DSA

Page No:
Date:

DAY 1 → Logic Building

{Arrays}

Brute force → Create New Array → Bubble sort.

① Moves Zeroes to End.
↳ Also maintain the Relative order of Non-zero element.

TC:-
nums = {0, 1, 4, 0, 5, 2}
Ans → {1, 4, 5, 2, 0, 0}

Optimal:-

```
j = 0;
for (i=0 ⟶ n){
    if (nums[i] != 0){
        nums[j] = nums[i];
        j++;
    }
}
```

we Are moving All Non-zero Ele to front.

```
for (i=j ⟶ n){
    nums[i] = 0;
}
```

filling 0's in Remaining Array.

Brute force → TC(n×n) Check All Ele [0→n) that it is there or Not. One by One

③ Find Missing Number.
↳ Given Array which contains 0 to n Elements. # means there will be one No. missing. We Need to find it & Return.

TC:- ① nums = {0, 1, 2, 4, 5, 6} → Ans → ③ missing

② nums = {0, 2, 3, 1, 4} → Ans → ⑤ missing

Optimal ①:-
(i) Calculate sum of 'n' Numbers using formula
SumN = N * (N+1) / 2
↳ 0 to 5 ⇒ 15
(ii) Calculate the sum of All Elements in Array (Sum.) → 0+2+3+1+4 ⇒ 10
(iii) missing Num = SumN − Sum. → 15 − 10 ⇒ 5
TC ②.

Brute force → use HashSet< >.

② Remove duplicates from Sorted Array.
↳ We Need to return Count of unique Element.

T.C:- nums = {0, 0, 3, 3, 5, 6}
Ans → ④.
↳ nums = {0, 3, 5, 6, _, _}

Optimal:-

```
j = 0;
for (i=1 ⟶ n){
    if (nums[i] != nums[j]){
        j++;
        nums[j] = nums[i];
    }
}
return j+1;
```

→ using 'j' for unique Element.
→ if (nums[i] != nums[j]) we find an new unique Element so j++ & store it.
→ j++ → Because At 'j' previous unique Ele is stored, we are using it to check next unique Ele.

Optimal ②:- → Using XOR.

```
xor1 = 0; xor2 = 0;
for (i=0 ⟶ n+1){
    xor1 ^= (i+1); xor of (1...N)
    xor2 ^= nums[i]; xor of Arr Elements.
}
return (xor1 ^ xor2);
```

↳ It will Give the missing Element.

④. Union Of two Sorted Arrays. → Brute Force → Use TreeSet<> → Stores Only Unique Ele. → It is in Sorted Order.

↳ Given 2 sorted Array. Need to Return Union Array Contains Unique Elements from nums1 & 2.

TC①:- nums1 = {1, 2, 3, 4, 5} nums2 = {1, 2, 7}.

Page No:
Date:

Ans → ans {1, 2, 3, 4, 5, 7}.

Optimal :- → Two Pointer

nums1 <

```
i = 0 ; j = 0.   → pointing to nums 2
                 → Using to avoid Insertion of duplicate Elements in List.
prevEle = Integer.MIN_VALUE;
while (i < m && j < n){          → Case① & Case②.
    if(nums1[i] <= nums2[j]){
        if(nums1[i] != prevEle){
            list.add(nums1[i]);
            prevEle = nums1[i];
        }
        i++;
    }                            → Case③
    else{
        // Do same thing for j pointer
        j++;
    }
}
// Still 'i' might be less than m or 'j' might be less n. So Add 'i' & 'j' part After this.
// Convert List to int[] & return.
```

Also do same for both 'i' & 'j' After completition of this while loop.

↳ It is storing just previously inserted Element so update it.

---

⑤ Intersection of two sorted Arrays.

↳ Need to return All elements that Appears in both the Array.

↳ In Ans. Ele's must appear As many times as it Appears in both Arrays

TC①. A[] = {1, 2, 2, 3, 3, 4, 5, 6} → m

B[] = {2, 3, 3, 5, 6, 6, 7} → n.

Ans → {2, 3, 3, 5, 6}.

