(1) Two-sum:

O(n) approach:

Using hash-map.

Focus on finding complement of each nums[i]; if complement is found in the map then return its index and i as answer. If not found then store the index of the element in the map.

```
unordered_map<int,int> mp;
for(int i=0; i<n; i++){
  int complement = target - nums[i];
if(mp.find(complement) !=
            mp.end()){
    return {mp[complement], i};
```

```
        mp[nums[i]] = i;
    }.
```

---

② Dutch national flag algo:

Time: O(N), space: O(1).

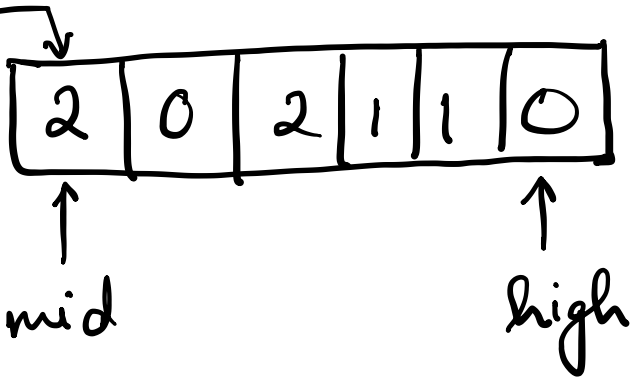
[0 ------ low-1] [low ----- mid-1]

[mid ----- high] [high+1 ---- n-1]

⇒ All the segments are sorted except the (mid - high) range.

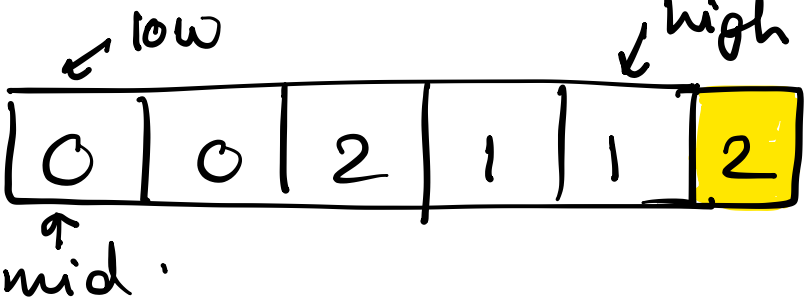
Question: Sort colors.

Ex:



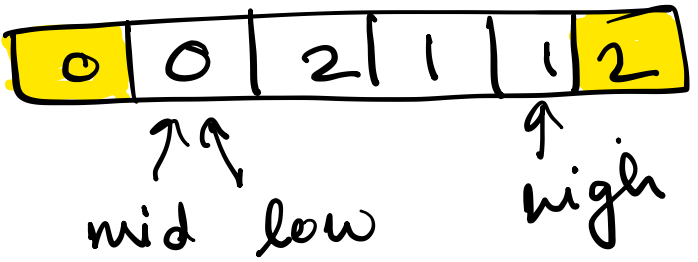
NOW arr[mid] = 2 ⇒ swap arr[mid] = 2 and arr[high] and high--

⇒



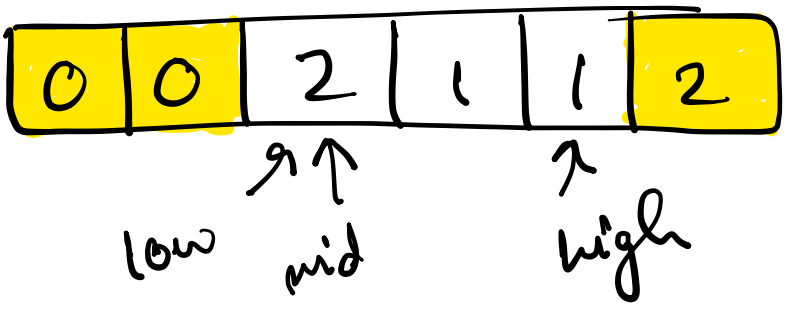
Now arr[mid] == 0, swap arr[mid] & arr[low], low++, mid++.

⇒

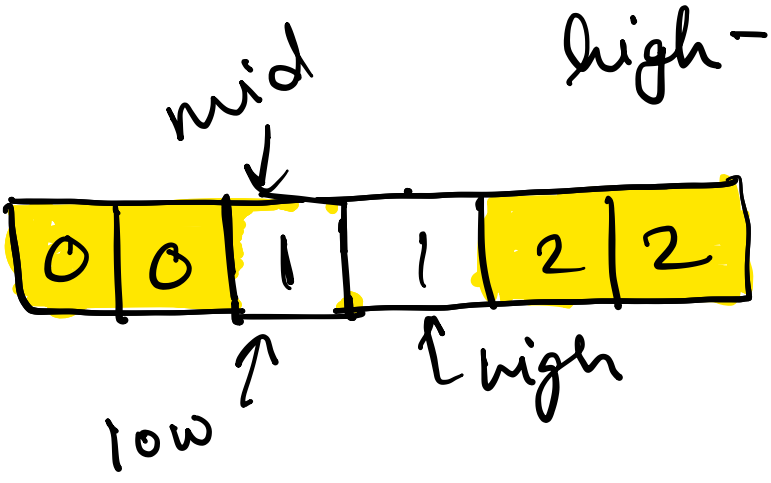


Now arr[mid] == 0, swap arr[low] & arr[mid], low++, mid++.

