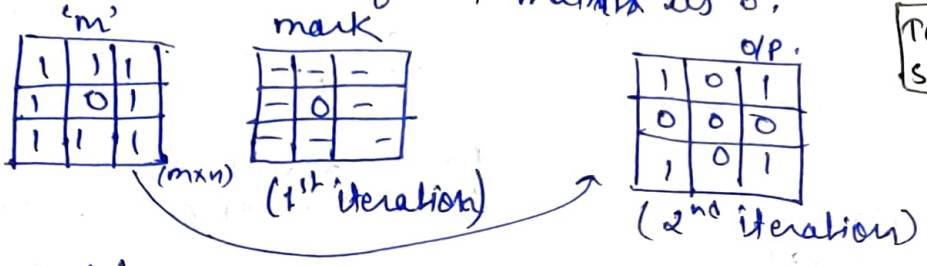
Day - 1

① **Set Matrix zeroes :-** Given m×n matrix, if an ele. is 0, set its entire row & column to 0. (inplace)

i) **Brute force :-** → Take another matrix & keep track of all zeroes → now on actual matrix, if you find 0 in temp, mark all rows & columns of that matrix as 0.

eg:-



TC :- $O(n^2)$
SC :- $O(n^2)$

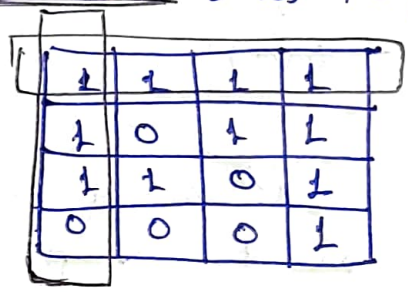
ii) **optimised :-** Take two 1D array of length 'm' & 'n', array → row[m] & col[n]

1st iteration → if m[i][j] = 0 → mark ⇓ row[i] = 0 & col[j] = 0.

IInd iteration → Traverse of matrix 'm' & if (row[i] == 0 or col[j] == 0) → mark 'm[i][j] = 0'

TC :- $O(n^2)$ SC :- $O(n)$

iii) **Most optimal :-** Instead of taking 2 array, use 0th row & 0th column as keeping track of 0's. (Algo)



mark = true

if in 0th col any value is 0. then mark m[0][0] = 0 or in jth col. if → 0 → m[0][j] = 0

** & if in 0th col any value is 0, then mark → mark = false.

if on ith row → 0 m[i][0] = 0

→ on 2nd iteration. traverse (from back) on matrix, if (m[0][j] == 0 or m[i][0] == 0) → m[i][j] = 0.

& for m[0][0] → if (mark == false ~~m[0][0] = 0~~) ↓ m[i][0] = 0

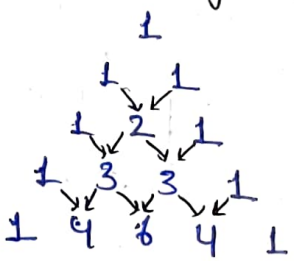
(Don't traverse 0th col)

TC :- $O(n^2)$
SC :- $O(1)$

** Key point
On 2nd iteration, start filling from back, to keep matrix unaffected -

(ii) **Pascal's triangle** :— Given 'n', give pascal triangle

n=5



① → if θ^n is give value of pascal traingle at r=5 & c=3

formula → $\boxed{{}^{R-1}C_{c-1}}$

$${}^nC_r = \frac{n!}{(n-r)!\,r!} \Rightarrow {}^4C_2 = \frac{4!}{2!\,2!} = \frac{4\times3\cdot\cancel{2}}{\cancel{2}\times1} = \boxed{6}$$

② if θ^n is give any one row of pascal triangle.

→ using formula :—

TC :— $O(n^2)$

because, for calc. one value TC = O(n) for cal, n values n^2

$\boxed{TC :- O(n)}$ (for calc. factorial)

Better Approach :— use this technique.

so, instead of calculating ^nC_r, we'll, if we calc. 5th row,

$${}^5C_3 = \frac{5\times4\times3}{3\times2\times1}$$

(3) = no. of ele. above & down

$${}^4C_0 = 1,\quad {}^4C_1 = \frac{4}{1};\quad {}^4C_2 = \frac{4\times3}{1\times2}$$

$${}^4C_3 = \frac{4\times3\times2}{1\times2\times3};\quad {}^4C_4 = \frac{4\times3\times2\times1}{1\times2\times3\times4}$$

for k=4, res=1

Algo

```
for( i=0 ---- i<k ){
    res *= (n-i);
    res /= (i+1);
    cout<< res << " ";
}
```

← optimised way to calc. value in pascal triangle for 1 row.

