# Day 01

## # Step 3 : Solve problems on Array :-

## # Sort an array of 0's 1's & 2's

arr =

| 0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|

Brute force :- merge sort (nlogn) — TC — SC: n

Better :- We will take counters

```
ctr0=0, ctr1=0, ctr2 = 0

for (int i=0 ; i< arr.size() ; i++)
{
    if (arr[i] ==0)
        ctr0 ++;

    else if (arr[i] ==1)
        ctr1 ++;

    else
        ctr2 ++
}
```

O(n)

```
for (int i=0 ; i<ctr0 ; i++)  arr[i] = 0;
for (int i=ctr0; i<ctr0+ctr1 ; i++)  arr[i]=1;
for (int i=ctr1 ; i<ctr0+ctr1+ctr2 ; i++) arr[i]=2;
```

O(n)

## # Soln :- Dutch National Flag Algorithm.

Best

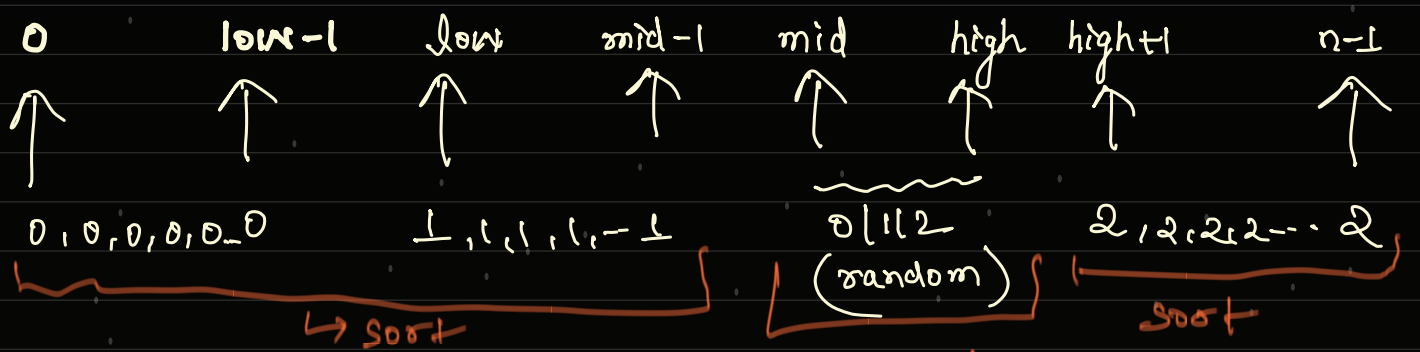
It says that :-

R0) [0 ..... low - 1] → 0

R1) [low ..... mid - 1] → 1

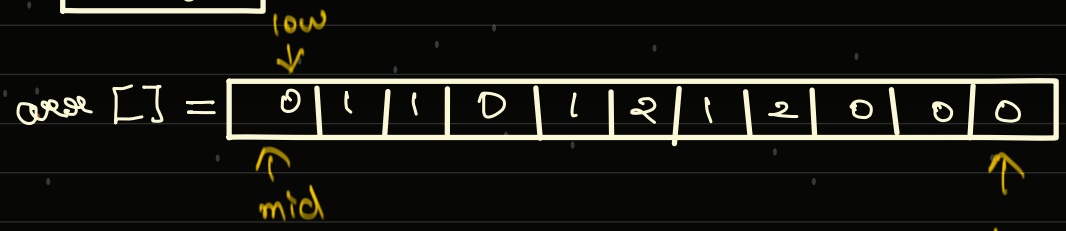
R2) [high+1 ..... n-1] → 2

## Imp Diagram to remember :-




We will be three pointer

low
mid
high




There is three possibility :-

a[mid] == 0 swap(arr[low], arr[mid])
↳ 0 come in correct order
low ++
mid ++

a[mid] == 1 mid ++;

a[mid] == 2 swap(arr[mid], arr[high])
high --;

So, Here → time complexity = O(n)
→ Space complexity = O(1) we don't use any extra array.