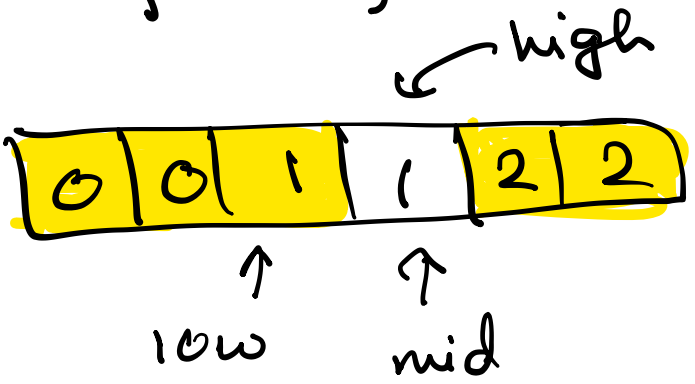
⇒



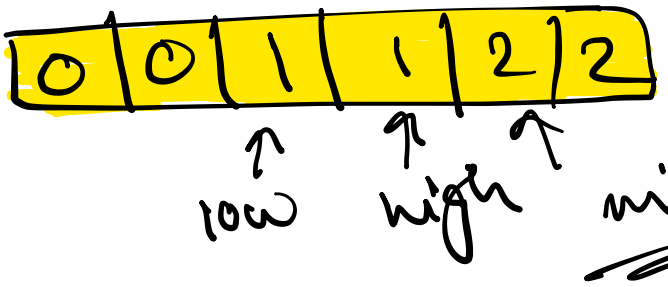
Now arr[mid] == 2, swap arr[mid] & arr[high]; high--.



⇒ arr[mid] == 1, mid++



⇒ arr[mid] == 1, mid++



Code:

```
int mid = 0;
int low = 0;
int high = n-1;
while (mid <= high) {
    if (arr[mid] == 0) {
        swap (arr[low], arr[mid])
        low++; mid++; }
    else if (arr[mid] == 1)
        mid++;
```

```
else { swap(arr[mid], arr[high]);
        high--;
    }
    return i;
}
```

(3) Moore's Voting Algorithm:

To find the majority 'element' present in the array.

It helps optimise the soln. to
O(N) time complexity. &
O(1) space complexity.

Example question: Majority Element.

logic: • If count == 0, update the candidate to the current number.

- If current number is equal to the candidate, increment count.

else decrement count.

Example: [2, 2, 1, 1, 1, 2, 2]

count = 0, candidate = 0

[2] → count = 1 ⇒ candidate = 2.

[2] → count = 2 ⇒ candidate = 2

[1] → count = 1 ⇒ "

[1] → count = 0 ⇒ " 2 → 1

[1] → count = 1 ⇒ candidate = 1

[2] → count = 0 ⇒ candidate = 1 → 2

[2] → " = 1 ⇒ candidate = 2

Thus..

(4) Kedane's Algorithm:

If the sum of a sub-array becomes less then 0, don't consider it.

Only consider those sub-arrays which gives +ve summation.

Ex: $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$.

Let sum = INT_MIN.

→ (-2) : sum = -2 (neglect)

→ (1) : sum = 1

→ (1, -3) : sum = -2 (neglect)

→ (4) : sum = 4

→ (4, -1) : sum = 3

→ (4, -1, 2) : sum = 5

→ (4, -1, 2, 1) : sum = 6 # desired.

→ (4, -1, 2, 1, -5) : sum = 1.

→ (4, -1, 2, 1, -5, 4) : sum = 5

Rough:

prices = [7, 1, 5, 3, 6, 4]

⇒ Buy at day 2 and sell at day 5.

Choose min then choose maximum.

(5) Buy and sell stock:

choose the min. element & then compare its difference with each of its next elements.

The pair with the maximum difference will be the ans.

else return 0.

