NOTES DATE

Striver SDE Sheet: Day 1 - Arrays ::

A] Set matrix zeros :-

A-1] Brute Force :-

1. Iterate on matrix
2. we cannot mark all row & colm = 0 for arr[i][j] = 0 because, further iterations on next ele. will be in problem. so mark them all col. & all row for a[i][j] = 0 to -1.
   Only mark non-zero terms to -1.

$$\begin{bmatrix} 1 & \cancel{1}^{-1} & \cancel{1}^{-1} & 1 \\ \cancel{1}^{-1} & \boxed{0} & \boxed{0} & \cancel{1}^{-1} \\ \cancel{1}^{-1} & \cancel{1}^{-1} & \boxed{0} & \cancel{1}^{-1} \\ 1 & \cancel{1}^{-1} & \cancel{1}^{-1} & 1 \end{bmatrix}$$

3. Now convert $-1 \to 0$

A-2] Two additional arrays

- TC $\to$ $\underbrace{n^2}_{\text{iterating}} \times n$ $\to$ + $\to$ iterating entire r & c for each '0'.
- so mark entire row & entire column even if 1 '0' is found
-

Colm [m] → initialized by 0, marked '1'. i.e. entire col. must become 0 later

	0	~~0~~ 1	~~0~~ 1	0
0	1	1	1	1
1 ~~0~~	1	0	1	1
1 ~~0~~	1	1	0	1
1 ~~0~~	1	0	0	1

row(n)

- Now reiterate, and change ele. of matrix to 0 by looking at row & colm array mark.

1	0	0	1
0	0	0	0
0	0	0	0
0	0	0	0

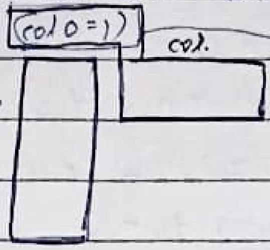
NOTES

DATE

A-3] using row[0] & col[0]

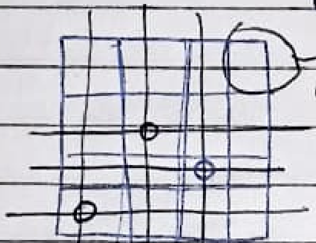
- Instead of taking row & col outside, move it inside the matrix.
- In this case, there is one common pt.

so take it like:-



This part was colliding so, we put in into variable.

•)



this will remain unmarked (in this case)

corner guy will not be converted.

•

$\boxed{1}$ 0   c

1	~~1~~ 0	~~1~~ 0	1
~~1~~ 0	0	1	1
~~1~~ 0	1	0	1
0	1	1	1

r

→ Now iterate

→ Now converting all '1' of matrix who should be '0'

→ Now arr[0][0] = 1 → r = 1 & c = 0

so this should be converted to 0

This will arr[0][3] = 1 → r = 0 & c = 1

so this will be converted to 0. ✗

→ so do not touch the r & c arr. elements, we'll deal with them later.

→ so iterate from (n-1, m-1) reverse & work on ①

→ $\boxed{0}$ var.

1	0	0	1
0	0	~~1~~	0 ~~1~~
0	~~1~~ 0	0	0 ~~1~~
0	~~1~~ 0	~~1~~ 0	0 ~~1~~

→ ②

→ ①

③

this ele ans dep on → [0][3] : itself
→ [0][0] : 1 ($\underbrace{[0][0]:1}_{r\ element}$)

→ so we'll solve ② then ③ as ② ele. is dep. on ③

this ele. ans dep on → [0][0] : itself
→ variable taken

08 4 17

NOTES DATE

B] Pascal's Triangle:-

n	n+1
0	1
1	2
2	3
3	4
4	5
5	6

$\lceil n \rightarrow k$ (idx)