Brute force

```
int[] visited → 'n' size
for(i=0 → m){
    for(j=0 → n){
        if(A[i] == B[j] && !visited[j]){
            ans.add(B[j]);
            visited[j] //Mark visited
            break;
        }
        if(B[j] > A[i]){
            break;
        }
    }
}
```

Optimal :- → Two Pointer

```
i = 0 , j = 0.
while(i < m && j < n){
    if(A[i] < B[j])
        i++
    else if(A[i] > B[j])
        j++
    else{ // A[i] == B[j]
        ans.add(A[i]);
        i++ ; j++;
    }
}
```

Arrays → Medium.

Page No: .....

Date: .....

⑥ Leaders in An Array.

↳ Leader :- Ele strictly Greater that All elements to its right.

T.C :- arr = {1, 2, 5, 3, 1, 2} → Last Element is always Leader

Ans → {5, 3, 2}.

Brute Force → Check every ele that it is greater or Not.
- ↳ n * n
- Also use boolean flag.

Optimal :- → Use the logic of Next Greatest ele.
- ↳ Iterate from n → 0.
- ↳ Create maxEle variable & It will be updated in every iteration.

Compare Every Ele by maxEle & update the Result

⑦ Rearrange Array Elements by Sign.

↳ Given An Array → Contains equal +ve & -ve Numbers.
↳ Need Rearrange in A way that Every Consecutive pair will have opposite sign.

T.C :- arr = {2, 4, 5, -1, -3, -4}

Ans → {2, -1, 4, -3, 5, -4} → All pairs has Opposite sign

Brute Force :-
- ↳ Create → +ve Arraylist (Stores +ve Ele)
- ↳ -ve Arraylist (Stores -ve Ele)

=> Then update them in Original Array. Even index → +ve ele, Odd index → -ve ele.

Optimal :-
- ↳ Create an Ans array of n len
- ↳ use two variable pos = 0, neg = 1.

=> Based on this variables Update Ans array & increase variable by +2.

⑧ Print the Matrix In Spiral Manner.

	Left 0	1	2	3	4	Right 5 → n
top → 0	1	2	3	4	5	6
1	20	21	22	23	24	7
2	19	32	33	34	25	8
3	18	31	36	35	26	9
4	17	30	29	28	27	10
bottom → 5	16	15	14	13	12	11
↓ m						

top = 0, bottom = m-1
left = 0, Right = n-1.

```
while(top <= bottom && left <= Right){
    for(Left → Right){
    }
    top++; // top most Row Not Needed.
    for(top → bottom){
    }
    Right--; // Right most Column Done So.
    for(Right → left){
    }
    bottom--; // Bottom Row Done.
}
```

Do this if top <= bottom.

Only if (left <= Right)

```
for(bottom → top){
}
left++ // Left most Column Done.
```

DAY – 2

(9)

Pascal's Triangle.

Every row will have row NO. of Elements.

Variations:
- I → Return a Row, Col Element.
- II → Return an single Row.
- III → Return Entire Pascal Triangle

Page No:
Date:

⇒ # Variation I

↳ Need to return value present at rowth & colth position

T.C.

r = 4, C = 2.

1
1 1
1 2 1
1 3 3 1

Ans → 3

Methods

① Generate entire Triangle And return the element based on row, col.

Optimal :- → Using Formula → $^{n}C_{r} \Rightarrow {}^{R-1}C_{C-1}$

$$^{n}C_{r} = \frac{n!}{r! * (n-r)!}$$

Observation → $\frac{7!}{2! * (5)!} \Rightarrow \frac{7 \times 6 \times (5 \times 4 \times 3 \times 2 \times 1)}{(2 \times 1) \times (5 \times 4 \times 3 \times 2 \times 1)} = \frac{7 \times 6}{2 \times 1}$

From 7 it goes for 2 places means r places.

$$^{10}C_{3} = \frac{10 \times 9 \times 8}{3 \times 2 \times 1} \Rightarrow \frac{10}{1} \times \frac{9}{2} \times \frac{8}{3}$$

consider like this.

```
void int ncr(10 n, 3 r){
    res = 1;
    for(i=0 → r){
        res *= (n-i);
        res /= (i+1);
    }
    return res;
}
```

(10-0) (10-1) (10-2)
10/1 9/2 8/3
(0+1) (1+1) (2+1)

Brute Methods
① Generate Entire Triangle and Return the row.
② Use ncr method col times. Here r will be constant & col will goes from 0 → r.