(6) Rearrange Array Elements by Sign:

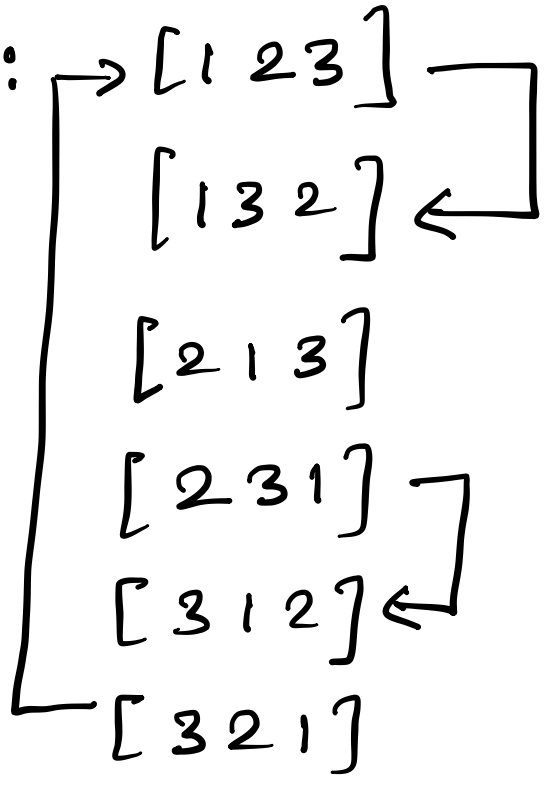
Example: nums = [3, 1, -2, -5, 2, -4]

Positive integers = 3, 1, 2.
Negative integers = -2, -5, -4.

Order: [3 -2 1 -5 2 -4]

(7) Next Permutation: (Pending)

Permutations:



(8) leaders in an Array.

Ex: [4, 7, 1, 0].

⇒ 7, 1, 0.

Ex: [10, 22, 12, 3, 0, 6].

⇒ 22, 12, 6

[10, 22, 12, 3, 0, 6]

i ↑ ↑ j

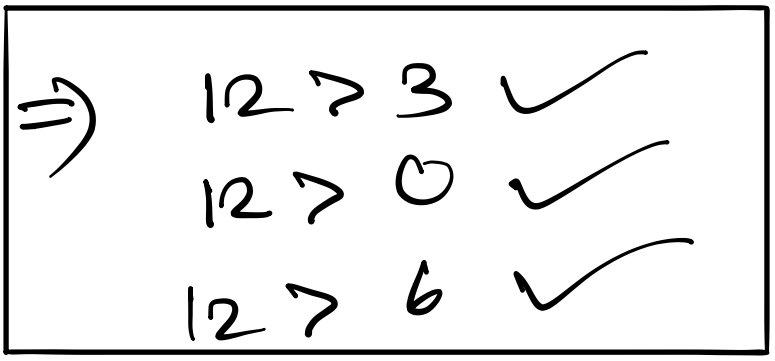
if (arr[i] > arr[j]) (j++

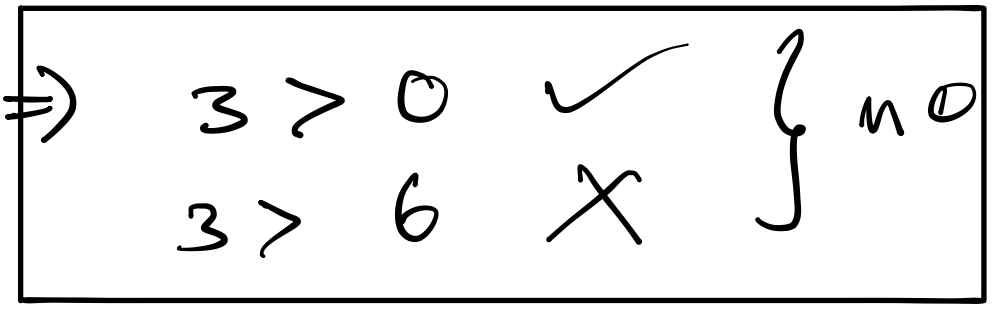
⇒ 10 > 22 ✗

⇒ 22 > 12 ✓ ;

22 > 3 ✓

22 > 0 ✓ 22 > 6 ✓.






```
vector<int> ans;
for(int i =0; i< nums.size(); i++){
bool leader = true;
for(int j = i+1; j< nums.size(); j++){
if(arr[j] > arr[i]){
      leader = false;
      break;
      }
if(leader){
      ans.push_back(arr[i]);
      }
return ans;
};
```

Here the complexity is O(n ^ 2)

Approach 2: To store the max element traversing from the right end towards the left end.

```
int n = nums.size();
int maxi = INT_MIN;
vector<int> ans;
int max = arr[n-1];
ans.push_back(arr[n-1]);
for(int i =n-2; i>=0; i--){
if(arr[i] > max){
      ans.push_back(arr[i]);
      max = arr[i];
}
return ans;
}
```

```
int max = arr[n-1];
ans.push_back(arr[n-1]);
for(int i=n-2; i>=0; i--) {
    if (arr[i] > max) {
        ans.push_back(arr[i]);
        max = arr[i];
    }
}
return ans;
}.
```

⑨ Find length of longest sequence which contains consecutive elements.

Example: [100, 200, 1, 3, 2, 4].

