4K arr 3 idx	returns 7
max(arr, 2)	4K arr 2 idx	returns 7
max(arr, 1)	4K arr 1 idx	returns 7
max(arr, 0)	4K arr 0 idx	returns 7

∴ max value is returned as 7.

⑥ first index

{ 20, 60, 10, 80, 50, 10, 20, 10 } , x = 10

↑ 2 → ans.

a) Expectation: firstIndex(arr, 0) : returns first index in [0, n)

b) Faith: firstIndex(arr, 1) : returns first index in [1, n)

c) work: check if arr[idx] == x, if true then return idx
else return firstIndex(arr, idx+1).

∴ CODE:

```
if( idx >= arr.length)
    return -1;
if(arr[idx] == x)
    return idx;
return firstIndex(arr, idx+1, x);
```

⑦ Last Index

Similar to first Index but post order work

∴ CODE

```
if (idx >= arr.length)
        return -1;
int y = lastIndex(arr, idx+1, x);
if (y != -1)
    return y;
if (arr[idx] == x)
    return idx;
return -1;
```