Striver SDE Sheet : D3 - Arrays :-

A] Search in a sorted 2D matrix :- (skip)

- Given : 2D array 'mat' of size N×m, where 'N' & 'm' denote number of rows & columns, respectively. The elements of each row are sorted in non-decreasing order. moreover, the first element of a row is greater than the last elements of the previous row (if it exist). You are given an integer 'target', & your task is to find if it exist in the given 'mat' or not.
- → Covered in BS notes. &

B] Pow (x, N) :- (skip)

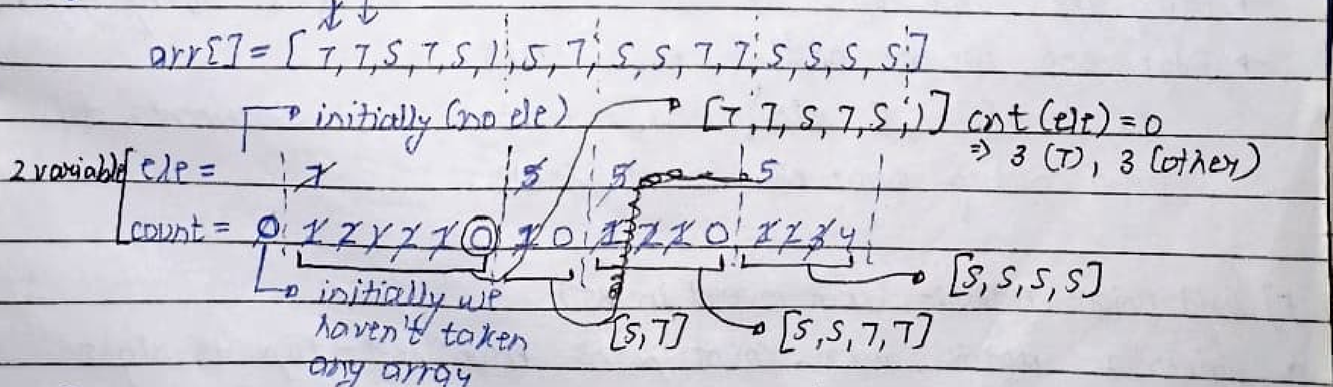
- Implement pow(x, N).
  x → double, n → Integer, calculate x^n

C] Find the majority element that occurs more than N/2 times :-

- Given an array of N integers, write a program to return an element that occurs more than N/2 times in the given array. You may consider that such element always exists in the array.
- Approach - 3 :- moore's voting Algorithm :-

*** → Intuition & thought process of all algo. is asked in Interview.

→



arr[] = [7, 7, 5, 7, 5, 1, 5, 7, 5, 5, 7, 7, 5, 5, 5, 5]
→ initially (no ele)
→ [7, 7, 5, 7, 5, 1] cnt (ele) = 0
⇒ 3 (7), 3 (other)
2 variable
ele = 7 5 5 5
count = 0 1 2 1 2 1 0 1 0 1 2 1 0 1 2 3 4
→ initially we haven't taken any array
[5, 7]
→ [5, 5, 7, 7]
→ [5, 5, 5, 5]

↳ considering initially (7) as our ele. [while iterating if we get more 7s ↑count, else ↓count)

↳ At the end of iteration, we get our majority ele. as

5. It survives till the end with count>0.

↳ If there exist a majority ele. (it will be 5),
stored ele. at end.

But if que. does not state that there exist a majority elee, then we need to check if stored ele. is majority elee or not. If not then array does not contain any majority element.

→ In the process:

- ↳ If array contains a majority ele, its occurance must be greater than floor (N/2)
- ↳ count of minority ele. & majority ele. is equal up to certain point in an array (0 count at ~~varios~~ various pts.)

→ Moore' Algo. guarantee that: If a majority element exist, it will always be final candidate ele, no matter how the elements are arranged (As Freq.(maj. ele.) > N/2 (floor))

D] Find the elements that appears more than N/3 times in the array:-

- Given an int[] array of size N. Find ele(s) that appears more than N/3 times in an array. If no such element exists, return an empty array.
- Approach-3:- Extended Boyer Moore's Voting Algorithm:-
  - → Using the same logic of cancellation as Moore's Voting Algo.
  - → Edge case:- We are adding extra checks:-
    el2 != v[i] , & el1 != v[i] in the first statements to avoid adding same ele. in el1 & el2.

E] Grid Unique Paths:- (Skip, covered in BS)

- Given a matrix m×n, count paths from left-top to right bottom of a matrix with the constraints that from each cell you can either only move to rightward direction or downward direction.

F] ★★★ Count Reverse Pairs:-

- int[] nums

  Return count of reverse pairs(i<j & arr[i] > 2*arr[j]

- Approach-2:- (modified merge sort)

→ arr[i] > 2* arr[j]

→ why same logic as count inversions won't work?

↪ Condition: arr[i] > 2*arr[j], introduces two main problems:-

i) cannont use simple merge condition.