→ Ans = 4.

Sort : (1, 2, 3, 4, 100, 200)

```
for (int i=0 ; i<n; i++) {
  if (arr[i+1] - arr[i] == 1)
        cnt++;
  if (arr[i+1] == arr[i]) continue;
  else  cnt = 1;
  maxi = max (ans, cnt);
```

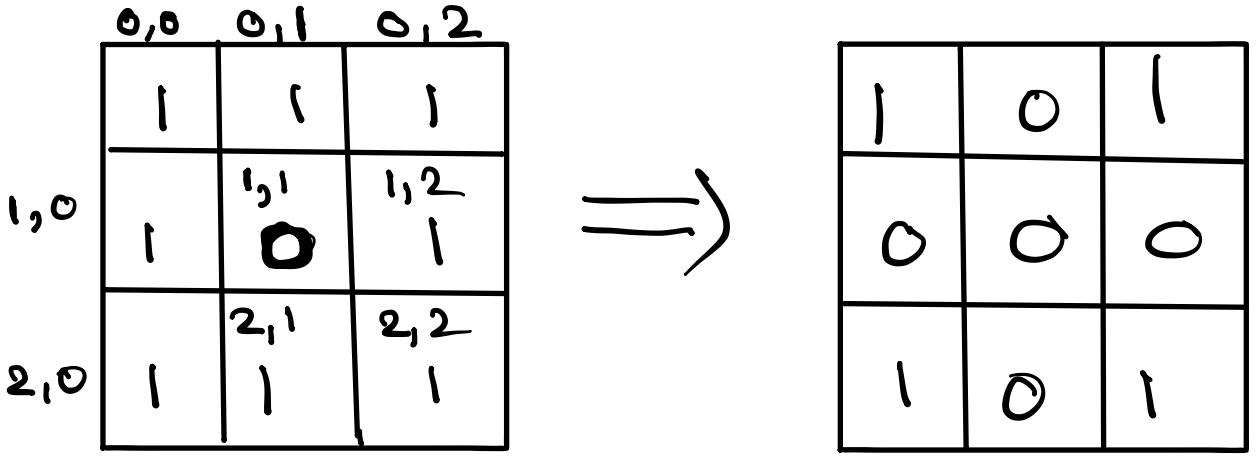
TC: O(n log n)

SC: O(1)

TC : O(n) ⇒ Approach : Approach 2:

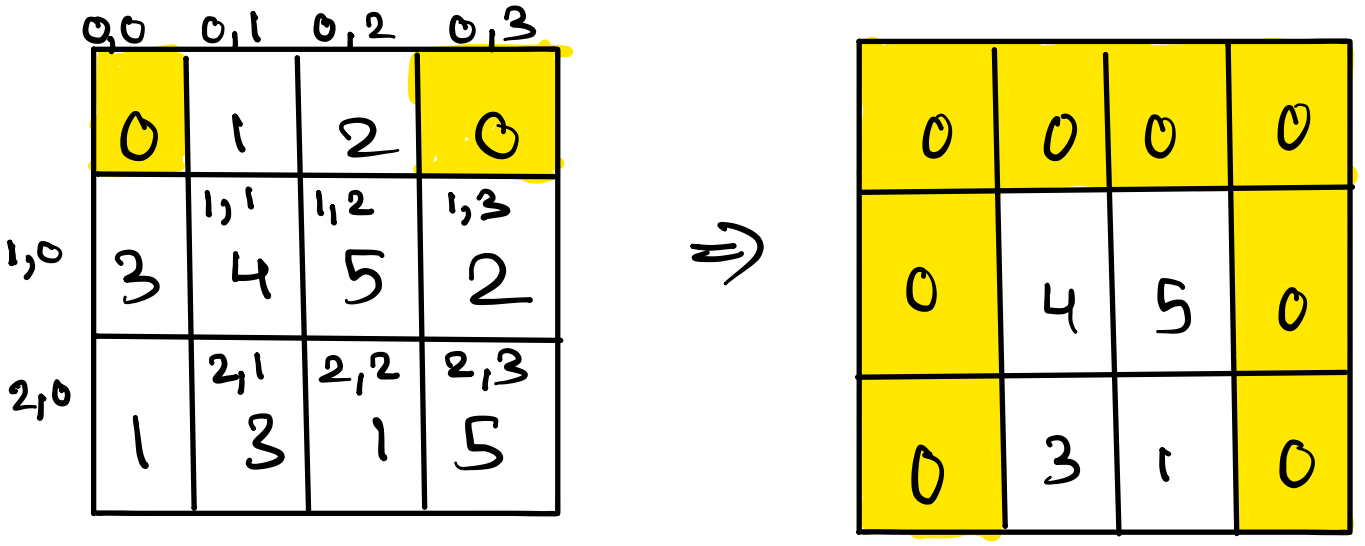
SC : O(n)

(10) Set matrix Zero :




```
if (matrix[row][col] == '0') {
    matrix[row + i][col] = 0;
    matrix[row][col + i] = 0;
}
```

↘ for(int i=0; i<n; i++) {.