Variation II

↳ Return an entire row.

T.C.

r = 4 → Ans → 1 3 3 1

Optimal :- → Optimized ncr method

```
ans = 1;
print(ans);
for(i=1 → r){
    ans *= (r-i);
    ans /= i;
    print(ans);
}
```

Observation

r = 6 →

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

ans ⇒ 1

5/1 → 5×4/1×2 → 5×4×3/1×2×3 → 5×4×3×2/1×2×3×4

we Already calculated this part for previous col.

this part is extra

ans = row − col / col.

(6-1) (6-2) (6-3) (6-4) (6-5)
5/(1) × 4/(2) × 3/(3) × 2/(4) × 1/(5)

Variation III → Return the 1st n rows of Pascal's Triangle.

Page No: .....
Date: .....

Brute Force

① Use Variation II to generate every row and add to Ans.

Optimal

n = 5

	0	1	2	3	4
0 →	1				
1 →	1	1			
2 →	1	2	1		
3 →	1	3	3	1	
4 →	1	4	6	4	1

arr[i][j] = arr[i-1][j] + arr[i-1][j-1]

```
List<List<Integer>> ans ;
for ( i = 0 ⟶ n ) {
    List<Integer> l ;                    // Every row has i Element.
    for ( j = 0 ⟶ i ) {
        if (j == 0 || j == i)            // 1st & last Col has 1
            l.add ( 1 );
        else {
            l.add ( get(i-1)(j) + get(i-1)(j-1) )
        }
    }
    ans.add ( l );
}
```

---

⑩ Rotate Matrix by 90 degrees.

T.C :-

arr

1	2	3
4	5	6
7	8	9

Brute Force →

7	4	1
8	5	2
9	6	3

Transpose →

1	4	7
2	5	8
3	6	9

Reverse Each Row

Brute force → Create new matrix of m × n.
→ Put every element to its correct position.
→ rotated[j][n-i-1] = arr[i][j];

Optimal.
→ Transpose in same given matrix.
→ Then reverse each row.

Transpose. → Only if cols & rows of arr are same.

```
for ( i = 0 ⟶ n ) {
    for ( j = 0 ⟶ i ) {
        swap (i, j, arr);
    }
}
```

---

⑪ Two Sum → Given → arr, Target.
→ Need to return indexes.

T.C : arr = { 1, 6, 2, 10, 3 } t = 7.
(0 1 2 3 4)

Ans → { 0, 1 }

1 + 6 ⇒ 7

If Only (True or False) ~~or Arr~~. Need to ~~Elements~~ Elements. [Not Indexes]
→ ~~Sort & use two pointer~~.
→ Then Sort the Array
→ And use Two Pointer Binary Search type logic

sum = arr[i] + arr[j]
(sum < t) ⟶ i++
(sum > t) ⟶ j--

Brute Fore → Check for Every pair if they are equal to target or Not.

Best. Better. → Use HashMap<int, int> ele, idx.
→ Every time you will check that Map contains remaining value or Not.
rem = ~~arr~~ t - arr[i]
→ if contains you got the Answer
→ if Not put current ele & its index to map.

(12) 3 - Sum → Given → Array.
→ Need to Return All Unique Triplets. (i != j, i != k, j != k)
→ arr[i] + arr[j] + arr[k] == 0. → Then Return this Triplet.

Page No: .....
Date: .....

I.C :- arr = { 2, -2, 0, 3, -3, 5 }
(Ans → { {-2, 0, 2}, {-3, -2, 5}, {-3, 0, 3} }

Better
→ arr[i] + arr[j] + arr[k] = 0.
arr[k] = -(arr[i] + arr[j]).
→ So you can run loop of n^2. And Check that Set contains arr[k] or Not.

```
Set< List<Integer> > AnsSet ;
for( i=0 → n ){
    set<Integer> set ;
    for( j = i+1 → n ){
        int k = arr[i] + arr[j];
        if (set contains (k)){
            // Do imp work.
        }
        else
            set.add (arr[j]);
    }
}
```

Brute Force
→ Run a Nested 3 loops of n^3.
→ Check All possible Triplet
→ if (Sum of Triplet == 0)
Imp → Then put in a list
→ Sort the List
→ Store the List in HashSet
[Imp for Unique Triplets.]