$n \rightarrow$ levels

$k \le n$

$k \in [0, n]$

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

n → this for maths

n+1 → use this mostly (code) *

- Properties:-
  1) Each entry in ~~binomial~~ Pascal's Triangle is binomial coefficient of expression: $\boxed{(x+y)^n \; ; \; C(n,k) = \frac{n!}{k!(n-k)!}}$

     e.g:- $C(4,0)=1$, $C(4,1)=4$, $C(4,2)=6$

     $k=4$, $(x+y)^4 = 1x^4 + 4x^3.y + 6x^2y^2 + 4x.y^2 + 1y^4$
  2) Pascal Triangle is symmetric : $\boxed{C(n,k) = C(n, n-k)}$
  3) Sum of elements in a row (n) : $\boxed{\sum_{k=0}^{n} C(n,k) = 2^n}$
  4) Sum of a diagonal set of numbers equals the next number in the next diagonal:-

     $\boxed{C(r,\underline{k}) + C(r+1,\underline{k}) + \ldots\ldots + C(n,k) = C(n+1, k+1)}$

     2 1, 3 3, 4 6, 5 10
  5) Sum of shallow diagonals gives Fibonacci Numbers:

     Pascal's Triangle follow: $\boxed{C(n,k) = C(n-1,k-1) + C(n-1,k)}$

     Fibonacci sequence follows: $F(n) = F(n-1) + F(n-2)$

     $\boxed{F(n) = C(n-1,0) + C(n-2,1) + C(n-3,2) + \ldots}$
  6) Each row represents power of 11 : $\boxed{11^n}$
  7) Path counting in a grid:-

     → No. of ways to reach a specific point in a grid only using right & down movement.

     → $C(\underbrace{m+n}_{\text{total}}, m) = \frac{(m+n)!}{m!\,n!}$ = No. of ways to reach (m,n)

     $(0,0) \rightarrow (m,n)$ → right, → down

PAGE

NOTES DATE

• 3 Types of Problem's on Pascal Triangle:-

① Given row & column, find element.

② Print any n^{th} row of Pascal Δ.

③ Print entire Pascal Δ for given n.

B-1] ①

• Element = $^nC_r = n! / r!(n-r)!$

$$= \frac{n(n-1)(n-2)(n-3)\ldots(n-r)!}{((r)(r-1)(r-2)\ldots 3.2.1)(n-r)!}$$

$$^7C_2 = \frac{7\times 6}{\underbrace{2\times 1}_{2\ \text{times}}}, \quad ^{10}C_3 = \frac{10\times 9\times 8}{\underbrace{3\times 2\times 1}_{3\ \text{times}}} = \frac{10}{1}\times\frac{9}{2}\times\frac{8}{3}$$

• Code :-

TC = O(r)

SC = O(1)

```
find nCr (int n, int r){
    long res = 1        → to avoid overflow
  ★ for(int i=0; i<r; i++){
        res = res * (n-i);
        res = res/(i+1); }
    return res; }
```

B-2] ②

• $\boxed{N^{th}\text{ row} \rightarrow N \text{ elements}}$

• $\boxed{\text{element} = {}^{R-1}C_{C-1}}$ → R & C are row & colmn no. of element (1-idx)

• r = 6 →

0	1	2	3	4	5
1	5	10	10	5	1

c=1, 2, 3, 4, 5

ans = 1, $\boxed{\frac{5}{1}}$, $\frac{5\times}{1}\boxed{\frac{4}{2}}$, $\frac{5\times 4}{1\times 2}\times\boxed{\frac{3}{3}}$, $\frac{5\times 4\times 3}{1\times 2\times 3}\times\boxed{\frac{2}{4}}$

→ using 0-based idx of column ★

$$ans = ans \times \frac{(row - col.)}{col.}$$

NOTES DATE

- code:-

```
ans = 1
print(ans)
for(i=1; i<n; i++){
    ans = ans x (n-i)
    ans = ans/(i)
    print(ans); }
```

TC = O(N)
SC = O(1)

no. of row = no. of column elements we print

*B-3] 3

- no need to print spaces
-

```
public static List<Integer> generateRow(int row){
    long ans=1;
    List<Integer> ansRow = new ArrayList<>();
    ansRow.add(1);   //inserting first element
  // calculate rest of elements
  for(int col=1; col<row; col++){
      ans= ans*(row-col);
      ~~ans = ans*(row-col);~~
      ans= ans/col;
      ansRow.add((int)ans); }
  return ansRow

public static List<List<Integer>> pascalTriangle(int n){
    List<List<Integer>> ans = new ArrayList<>();
    //store entire pascal Δ
    for( int row=1; row<=n; row++){
        ans.add(generateRow(row)); }
        return ans; }
```