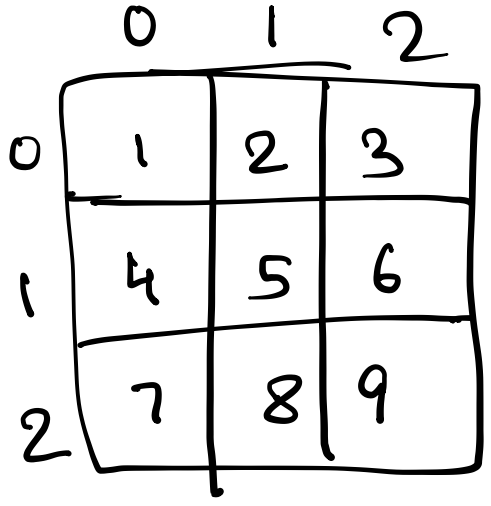
m x n ⇒ row x col.

n columns.

m rows.

```
vector<bool> row(n, false);
vector<bool) col(m, false);
for(int i=0; i<n; i++) {
    for(int j=0; j<m; j++) {
        if (matrix[i][j] == 0) {
            row[i] = true;
            col[j] = true;
        }
    }
}
```

(11) Rotate matrix by 90° degree :



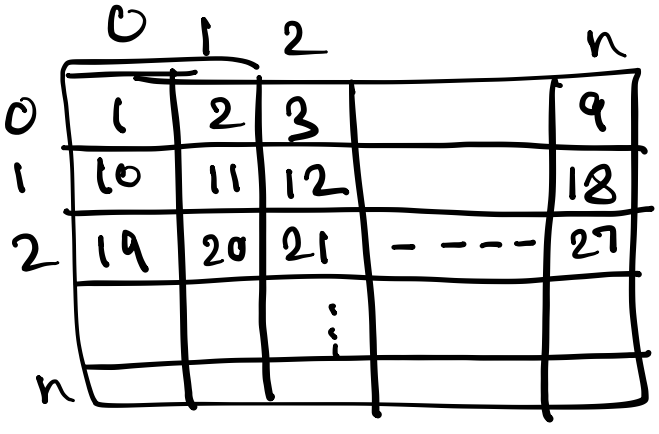
$\Rightarrow$

7	4	1
8	5	2
9	6	3

0^{th} row $\Leftrightarrow$ 2^{nd} col.

1^{st} row $\Leftrightarrow$ 1^{st} col.

2^{nd} row $\Leftrightarrow$ 0^{th} col.



0^{th} row $\Leftrightarrow$ n^{th} col.

1^{st} row $\Leftrightarrow$ $(n-1)^{th}$ col.

2^{nd} row $\Leftrightarrow$ $(n-2)^{th}$ col

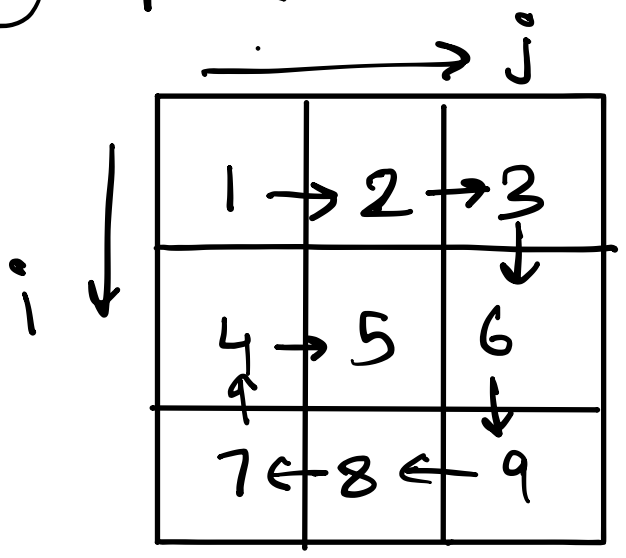
$\vdots$

$(n-1)^{th}$ row $\Leftrightarrow$ 1^{st} col.

n^{th} row $\Leftrightarrow$ 0^{th} col.

Transpose and reverse.

(12) Spiral Matrix :



=> Ans: [1 2 3 6 9 8 7 4 5]

if j reaches (n-1) →
iterate column [n-1]

if i reaches (n-1) →
iterate ~~REV~~ row [n-1].

if j reaches 0 →
iterate column [0]

```
int top = 0;
int bottom = matrix.size() - 1;
int left = 0;
```

```
int right = matrix[0].size()-1;
while (top <= bottom &&
       left <= right) {
    for(int i= left; i<=right; i++) {
        result.push_back(matrix[top][i]);
    }
    top++;
    for (int i= top; i<= bottom; i++){
        result.push_back(matrix[i][right]);
    }
    right--;
```

(3) Imp

Subarray-sum Equals K.