→ In merge sort, we merge based on values, comparing arr[i] $\overset{>}{\&}$ arr[j] to maintain order

→ For Reverse pair, arr[i] > 2* arr[j] is a comparison we cannot use for sorting. merging using this condition won't give a sorted array.

ii) Assymetry in comparison.

→ Inversion count : $\underbrace{arr[i] > arr[j]}_{i<j} \Rightarrow \underbrace{arr[k] > arr[i]}_{k<i}$

(Transitive Rel.)

→ Reverse pairs : The relation given is not transitive in some way

→ count is during merging → count before merging

$\cancel{i}$ $\cancel{i}$ $\cancel{i}$ $\cancel{i}$ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

→ [6, 13, 21, 25] sorted [1, 2, 3, 4, 4, 5, 9, 11, 13] sorted

6 → [1,2] → 2 → sum = ans

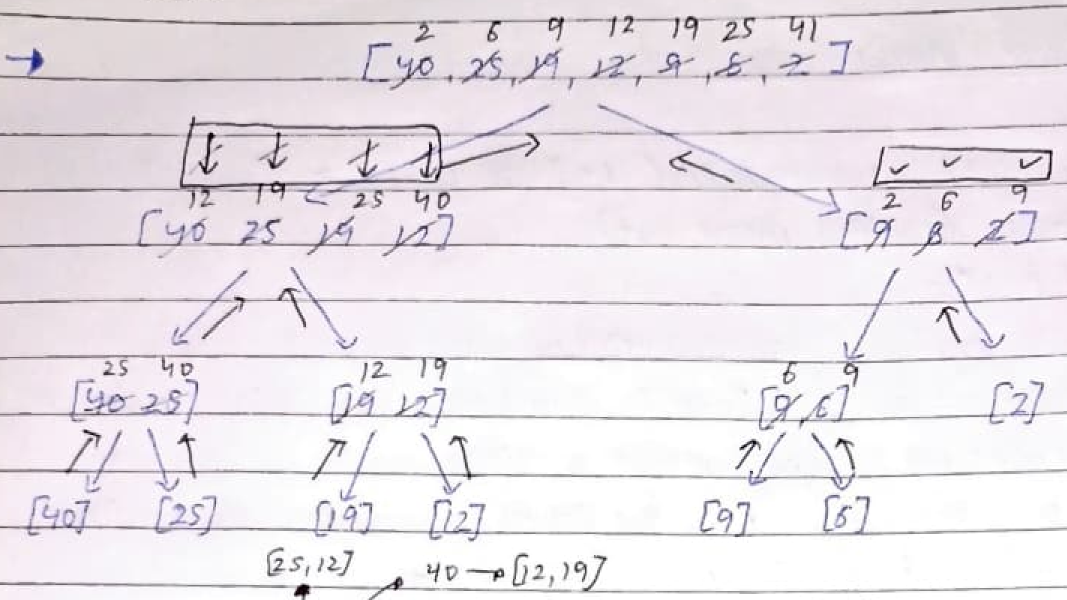
13 → [1,2,3,4,4,5] → 6

21 → [——— ,9] → 7

25 → [——— ,11] → 8

→ The initial given arr. is unsorted, during merge sort we get sorted parts.

→



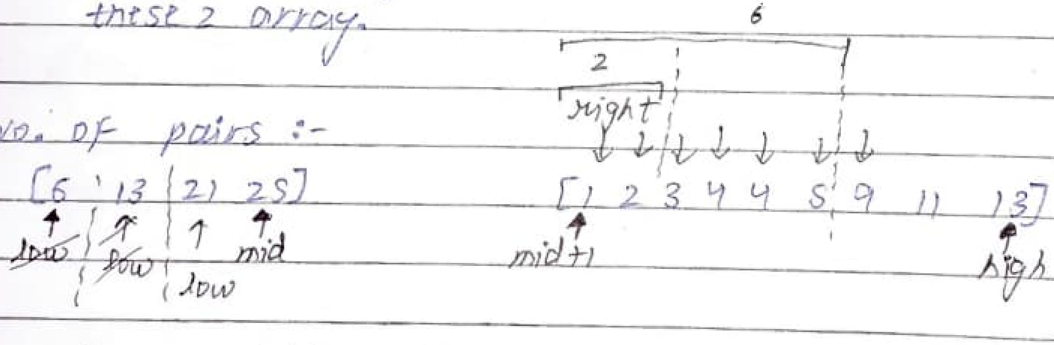
↳ count = $1+2+1+1+\cancel{1}+\boxed{3+3+3} = 15$

↳ [From lowest level, we try to apply previous logic of 2 array to calculate count. This is done from lowest to above levels.

↳ After counting, we simply merge.

↳ for above levels, this step is applied when the given 2 array are sorted, then count then merge these 2 array.

→ no. of pairs :-



```
cnt=0,  right = mid+1
for( int i = low → mid) {
    while( right <= high && a[i] > 2* a[right])
        right++;
    cnt = cnt + (right - (mid+1));
```

↳ currently points to ele. which does not satisfy the condition (right)

NOTE:- This will distort the input array. (mention it)