Best Optimal → You Not Needed index. so Two Pointer can be used.
{ 2, -2, 0, 3, -3, 5 }
Sort → { -3, -2, 0, 2, 3, 5 }
i, j, k
Use Two Pointer, while (j < k).
sum = i + j + k.
if (sum == 0){ // Store the Ans. j++, k--. }
else if (sum > 0){ k--; }
else j++.
Check :- if (arr[i] == arr[i-1]) Continue; only unique Needed.
{ -3, -3, ----, }
i →

(13) 4 Sum → Given → Array, Target.
→ Need to return All the Combination of 4 elements that Sums to target.
→ All Only Unique & [i != j != k != l]

T.C arr = { 1, 0, -1, 0, -2, 2 }. t = 0.
(Ans → { {-2, -1, 1, 2}, {-2, 0, 0, 2}, {-1, 0, 0, 1} }

Better
→ arr[i] + arr[j] + arr[k] + arr[l] = t.
Sum + arr[l] = t
arr[l] = t - sum
→ Rest is same as 3 Sum
→ Just you Need to Run n^3 Nested Loop.

Brut Force → Nested Loop n^4

Optimal :- → Same
→ Sort the Arr.
{ 1, 0, -1, 0, -2, 2 } → 1st sort this
i, j, k, l
two Pointer
diff:- (Sum == t)
k++ → till (arr[k] == arr[k-1])
l-- → till (arr[l] == arr[l+1]).

All the Logic are Same As 3 - Sum

(14) Sort An Array of 0's, 1's And 2's. → Given → Array → only contain elements 0, 1, 2.
↳ Need to sort the Array

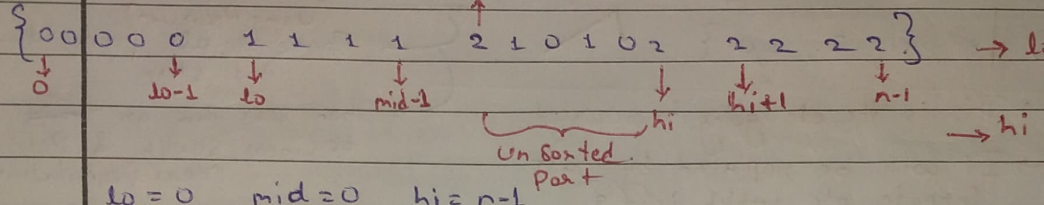
Page No: .....
Date: .....

I.C :- arr = { 1, 0, 2, 1, 0 }.
(Ans) { 0, 0, 1, 1, 2 }.

Brute Force → use Arrays.sort( ).

Optimal → Dutch Flag Algo.

Visualization



Bellos → Count 0's, 1's, 2's.
↳ And fill Accordingly.

→ lo se pehle sare zero hone chaiye.
→ hi k baad sare Two hone chaiye.
→ mid se lo tak sare 1 hone chaiye.

lo = 0, mid = 0, hi = n-1.

```
while( mid <= hi ){
    if( arr[mid] == 0 ){
        swap( lo, mid, arr );
        lo++, mid++
    }
    else if( arr[mid] == 1 ){
        mid++.
    }
    else {
        swap( hi, mid, arr );
        hi--;
    }
}
```

(15) Kadane's Algorithm → Given Array
↳ Need to find Subarray with Largest Sum.
↳ And Need to return Sum. [Largest].

I.C arr = { 2, 3, 5, -2, 7, -4 }.
Ans → 15

Brute force → Try all subarrays.
↳ Ans Store. Max mein.
↳ n^2.

Optimal :-

Sum = 0., maxSum = Int.Min.

```
for( i = 0 → n ){
    sum += arr[i];
    if( sum > maxSum ){
        maxSum = sum;
    }
    if( sum < 0 ){ -ve.
        sum = 0
    }
}
```

Follow - up. Question.
↳ Print the Subarray having max Sum?
↳ Kadane's Algo Observation
↳ whenever sum becomes zero. We starts forming new sub array.
↳ whenever sum becomes greater than maxSum. that index might be end.