# Majority element ( > N/2 times)

ex). arr[] = { 2 2 3 3 1 2 2 }

N = 7

We need to find element appears more than 3 times.

Ans) 4

Brute force :-

$$\text{times} = \frac{N}{2} = \frac{7}{2} = 3$$

```
int cnt = 1;
for (int i = 0; i < N; i++)
{
    for (int j = i+1; j < N; j++)
    {
        if (arr[i] == arr[j]) cnt++;
    }
    if (cnt > N/2) return arr[i];
}
return -1;
```

# Hashmap (Better $\Sigma o(n)$)

arr[] = { 2 2 3 3 1 2 2 }

We will unordered_map<int, int> mp;

```
for (int i=0; i < v.size(); i++)        // O(n)
    mp[v[i]]++;
for (auto ele : mp)                     // O(n)
{
    if(ele.sec > v.size()/2) {
        return ele.first();
    }
}
return -1;
}
```

| element (Key) | cnt (value) |
|---|---|
| 2 | 4 |
| 3 | 2 |
| 1 | 1 |

So, TC : O(1) or O(n) + O(n)
SC : O(n) → using map if contains unique element.

# Moore's Voting Algorithm :

arr[] = [7 7 5 7 5 1 5 7 5 5 7 7 5 5 5]

Algorithm takes two variable

hypothetical assumption 7 is my ans.

v1) ele = 7, 5

v2) cnt = 0 1 2 1 2 1 0, 1, 0 1 2 1, 0 1 2 3 4

initially

→ So, [7 , 7 , 5 , 7 , 5 , 1]

ele=7 → 3 times

→ others (5,1) → 3 times

→ [5 , 7] → 1 → no

→ [5 5 7 7] → 2 → NO

So, steps are :-

① Apply Moores Voting Algorithm

② Verify elem is majority or not

```
for (int i =0 ; i < N ; i++)
{
    cnt = 0
    if (arr[i] == elem) cnt++
    if (cnt > N/2) return ele;
}
```

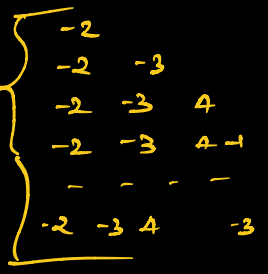
# Maximum Subarray Sum

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

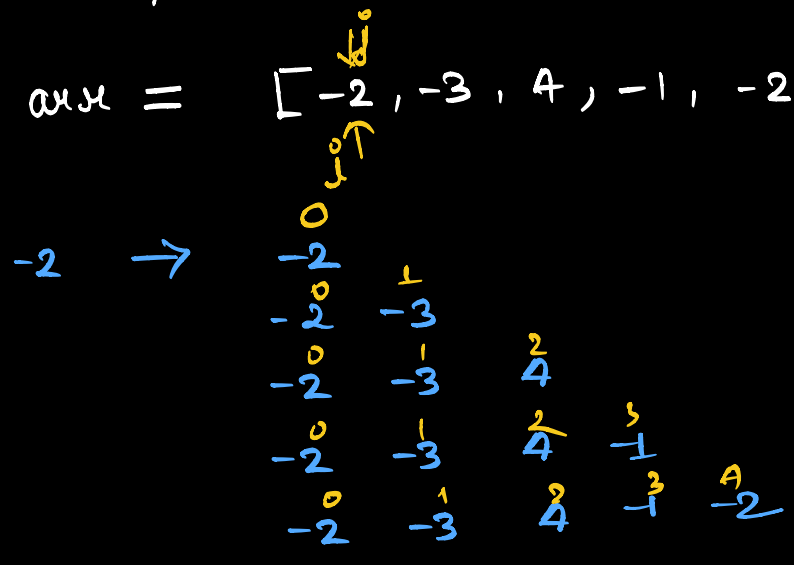
max = 0

```
for (int i=0 ; i<n ; i++)
{
    for(int j= i ; j<n ; j++)
    {
        sum = 0
        For (k=i ; k<j ; k++)
        {
            sum += arr[k]
        }
    }
}
```