TC = O(n * n)   SC = O(1)

- generating entire row
- for n row

NOTES DATE

c] Next Permutation:- ⓡ

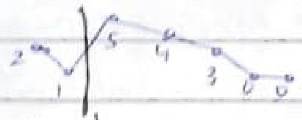
c-i] Brute force:-

- Generate all permutation (sorted order) → Recursion $O(N!)$
  Linear search for given perm. → $O(N)$
  go for next idx perm., if it dne then go to first idx.
  $TC = O(N!N)$, $SC = O(1)$

c-ii] optimal:-

- Longer prefix match (raj > rax > rha)
  (next elements in lexo. order)

arr[] = [2, 1, 5, 4, 3, 0, 0] — initial per. [n-2 is the last idx] we can see break pt., if it exist



(i) (i+1) → Break pt.

✓ (dip) [1 2 3 4 | 5] × (no dip) [5 4 3 2 1] → final perm.

→ find break pt. : $a[i] < a[i+1]$

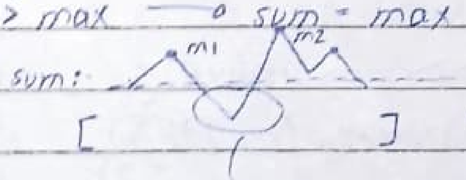
- find $arr[j] > arr[i]$ s.t. $arr[j]$ = min. ($i+1 \to n-1$ ele.) but > $arr[i]$ → swap

[2 1 | 5 4 3 0 0]

- [2 3 0 0 1 4 5] → keep them in sorted order (min. first)

D] Kadane's Algorithm:- ⓡ

- maximum Subarray Sum in an Array.
- Approach-3:-
  → If at any idx (sum < 0) → sum = 0. (as it only reduce sum)
  → If sum > max → sum = max



↓ sum < 0

→ neglect → sum = 0 (reinitialize)

m2 ans

→ for (i = 0 to n) sum += arr[i]

2 conditions.

PAGE

→ To print the max sum subarray (can be more than one)

- → if (sum==0) start=i ; → we are having new start
- → if (sum > max) $\underbrace{\text{ansStart = start, ansEnd = i;}}_{\text{this keeps track of subarray}}$

E] Sort an array of 0s, 1s & 2s :- (Optimal Approach)

- An array contains only 0s, 1s, 2s. Sort the array in-place.
- Dutch National Flag Algorithm :- TC = O(N) , SC = O(1)
  → sorting one ele at each step
- Use 3 pointers :- low, mid, high
- 3 Rules :-
  - [0, ... low-1] → 0 extreme left
  - [low, ... mid-1] → 1 ~~middle~~
  - [high+1 .... n-1] → 2 extreme right
- Rem.

  0 .... low-1, low ...... mid-1, mid ...... high, high+1, ... n-1

  $\underbrace{0\ 0\ 0\ 0\ 0}_{\text{sorted}}$ $\underbrace{1\ 1\ 1\ 1\ 1}$ (unsorted) $\underbrace{2\ 2\ 2\ 2\ 2}_{\text{sorted}}$

  → (first 1 at low ptr)

- arr[] = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0] (low and mid at index 0, high at index 10)

→ initially unsorted

→ In terms of a[mid] (also write in-terms of a[high] as they are part of unsorted arr)

→ arr[mid] = 0 / 1 / 2

→ arr[mid] = 0

swap (a[low], a[mid]) low++, mid++;
size of leftmost region increases

arr[mid] = 1 → mid++

arr[mid] = 2
size of rightmost region ↑.

swap (a[mid], a[high]) high--;

- Dry Run :- above arr

low = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

mid = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ 9

high = ~~10~~ ~~9~~ ~~8~~

→ mid > high ⇒ arr is sorted

NOTES DATE

E] Stock Buy & Sell :-

- Given: arr[] prices → prices[i] : price of a stock on i^{th} day.
- maximize profit, by choosing a single day to buy stock & choose a different day to sell stock