⇒ Update the code.

start = 0; AnsSt = -1, AnsEnd = -1;

```
for (i = 0 → n) {
    if (sum == 0) start = i;
    sum += arr[i];
    if (sum > maxSum) {
        maxSum = sum.
        AnsSt = start;
        AnsEnd = i;
    }
}
if (sum < 0) sum = 0;
```

(16) Next Permutation → [Given Array]

↳ We need to modify the Array to its next Permutation.

No. of Permutation ⇒ n!

Page No:

Date:

IC ① arr = {3, 1, 2}

Ans {3, 2, 1}

② arr = {3, 2, 1}

Ans → {1, 2, 3} → Go to Start.

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2 → next
3 2 1

Optimal

→ find the break point.

arr = {2, 1, 5, 4, 3, 0, 0}

↳ This is the ele going to replaced.

↳ It will be replaced by just Greate ele which is (3).

{2, 3, 5, 4, 1, 0, 0}

It is Already in Increasing order.

So, reverse This you will get Next Permutation.

Brute Force

(Using Recursion)

↳ Generate all permutation And store

↳ Then sort then or Do Linear search & Return Next Permutation.

```
ind = -1;
// To find Break Point
for (i = n-2 ; i>=0 ; i--){
    if (arr[i] < arr[i+1]){
        ind = i;
        break;
    }
}
```

⇒ If there is No Break Point it Means This is Last permutation so, fall back to 1st. To DO so Reverse whole Array.

```
if (ind == -1) // No Break Point.
    for (i = n-1 ; i > ind ; i--){
        if (arr[i] > arr[ind]){
            swap (i, ind, arr);
            break;
        }
    }
reverse (ind+1, n-1, arr);
```

(17) Majority Element - I

↳ Given → Array

↳ Need to Return Majority Element which is Appeared more than n/2 times.

IC :- arr = {0, 7, 0, 1, 7, 7, 2, 7, 7}

Ans → (7)

Brute Force

↳ Check for every Element

↳ Count No. of Occurence for every Element And return

Optimal → Boyer-Moore Voting Algorithm.

```
ele ; count = 0;
for (int i = 0 ⟶ n){
    if (count == 0){
        ele = nums[i];
        count = 1;
    }
    else if (nums[i] == ele){
        count ++;
    }
    else if (nums[i] != ele){
        count --;
    }
}
return ele;
```

Better

↳ use Hash Map<ele, occurance>

↳ Iterate on map { return ele.

⇒ Algo will give the Ele which Occurs the Most Number Times in Array.

↳ So Check Is it occur more than n/2 times.

(18) Majority Element - II → Need to Return Ele's which Appears more than n/3 times

T.C. ① arr = {1, 2, 1, 1, 3, 2, 2} → Ans → {1, 2}

② arr = {1, 2, 1, 1, 3, 2} → Ans → {1}

Optimal :- → Modified Boyer-Moore Voting Algo.

```
ele1 = Int.MIN ; ele2 = Int.MIN ;
count1 = 0 ; count2 = 0 ;
for(i = 0 ⟶ n){
    if (count1 == 0 && ele2 != arr[i]){
        ele1 = arr[i];
        count1 = 1 ;
    }
    else if (count2 == 0 && ele1 != arr[i]){
        ele2 = arr[i];
        count2 = 1
    }
    else if (arr[i] == ele1){
        count1++;
    }
    else if (arr[i] == ele2){
        count2++;
    }
    else {
        count1--;
        count2--;
    }
}
```

Brute Force & Better → Same As ME - I.

(19). Find The Repeating And Missing Number

→ Given → Array → Containing only Ele from 1 to n.
→ So. one Ele is repeated & one is missing. Need to Return.

T.C. arr = {3, 5, 4, 1, 1} → Ans → {1, 2}

#Optimal. - (I) → Math

Arr = {4, 3, 6, 2, 1, 1} n = 6

(4+3+6+2+1+1) − (1+2+3+4+5+6)

S − Sn.

$S_n = \frac{n \times (n+1)}{2}$ $S_{2n} = \frac{n \times (n+1) \times (2n+1)}{6}$

S − Sn = (1) − (5) = −4 → x − y = −4 → (I)

x → Repeating
y → Missing

$(4^2+3^2+6^2+2^2+1^2+1^2) - (1^2+2^2+3^2+4^2+5^2+6^2)$

$S_2 - S_{2n} = (1)^2 - (5)^2 = -24$

$x^2 - y^2 = -24$.