(Traverse)




ex). arr = [-2, -3, 4, -1, -2]




-2, -2 -3

for → har element ko traverse ke liye

```
int max = -∞ or INT_MIN
for(int i=0;i<n;i++)
{
    for(int j=i ; j<n ; j++)
    {
        sum = 0
        for(int k=i ; k<=j ; k++)
            sum += arr[k]
        max = max(sum, max)
    }
}
```

Note :- Kth loop is necessary as we are tracking subarray through it, it iterates from i to j

Note :- TC :- O(n3) , S.C. = O(1)

# Better :-

```
for (int i=0 ; i<n ; i++)
{
    sum = 0
    for (j=i ; j<n ; j++)
    {
        sum += arr[j]
        max = max (max, sum)
    }
}
```

Better $O(n^2)$ ✗

Best :- Kadane's Algorithm

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

max = INT_MIN -2 4 7

sum = 0 -2 0 -3 0 4 3 4 2 7 4

(ans)

S1). Add -2 to sum

S2) -2 (sum) is compared with max, as -2 > max

S3) update max with -2

S4) If we move to -3, will it make sense to carry -2 (sum), if sum is -ve donot need to carry for negative sum.

S5) make sum = 0

S6). Now, we are −3, add to sum

S7) Compare sum with max (max > sum)

Do not replace

S8). As sum is −ve ⟶ make it zero

S9). Add sum = 4

S10) Compare it with max −2 (4 > −2)

replace max.

S11) Add sum = sum + next element (As sum +ve) (4)

⇓

carry

# Probable followup question :-

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int max_sum = INT_MIN;
        int sum = 0;
        int start = 0, end = 0;       // final indices of max subarray
        int temp_start = 0;           // temp start index for current subarray

        for (int i = 0; i < nums.size(); i++) {
            if (sum < 0) {
                sum = 0;
                temp_start = i;       // new subarray starts here
            }
            sum += nums[i];

            if (sum > max_sum) {
                max_sum = sum;
                start = temp_start;   // update final indices
                end = i;
            }
        }

        cout << "Max Subarray starts at index " << start
             << " and ends at index " << end << endl;

        return max_sum;
    }
};
```

jha se humara subarray shuru hota h wha sum = 0

So, take two var ans_start = -1, ans_end = -1

# # Best time to buy & sell stock :-

arr[] → [7, 1, 5, 3, 6, 4]  n=6

Buy → 1
sell → 6
} Profitize ✓

Common Sense :- Buying 1st Selling

Note :- If you are selling on $i^{th}$ day you should buy on minimum price from Day 1 to $(i-1)^{th}$ day.

arr[] → [7, 1, 5 | 3, 8, 4]

⇓ minimum cost aani chahiye cost

```
mini = arr[0] , profit = 0
for (int i=1 ; i < n ; i++)
{
    cost = a[i] - mini
    if (cost > 0){
        profit = max(profit , cost)
    }
    mini = min(mini, a[i])
}
```

# Rearrange the array in alternating positive and negative items

**Problem Statement :-** We are given array of size $N$ where we have $\frac{N}{2}$ +ve elements and $\frac{N}{2}$ negative elements.

arr[] = [3, 1, -2, -5, 2, -4]

↓ rearrange

arr[] = [3 -2 1 -5 2 -4]

+ - + - + -

Note :- +ve :- 3, 1, 2

-ve :- -2, -5, -4

order is maintained.

Brute Force :-

Iterate entire array

+ve vec → [3, 1, 2] → even

-ve vector → [-2, -5, -4] → odd

```
for(int i=0 ; i < N/2 ; i )
{
    arr[2i] = vec1[i]   (vec1 - pos)
    arr[2i+1] = vec2[i]
}
```