Example: nums = [1,1,1] ; k=2

subarray 1 = [1+1] = 2

subarray 2 = [1+1] = 2

nums = [1 2 3] ; k = 3.

subarray 1 = [1+2] = 3

subarray 2 = [3] = 3

$O(N^2)$ soln. → Easy (don't do)

Try PrefixSum approach. ($O(N)$)

```
int i=0;
while (i<n) {
    sum = arr[i];
if (sum == k)  cnt++;
  i=
```

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int n = nums.size();
        int cnt = 0;
        int sum = 0;
        unordered_map<int, int> mp;

        mp[0] = 1;  // To handle the case when subarray starts from index 0

        for (int i = 0; i < n; i++) {
            sum += nums[i];

            if (mp.find(sum - k) != mp.end()) {
                cnt += mp[sum - k];
            }

            mp[sum]++;
        }

        return cnt;
    }
};
```

(14) Majority element > $\lfloor n/3 \rfloor$.

Example: nums = [3, 2, 3]

Ans = 3

⇒ Simply do using frequency mapping.

TC : O(n), SC : O(n).

Optimal Approach: TC : O(n)
SC : O(1)

Idea: Atmost, 2 elements in an array can appear more than $\lfloor \frac{n}{3} \rfloor$ times.

```
vector<int> majorityElement(vector<int>& nums){
int n = nums.size();
int count1 = 0;
int count2 = 0;
int candidate1 = 0;
int candidate2 = 1;
for(int num: nums){
    if(num == candidate1){
        count1++;
    }
    else if(num == candidate2){
        count2++;
    }
    else if(count1 = 0){
        candidate1 = num;
        count1 = 1;
    }
    else if(count2 = 0){
        candidate2 = num;
        count2 = 1;
    }
    else{
    count1--;
    count2--;
    }
}

count1 = 0;
count2 = 0;
for(int num: nums){
    if(num == candidate1) count1++;
    else if(num == candidate2) count2++;
}

vector<int> ans;
if(count1 > n/3) ans.push_back(candidate1);
if(count2 > n/3) ans.push_back(candidate2);
return ans;

    }
};
```

(15) 3 Sum problem :

Find 3 such elements so that the sum is equal to 0.

nums = [-1, 0, 1, 2, -1, -4]

Ans = [-1, 0, 1] ; [-1, -1, 2] .

Idea: Fix i and move j & k pointers.

- Sort the array vector.
- to prevent duplicates ignore consecutive elements.

16

General Approach to solve K-sum problem:

General Strategy →

1. Sorting the array.
   → to skip duplicates.
   → Two pointers logic for 2 sum.
2. Using recursion to reduce K-sum into (K-1) sum.
3. Base-case is 2sum, solved with pointers.

Template: (Remember) :

Template for k-sum:

```
vector<vector<int>> kSum(vector<int>& nums, int target, int k , int start){
vector<vector<int>> ans;
int n = nums.size();
if( k == 2){
int left = start;
int right = n-1;
while(left < right){
      int sum = nums[left] + nums[right]);
      if(sum == target){
            ans.push_back({nums[left], nums[right]);
            left++;
            right--;
            while(left < right && nums[left] == nums[left-1]) left++;
            while(left < right && nums[right] == nums[right+1]) right--;
      }
      else if(sum < target) left++;
      else{
                  right--;
            }
      }
}
else{
      for(int i = start; i< n-k+1; i++){
            // tpo avoid the duplicate elements
            if(i > start && nums[i] == nums[i-1]) continue;
            vector<vector<int>> temp = kSum(nums,  target-nums[i], k-1, i+1);
            for(auto& t: temp){
                  for (auto& t : temp) {
                  t.insert(t.begin(), nums[i]);
                  res.push_back(t);
                  }
            }
      }
return res;
}