$(x-y)(x+y) = -24$.

$(x+y) = \frac{-24}{x-y} = \frac{-24}{-4} = 6$ → (II)

x − y = −4 → (I)
+ x + y = 6 → (II)
2x = 2.
x = 1 → Repeating No.

x + y = 6.
y = 6 − x = 6 − 1 = 5 → Missing

Brute Force
→ Run A loop from 1 to n.
→ Count Occurance of every (i) in Array.
→ If (count == 0) → Missing No.
→ If (count == 2) → Repeated No.

Better → use HashMap.
→ Put ele & its frequency in map [Array Elements].
→ Run a loop from 1 → n.
→ if (freq == 2) → Repeated.
if (!map.contains) → Missing.

S, Sn, S2n, S2;

Val1 = S - Sn; → // x - y → (I)

Val2 = S2 - S2n;

Val2 = Val2 / Val1; → // x+y → (II)

longx = (Val1 + Val2) / 2; // (I + II)/2

longy = x - Val1;

Page No:
Date:

Sorted Array has Inversion of 0.

Sorted in Descending order has maximum Inversion.

(20) Count Inversions → Given → Array. → Need to Return No. of Inversions.

Inversion
- a[i] > a[j]
- i < j

Brute Force n^2
- Run nested loops
- Check all the pairs & count.

T.C :- Arr = {2, 3, 7, 1, 3, 5}
Ans (5)
2 > 1
3 > 1
7 > 5
7 > 1
7 > 3

Optimal :- → Merge Sort → Modify merge two Sorted Array

{1, 2, 3, 4, 5}
{5, 3, 2, 4, 1}

{5, 3, 2} (3) + (1) {4, 1}
{2, 3, 5}
{5, 3, 2}
{5, 3} (2) {2}
{3, 5}
{5, 3}
{5} (1) {3}
{4} (1) {1}

Inv. = (1) + (2) + (1) + (3) + (1)
= (8) //

# Logic :-
a => {2, 3, 5, 6} b => {2, 2, 4, 4, 8}
a[i] > b[j] → inv += (a.length - i)
Inv. :- +3 +3 +2 +2
Inv. = 10 //

(21) Reverse Pairs → Given → Array → Need to Return No. of #Reverse Pairs in Arr.

Reverse Pairs.
- i < j
- Arr[i] > 2 * Arr[j]

Brute Force
```
for(i=0 → n){
    for(j=i+1 → n){
        if(arr[i] > 2*arr[j])
            count++;
    }
}
```

T.C
Arr = {6, 4, 1, 2, 7}
Ans (3)
- 6 > 2*1
- 6 > 2*2
- 4 > 2*1

Optimal: → Modify Merge Sort → Before Merge two Arrays call Inversion.

{6, 13, 21, 25} {1, 2, 3, 4, 4, 5, 9, 11, 13}

Count = 2 + 6 + 7 + 8

{Both code doing same thing, just logic is diff}

```
for(int i=0 → i<a.length){
    while(j a[i] > 2*b[j]){
        j++
    }
    count += j;
}
```

```
int inversion(int[] a, int[] b){
    i=0, j=0, count=0;
    while(i<a.len && j<b.leng){
        if(a[i] > 2*b[j]){
            count += a.len - i;
            j++;
        }
        else
            i++;
    }
    return count;
}
```

(22) Maximum Product SubArray In An Array. Page No: Date:

↳ Given → Array → Need to Return the Max^m Product of SubArray

T.C ① Arr {4, 5, 3, 7, 1, 2} Ans {84 0}

② Arr = {-5, 0, -2} Ans → (0)

Optimal → use Prefix & Suffix.

Brute Force → Generate All the Subarrays & Return Max^m product.

```
prefix = 1 , suffix = 1
for (i=0 → n) {
    if(prefix==0)  prefix = 1;
    if(suffix==0)  suffix = 1;
    prefix *= nums[i];
    suffix *= nums[n-i-1];
    Ans = Math.max(Ans , max(pref , Suf));
}.
```

---

(23) Merge two Sorted Array Without Extra Space.

↳ Given → Arr1 → m
↳ Arr2 → n.

