Date / /
Page No.

Date → 8-Jan
Topic → 3.3 (Hard level Question)

Q. Pascal Triangle

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
1   5   10  10  5   1
```

Varities of asking Ques^n
1) Given Row & column, tell the element
2) Print any n^th row of pascal △
3) for Given N, print the entire pascal triangle (N=6)

• Variety 1

Row = R, column = C

$\boxed{{}^{R-1}C_{C-1}}$

let R=5
C=3

$^{5-1}C_{3-1} = {}^4C_2 = 6$

$\frac{4!}{(2!)(4-2)!}$

$^7C_2 = \frac{7!}{2! \times 5!} = \frac{7 \times 6 \times 5!}{2 \times 1 \times 5!} = 21$

Brute:

• use long long for these type of problems

```
func NcR (n,r){
    res = 1
    for(i=0; i<r; i++){
        res = res * (n-i)
        res = res/(i+1);
    }
    return res;
}
```

T.C → O(r)
S.C → O(1)

- Variety 2 (N=5)

  N^{th} row ⇝ n elements

T.C ⇝ O(n×r)
S.C ⇝ O(1)

Brute

```
for(c=1 ; c <= n ; c++){
    cout << func NcR (n-1 , c-1) << cout " " ;
}
```

Optimal

Row → 1 5 10 10 5 1

$1 \quad \frac{5}{1} \quad \frac{5}{1}\times\frac{4}{2} \quad \frac{5}{1}\times\frac{4}{2}\times\frac{3}{3} \quad \frac{5}{1}\times\frac{4}{2}\times\frac{3}{3}\times\frac{2}{4} \quad \frac{5}{1}\times\frac{4}{2}\times\frac{3}{3}\times\frac{2}{4}\times\frac{1}{5}$

let ans = 1

$\frac{ans \times (Row - column)}{Column}$

```
func ansRow {
    ans = 1
    cout << ans ;
    for(i=1 ; i < n ; i++){
        ans = ans * (n-i) ;
        ans = ans / i ;
    return ans ;
    }
}
```

T.C ⇝ O(n)
S.C ⇝ O(1)

- Variety 3

Brute

```
for(row = 1 → n){
    temp = [ ] ;
    for(col = 1 → row){
        temp.add(NcR (row-1, col-1) ;
    }
    ans.add(temp) ;
}
return ans
```

T.C ⇝ O(n×n×r)
≈ O(n^3)

Better/optimal

use 2nd type

```
func^n(generate Row){
    ans = 1 ;
    ansRow [ ]
    ansRow.push_back (1) ;
    for (col = 1; col < row ; col ++){
        ans =  ans * (col - row)
        ans = ans / col
        ansRow . push (ans) ;
    }
    return ansRow ;
}
```

T.C → O(N²)

```
main(){
    ans = [ ] ;
    for(i = 1 → n){
        ans.push_back(ansRow) ;
    }
    return ans ;
}
```

Q- Majority element (> ⌊N/3⌋ times)

$A = [1,1,1,3,3,2,2,2]$ $N=8$

Brute

```
ls = [];
for (i=0 → n-1){
    if (ls.size()==0 || ls[0]! = nums[i]){
        cnt = 0;
        for (j=0→n-1){
            if(nums[j]== nums[i]){
                cnt ++;
            }
            if (cnt > n/3){
                    ls.add(nums[i]);

        }
    if (ls.size()==2) break;
}
return ls;
```

T.C → $O(n^2)$

S.C → $O(1)$

Better   using STL → unorder_map

~~[illegible]~~   (See the soln from leetcode / Vscode)

here when we check for mp[nums[i]] > target

→ we dont use for(0 → n)

bcz that will add duplicate element

→ we use for(auto it : mp)

cnt1 = 0   cnt 2 = 0
el 1   el 2

Moore voting algo (for n/3)

optimal:

```
for (i=0 → n-1){
  if (cnt1 == 0 && nums[i] != el2){
     cnt1 = 1 , el1 = nums[i];
  }
  else if (cnt2 == 0 && nums[i] != el1){
   cnt2 = 1 , el2 = nums[i];

  }
  else if (el1 == nums[i]) cnt1 ++ ;
  else if (el 2 == nums[i]) cnt2 ++;
  else {
     cnt1-- , cnt2--;