vector<vector<int>> fourSum(vector<int>& nums, int target){
      sort(nums.begin(), nums.end());
      return kSum(nums,. target, 4, 0);
      }
```

Length of the longest subarray with zero sum.

→ Example: {9, -3, 3, -1, 6, -5}

⇒ [[-3,3], [-1,6,-5], [-3,3,-1,6,-5]]

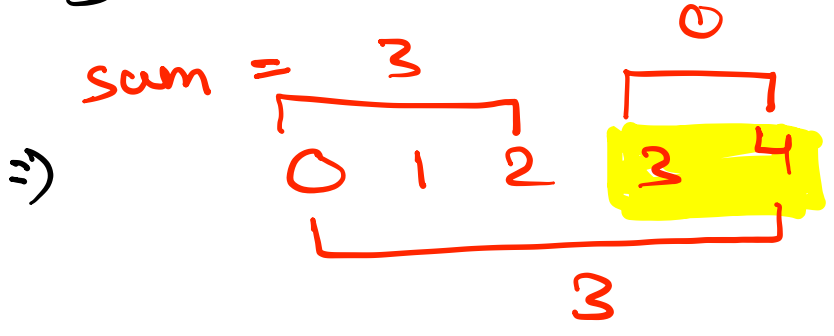
Example: 1, -1, 3, 2, -2, -8, 1, 7, 10

Hashmap = (1, 0); [sum = 1]

(prefixsum, index)

- Sum = 1 - 1 = 0 ⇒ max length = 2.
- Sum = 1 - 1 + 3 = 3
  Hashmap = (1, 0); (3, 2).
- Sum = 1 - 1 + 3 + 2 = 5
  Hashmap = (1, 0); (3, 2); (5, 3)
- Sum = 1 - 1 + 3 + 2 - 2 = 3

Now hashmap contains the sum 3 in it with index value = 2.



Current length = 4 - 2 = 2

- Sum = 1-1+3+2-2-8 = -5.

Hashmap = (1,0); (3,2); (5,3); (-5,5)

- Sum = -5+1 = -4

Hashmap = (1,0); (3,2); (5,3); (-5,5); (-4,6)

- Sum = -4+7 = 3

Hashmap = (1,0); (3,2); (5,3); (-5,5); (-4,6)

Now 3 exist at index 2 as well.

Length = 7-2 = 5

- Sum = 3+10 = 13

Hashmap = (1,0); (3,2); (5,3); (-5,5); (-4,6); (13,8)

```
int maxLen(vector<int> arr){
    int n = nums.size();
    int currLen = 0;
    unordered_map<int, int> mp;
    int maxi =0;
    int sum =0;
    for(int i =0; i< n; i++{
        sum += arr[i];
        if(sum == 0){
            maxi = i +1;
        }
    else{
        if(mp.find(sum) != mp.end()){
            maxi = max(maxi, i-mp[sum]);
            }
            else{
                    mp[sum] = i;
                }
            }
        }
return maxi;
```

Pattern Identification:

1. Longest subarray with given sum K.
2. Maximum subarray sum.
3. Count subarrays with given sum.
4. Longest subarray with sum = 0
5. K-sum.

Same

1. Longest subarray with given sum K.

very Important.

K = 3

0	1	2	3	4	5	6	7	8	9
1	2	3	1	1	1	1	4	2	3

arr[] = [1, 2, 3, 1, 1, 1, 1, 4, 2, 3].

prefixSum (sum, index) →

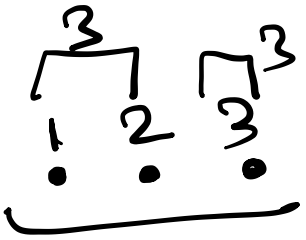
(1, 0) ; (3, 1)

As we find (1+2) = 3 at index = 1 ⇒

length = 2

(1,0) (3,1) (6,2)

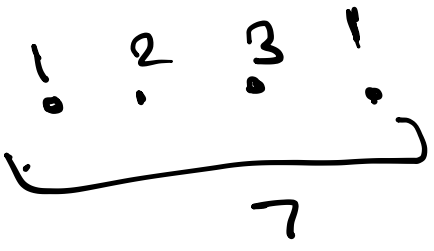
Now in the hashmap we have (3,1) ⇒



⇒ length = 2-1 = 1.

6 maxlen = max(1,2) = 2

Hashmap: (1,0),(3,1),(6,2),(7,3)

Now  ; I need a

4 in the hashmap but it not present.

Hashmap: (1,0),(3,1),(6,2),(7,3),(8,4)

We dont have 5 in hashmap.

so move ahead.

Hashmap: (1,0),(3,1),(6,2),(7,3),(8,4),
(9,5).

Now (9-3) = 6 ⇒ 6 is present in

the map, so indice(=5) - indice(=2) = 3

maxlen = max(3, 2) = 3

```
int maxLen(vector<int> arr){
    int n = nums.size();
    int currLen = 0;
    unordered_map<int, int> mp;
    int maxi =0;
    int sum =0;
    for(int i =0; i< n; i++{
        sum += arr[i];
        if(sum == 6){
            maxi = i +1;
        }
    else{
        if(mp.find(sum) != mp.end()){
            maxi = max(maxi, i-
mp[sum]);
            }
            else{
                    mp[sum] = i;
                }
            }
        }
return maxi;
};
```

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int n = nums.size();
        int cnt = 0;
        int sum = 0;
        unordered_map<int, int> mp;

        mp[0] = 1;  // To handle the case when subarray starts from index 0

        for (int i = 0; i < n; i++) {
            sum += nums[i];

            if (mp.find(sum - k) != mp.end()) {
                cnt += mp[sum - k];
            }

            mp[sum]++;
        }

        return cnt;
    }
};
```