$\boxed{TC :- O(n)}$

③ for calc. whole triangle. (ith row has ith no. of ele's)

1st & last ele of every row are 1, be p[0][0] = 1

for ele. other than start and end of a row.

$$p[i][j] = p[i-1][j-1] + p[i-1][j]$$

$\boxed{TC :- O(n^2)}$

(iii) Next Permutation

:- Given Array, return next lexicographically greater permutation of it.

eg:- '132'
o/p:- 213

123
132
213
231
312
321

① **Brute force**:- store all combo's, & return next to given ~~com~~ one, if given one is last permutation return 1st permutation.

TC:- O(n! * n) → n → no. of digits in n!
SC:- O(n!) → for storing all combo's.

② **Algo (optimised)**

eg:- 1 3 5 4 2
0 1 2 3 4
k i

→ on 1st iteration. for(k = n-2 —— k>=0) {
if(arr[k] < arr[k+1]) break;
}
got k=1

→ on 2nd iteration → for (i = n-1 —— i > k){
if(arr[i] > arr[k]) break;
}
got i = 3

→ swap(arr[k], arr[i])
→ reverse(arr.begin()+k+1, arr.end());
→ **Base case**:- If we get k<0 ⇒ (no break point)
so, after 1st iteration, just reverse arr & return.

eg:- 5 4 3 2 1 (5 4 3 2 1)

o/p:- 1 2 3 4 5

TC:- O(n)
SC:- O(1)

Intution Behind Algo:-

→ In dictionary order, no. from back will always increasing till some index, atleast for 1 index eg: 1 2 [3].
for 2 index eg:- 1 3 2 (1 3 2↑)
or in 1 3 5 4 2 (1 3 5 4 2, 1 3 2↑)

break point, so, 1 3 | 5 4 2

for, this we need to find Prefix need to be greater, just next greater than 3, so to find it we'll again traverse from back & find next greater ele. than 3

→ So, ~~swap~~ swap 3 & 4 → 1 4 5 3 2, now right side lies 532, which is sorted in increasing order from back.
→ To get exact next permu., we need to make remaining ele. as min. as possible, As it is already sorted in decreasing order from front so will reverse the remaining portion. (1 4 2 3 5)

(iv) Maximum Subarray Sum (Kadane's Algorithm)

eg:- [-2, 1, -3, 4, -1, 2, 1, -5, 4] o/p:- 6 → {[4, -1, 2, 1]}

① **Brute force:-** Consider all subarray, sum them, take max among a

TC:- $O(n^3)$
SC:- $O(n)$ → for storing subarray's

② **Improvised:-** Instead of summing every subarray, take a variable sum, & update as i & j changes

TC:- $O(n^2)$
SC:- $O(1)$

③ **using prefix sum:-** make prefix sum array. and

```
for(i=0 —— n) {
    for(j=i —— n) {
        cur = i>0 ? PS[j] - PS[i-1] : P[j];
        mx = max(mx, cur);
    }
}
```

TC:- $O(n^2)$
SC:- $O(n)$

④ **Optimal Approach/Algo:-**

eg:- {-2, -3, 4, -1, -2, 1, 5, -3}

Sum = ∅ -2 ∅ -3 ∅ 4 3 1 2 7 4

mx = INT_MIN
-2
4
7

```
{
    mx = INT_MIN
    sum = 0
    for(auto it : nums) {
        sum += it;
        mx = max(mx, sum);
        if(sum<0) sum = 0;
    }
    return mx;
}
```

TC:- $O(n)$
SC:- $O(1)$

(V) Sort an array of 0's, 1's & 2's :

① Brute force :— Simply sort array, TC :- O(n log n) → merge sort
SC :- O(1)

② Better Approach :— Using 2 loop, on first loop calculate no. of 0's, 1's & 2's using 3 variables, & after that insert that many time 0, 1, 2 in array.

TC :- O(n) + O(n) ≃ O(n)
SC :- O(1)

③ Algo (Using Variation of Dutch National flag Algorithm)
(using only 1 loop) 3 pointer's (low, high, mid)
low = 0, mid = 0, high = n-1

while (mid <= high)
- if (a[mid] = 0) → swap (a[low], a[mid]), low++, mid++
- if (a[mid] = 1) → mid++
- if (a[mid] = 2) → swap(a[mid], a[high]) high--;

So, after loop gets over, [0 — low-1] → 0
[low — mid-1] → 1
[mid — n] or [high+1 — n] → 2

TC :- O(n)
SC :- O(1)

<vi> Best time to Buy and sell stock. (Buy once, sell once)

① **Brute force**:- we can store minimum array, keep minimum till now & on next iteration and on next ~~loop~~ iteration we'll check if we can make max profit.

TC:- $O(n) + O(n) \simeq O(n)$
SC:- $O(n)$

② **Optimised**:- Instead ~~of~~ keeping minimum array, we'll simply keep minimum variable to store min, till now.

```
{ mini = p[0];
  mxpro = 0
  for(i=1 —— n) {
        mxpro = max(mxpro, (price[i] - mini));
        mini = min(mini, price[i]);
  }
  return mxpro;
}
```

TC:- $O(n)$
SC:- $O(1)$