  }
}
```

T.C ~ O(n)
S.C ~ O(1)

Manual check el1 & el2

```
vector<int> ls;
cnt1 = 0 , cnt2 = 0
for (i = 0 → n){
   if(el1 == nums[i]) cnt1 ++;
   if(el2 == nums[i]) cnt 2 ++;
}
mini = n/3 + 1
if(cnt1 >= mini) ls.push(el1)
if(cnt2 >= mini) ls.push(el2)
sort (ls.begin(), ls.end());
return ls;
```

Note: Watch 3sum and 4sum from striver later
Count inversion and reverse pairs

Q- 3sum

a[i] + a[j] + a[k] = 0 {s.t. (i != j != k)}

- order of triplet does not matter

A = {-1, 0, 1, 2, -1, -4}

Brute:

```
set< > st;
for(i=0 ; i<n ; i++){
    for(j=i+1 ; j<n ; j++){
        for(k=j+1 ; k<n ; k++){
            if(arr[i] + arr[j] + arr[k] == 0){
                temp = {nums[i], nums[j], nums[k]};
                sort(temp.begin(); temp.end());
                st.insert(temp);
            }
        }
    }
}
ans = {}
ans(st.begin() ; st.end());
return ans;
```

T.C → O(n³ × log(no. of unique triplet)

S.C → O(no. of triplet) × 2

Optimal:

Sort array first

A = {-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2}

↑ i ↑ k

-2 + ~ + 2 = 0

We have to move j in order to find

Resultant triplet is always in sorted order

When j > k → stop
move i

Note: From now, Codes will be on VScode
just write approach/imp piece of code

Date / /
Page No.

Q- Largest subarray with sum 0

$A = \{1, -1, 3, 2, -2, -8, 1, 7, 10, 23\}$

Brute: Generate all subarrays and check which has sum = 0

T-C → $O(n^2)$

Optimal: using prefix sum stored in map

Sum = S

Sum = S   Sum = 0

see the code from Vscode

(-4, 6)
(-5, 5)
(5, 3)
(3, 2)
(1, 0)

<key, value>

<prefix sum, index>

Sum = 0

1, -1, 3, 2, -2, -8, 1, 7, 10, 23

maxi = ~~1~~ ~~2~~ 5

→ when we move to 3, we can't see it in hashmap, so insert it

→ When sum = 5 and we add -2 it becomes 3 and we know 3 is there in hashmap
this means (3,2) at index we have sum = 3 and we are at 4th index

Sum = 3

0 1 2 3 4

Sum = 3   Sum = 0   len = 4-2 = 2

We will not update this (3,4) in hashmap

bcz we only add only once

→ When Sum (-4) and we add 7 it becomes 3 and it is there in hashmap

Sum = 3

0 1 2 3 4 5 6 7

Sum = 3   Sum = 0 → len = 5 (we update maxi)

sum = ~~1~~ ~~0~~ ~~3~~ 8
~~3~~ ~~-8~~ -4

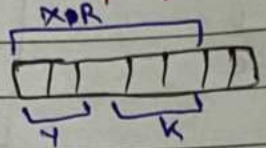
Q- Count no. of Subarray with given XOR as K

$A = \{4, 2, 2, 6, 4\}$ $k = 6$

Brute : Generate all subarrays and those with XOR = 6 we inc. the counter and return it

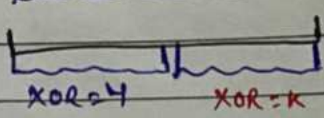
T.C ~> $O(N^2)$
S.C ~> $O(1)$

Optimal : We will use prefix XOR



$y \wedge k = XOR$
$y = XOR \wedge k$

prefix XOR = XR



$y \wedge k = XR$
$y = XR \wedge k$

we need to calculate no. of y

Dry run XOR = 0 Cnt = ~~0~~ ~~1~~ ~~3~~ 4 we will use a hashmap to store prefix XOR as key and counter as value

[4, 2, 2, 6, 4]

~> first when we are at 4, XOR = 6, but not equal to k
move it in hashmap

~> XOR = 4^2 = 6, Cnt = 1

~> 6^2 = 4, but already in hashmap, increase the counter

~> 4^6 = 2,

(2,1)
(6,1)
(4,2)

T.C ~> O(NlogN)

Note: if we take {1,18} we have some left places and we took it that's why we took that output

Date / /
Page No.

Q- Merge overlapping subintervals
{(1,3), (2,6), (8,9), (9,11), (8,10), (2,4), (15,18), (16,17)}
output → {(1,6), (8,11), (15,18)}

Brute: first sort it from pairs
(1,3)(2,4)(2,6)(8,9)(8,10), (9,11)(15,18), (16,17)

(1,3) (2,4)
we check last pair 2nd element and check current pair 1st element
3 2 we can see they are overlabbing
((1,3)(2,4)(2,6)) → (1,6)
(2,6) (8,9)
6 8 not overlabbing

T.C → O(NlogN) + O(2N)
S.C → O(N)

(8,9)(8,10)(9,11) → (8,11)

(15,18)(16,17) → (15,18)

(1,6) (8,11) (15,18)

Better: we will go with a single iteration and check if next pair is overlabbing, we call it visited and part of it and move to next then again do same

But when we reach where pair is not a part of it we stopped and move our pointer to 2nd position & we will get to know that it is already visited. We do the same until we reach where it is not a part of it and then start a new interval from new on

T.C → O(NlogN + N)
S.C → O(N)

Note: in interview, this "without space" is not given so for simplest solⁿ we take a 3rd array. and place 1,1 pointer on both array and which one is smaller we insert and move that pointer

$A3 = [0\ 1\ 2\ 3\ 5\ 6\ 7\ 8\ 9]$

Q- Merge two sorted arrays without extra space
arr1[] = [1 3 5 7]   arr2 = [0 2 6 8 9]
output : [0 1 2 3 5 6 7 8 9]

Optimal: We compare last element of arr1 and first element of arr2[] and then swap them then we move arr1 pointer left and arr2 pointer right and again compare them and swap but at the next moment it looks like 3 6
and we know 3 < 6 so we dont do anything just kept them where they are and stop iterating bcz from nowonwards all will be at correct place

now we just sort both array
and it will be the answer

T.C ⇝ $O(\min(n,m)) + O(n\log n) + O(m\log m)$
S.C ⇝ $O(1)$

Q- Find the repeating and missing number from 1 to N
arr = [4, 3, 6, 2, 1, 1]   here n=6
output : {1, 5}

Brute: Pick all numbers from 1 to 6 and check in the array with how many time it appears. if >1 then no. is repeating and if ==0 means its missing