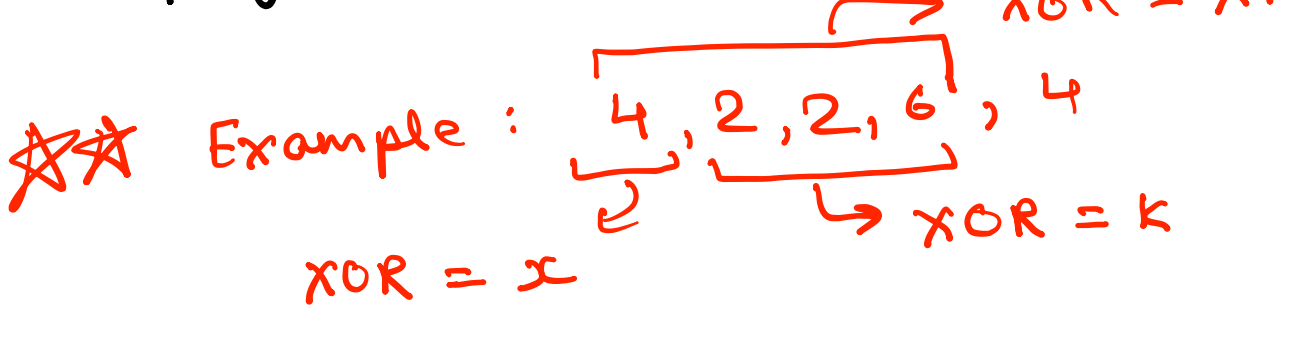
Question 2) count subarrays with given XOR k.

Properties of XOR: ① $A \wedge A = 0$

② $A \wedge 0 = A$

③ For an array arr, the XOR of subarray arr[i, ... j] is :

prefix XOR[j] ^ prefix XOR[i-1].



$\Rightarrow \quad x \wedge k = xR.$

$\Rightarrow \quad (x \wedge k) \wedge k = xR \wedge k.$

$\Rightarrow \quad \boxed{x = xR \wedge k}$ Remember.

Now, we will search for a subarray ending at 6 & having XOR of K.

For example in the above test:

4	2	2	6

$xR = 4 \wedge 2 \wedge 2 \wedge 6 = 2.$

$K = 6$

If $(x = xR \wedge K)$ exist in the array, then yes increment cnt.

$x = 2 \wedge 6 = 4$

XOR = 4 is present for the first element.

lets look the impkmentation.

Declare a hashmap.

Arr = [4 2 2 6 4] ; k = 6

hashmap : (front XOR, cnt)

= (0,1)

xR = 4 ; K=6 ; x = 2

2 is not present in hashmap.

hashmap = (0,1) ; (4,1)

xR = 4^2 = 6

K=6

x = 0 ⇒ 0 is present.

Thus cnt = 1

hashmap = (0,1) ; (4,1) ; (6,1)

$xR = 4 \wedge 2 \wedge 2 = 4$

$K = 6$

$x = 2$ ⇒ 2 not present.

hashmap = (0,1) ; (4,2) ; (6,1)

$xR = 4 \wedge 2 \wedge 2 \wedge 6 = 2$

$K = 6$

$x = 4$ ⇒ 4 is present.

⇒ cnt = 1+2 = 3

hashmap = (0,1) ; (4,2) ; (6,1) ; (2,1)

$xR = 4 \wedge 2 \wedge 2 \wedge 6 \wedge 4 = 6$

$K = 6$

$x = 0$ ⇒ 0 is present

⇒ cnt = $1 + 2 + 1 = 4$

Ans = 4

```
int subArraywithXORk(vector<int> arr, int k){
    int n = arr.size();
    int cnt = 0;
    unordered_map<int, int> prefixXOR;
    prefixXOR[0] = 1;
    int xr = 0;
    for(int i =0; i< n; i++){
        xr ^= arr[i];
        int x = xr^k;
        if(prefixXOR.find(x) != prefixXOR.end()){
            cnt += prefixXOR[x];
            }
        else{
            prefixXOR[xr]++;
            }
        }
    return cnt;
}
```

Q 19 Merge intervals : Pending

Example: [[1,3],[2,6],[8,10],[15,18]]

⇓

[[1,6], [8,10], [15,18]]

If last element > first element of the other block, then

merging will take place..

Q20) maximum Product Subarray:

→ $O(N^2)$ soln.

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

→ Optimal: $O(N)$.

- even negatives → return the product of the whole array.
- odd negatives →
- zeros → reset value to 1.

```cpp
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int maxi = INT_MIN;
        int n = nums.size();
        int suffix = 1;
        int prefix = 1;
        for(int i=0; i< n; i++){
            if(prefix == 0) prefix = 1;
            if(suffix == 0) suffix = 1;
            prefix *= nums[i];
            suffix *= nums[n-i-1];
            maxi = max(maxi, max(suffix, prefix));
        }
        return maxi;
    }
};
```