T.C ⇒ Arr1 = {-5, -2, 4, 5, 0, 0, 0} m = 4
Arr2 = {-3, 1, 8} n = 3.

Ans → Arr1 = {-5, -3, -2, 1, 4, 5, 8}

Brute Force → Use Extra Space.

```
Ans[] = [m+n]
i=0 , j=0 , k=0
while (i<m && j<n){
    if (Arr[i] <= Arr[j]){
        Ans[k++] = Arr[i++];
    }
    else {
        Ans[k++] = Arr[j++];
    }
}.
```

⇒ Fill Back to Array One.

Optimized

```
i = m-1 , j=0;
while(i>=0 && j<n){
    if(Arr1[i] > Arr2[j])
        swap(i, j, Arr1, 2);
        i--;
        j++;
    }.
    else break;
}
```

⇒ Sort Both the Array
⇒ Fill Arr2 Elements to Back of Arr1

Arr1 = {1, 3, 5, 7} Arr2 = {0, 2, 6, 8, 9}
swap(2)
Sort → {0, 1, 3, 3}
Sort → {5, 6, 7, 8, 9}

(24) Search A 2D Matrix II → Need to check that target is present in Matrix. Page No: ..... Date: .....

Given → Matrix, Target

Matrix → Each row is sorted in Ascending Order Left to Right.
→ Each Column is sorted in Ascending Order top to bottom

t = 20

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

Left, Down

Optimal.

Brute Force → Traverse in Entire Matrix.

Logic

```
if (arr[i][j] > target) {
    j--; // All Down Ele's are Greater, So Go Left.
}
else if (arr[i][j] < target) {
    i++; // All Left Ele's are Smaller, So go Down.
}
else return true;
```

(25) Score After Flipping Matrix. → Given → Binary Matrix [containing 0's & 1's]
→ Return Score.

Move → You can take any Row or Col. & [toggle]. its all Elements considered As one Move

0 → 1, 1 → 0

⇒ Every Row of a matrix is equivalent to Binary Numbers.
⇒ You Need to find maxᵐ possible score. → Sum of all Binary Numbers
⇒ You can use Any No. of Moves. At the end score should be maxᵐ.

Hint:
100000 > 001111
1000 > 0111

Optimal.
→ Consider Hint: Put (1) At the 0th position of every Row.
→ Now travers in Every Col & Count 0's & 1's if 0's are more the toggle that Col.
→ You will get the Maxᵐ Score.

0	0	1	1	0th Col to 1
1	0	1	0	
1	1	0	0	

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

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

Consider All the 1's As ⇒ 8 4 2 1

(8×3)+(4×2)+(2×2)+(1×3)
⇒ 39

```
# Calculate A score..
score = 0, factor = 1;
for (j = n-1 → 0) {
    for (i = 0 → m) {
        score += arr[i][j] * factor;
    }
    factor *= 2;
}
```

(26) Set Matrix Zeroes.

→ Given → Matrix.

→ if element is 0 the set entire row And col to 0's.

Page No:

Date:

T.C.

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

→

0	0	0	0
0	4	5	0
0	3	1	0

Brute force.

→ Create Copy of matrix.

→ Check for 0's in helper matrix & make Changes in Original matrix.

Better. → use two array of m & n size. Boolean Array.

→ if 0 found mark that row True & Col true.

0	T
1	F
2	F

0	1	2	3
T	F	F	T

1	2	2	0	21
3	0	24	21	9
6	10	7	0	-8
0	4	19	3	99

zeroRow → True

zeroCol → True.

Optimal. → O(1) space

→ Create two boolean variable

→ zeroRow = false

→ zeroCol = false

⇒ 1st Iterate on 0th row & 0th Col. if Any 0 found make that variable True.

⇒ Now you can use 0th row & 0th Col for remaining matrix.

1	0	2	0	21
0	0	24	21	9
0	10	7	0	-8
0	4	19	3	99

→ Use this row like Better App. Remaining Part

→ Use this Col. like Better Approach.

⇒ Now iterate on Row if 0 found make that entire row 0's.

⇒ Now iterate on Col if 0 found make that entire Col 0's.

⇒ if zeroRow is True make 0th row to zeroes

⇒ if zeroCol is True make entire 0th Col to zeroes.