```
Repeating=-1 , Missing=-1
for(i=1→n){
    cnt=0
    for(j=0→n-1){
        if(arr[j]==i){
            cnt++);
    }
    if(cnt==2) repeating=i;
    else if(cnt==0) missing=i
    if(repeating!=-1 && missing!=-1)
        break;
}
```

T.C ⇝ $O(n^2)$
S.C ⇝ $O(1)$

Note: learn about hash array, how to declare & use it.

Date / /
Page No.

Better: we will take a hash array of size $(n+1)$ & initialize everyone with 0 and when we iterate we keep updating the count of every number

0	1	2	3	4	5	6
0	~~0~~ 1	~~0~~ ~~1~~ 2	~~0~~ 1	~~0~~ 1	0	~~0~~ 1

$n = 7$

now check (1 to 6)
we see 5 has 0 counts and 1 is repeating

$T.C \rightsquigarrow O(2N)$
$S.C \rightsquigarrow O(N)$

Optimal:
$x \rightarrow$ repeating
$y \rightarrow$ missing

$SN \rightsquigarrow$ sum of first n natural number $\left(\frac{n(n+1)}{2}\right)$

$S \rightsquigarrow$ Sum of given number

$S2 \rightsquigarrow$ Sum of Square of given no.

$S2N \rightsquigarrow$ Sum of Square of first n natural no. $\left(\frac{n(n+1)(2n+1)}{6}\right)$

$$S - SN = (1) - (5)$$
$$= -4$$

$$x - y = -4 \quad - (I)$$
$$x + y = 6 \quad - (II)$$

$$x = 1$$
$$y = 5$$

$$S2 - S2N = 1^2 - (5^2)$$
$$= -24$$
$$x^2 - y^2 = -24$$
$$(x+y)(x-y) = -24$$
$$(x+y)(-4) = (-24)^6$$

Q- Find the Maximum product subarray

$A[\,] = \{2, 3, -2, 4\}$ output = 6

Brute: Generate all subarray and see which has max. product

$T.C \rightsquigarrow O(n^2)$
$S.C \rightsquigarrow O(1)$

Note: Whenever you see subarray, the brute always be by generating all the subarray

Date / /

Page No.

Optimal: {2, 3, -2, 4}

observation

1) if all +ve → product all
2) if even -ve → product all
3) if odd -ve → we will not take the highest negative element (like -1 etc) we take -6, -4 instead of -1 we take even negative and rest all +ve and multiply them (except 0)
4) when you find zero make separate-separate subarray

maxi = ~~INT_MIN~~ 2 6

prefix = ~~2~~ ~~6~~ ~~-12~~ -48

Suffix = ~~4~~ -8 ~~-24~~ -48

T.C → O(N)

S.C → O(1)

either my answer is prefix or suffix

if i ignore this

{2, 7, -1, 3, 5, -2, 4, -6}

prefix

suffix

we see that our answer is

Note: Ques^n like → 4sum, → ~~count inversion~~, → Reverse pair are left

Will do it direct in vscode