Q- Find the number that appear once, and others twice

arr[ ] = {1, 1, 2, 3, 3, 4, 4}

(Sorted assumption)

```
if not sorted then find
maxi = arr[0].
  for(i = 0 → n){
    if (A[i] > maxi){
      maxi = A[i]
  } }
```

Brute:

```
n = arr.size();
int maxi = arr[n-1];
int visi[maxi+1] = {0};

for (int i = 0 → n){
    visi[arr[i]]++;
}
for(int i = 0 ; i <= maxi ; i++){
    if (visi [i] == 1){
        cout << i
        break
    }
}
```

T.C ⤳ O(N+N)

S.C ⤳ O(N)

Optimal: for sorted & other appear twice only

```
for (i = 0 ; i < n-1 ; i+=2){
    if(A[i] != A[i+1]){
        cout << A[i] ;
    }
}

// if single element is last
cout << A[n-1]
```

T.C ⤳ O(n)

S.C ⤳ O(1)

Best Sol^n is XOR Method

hen find
J.
{
axi){
i]

XOR :

```
int XOR = 0;
for(i = 0 → n){
    XOR = XOR ^ A[i];
}
Cout << XOR
```

T.C ⤳ O(N)
S.C ⤳ O(1)

Q- Longest Subarray with Sum K

arr[] = {1,2,3,1,1,1,1,4,2,3} K=3

Generate all subarray
(i → j)

```
              j  j  j - - - - - - - - j
           j                          j
A[] = {1, 2, 3, 1, 1, 1, 1, 4, 2, 3}
       i     i
```

Brute :

```
int len=0;
for(i=0; i<n; i++){
   for(j=i j<n; j++){
      Sum=0
      for(k=i → j){
          Sum = Sum + A[k];
      }
      if(Sum==k) len= max(len; j-i+1);
   }
}
print(len)
```

T.C ⤳ O(n^3)
S.C ⤳ O(1)

We can optimize it by

```
for(i=0 → n){
  Sum=0;
  for(j=0 → n){
    Sum= Sum + A[j];
  } ~~if(Sum~~
  ~~if(Sum==k) len=max(len,j-i+1);~~
}
```

T.C ⤳ O(n^2)
S.C ⤳ O(1)

Note: Any time you deal with sum use
long long
especially when:
- prefix sum
- sliding window
- Subarray sum
- constraint have $10^5, 10^9$ etc.

Optimal:

```
int longestSub(int A[], int n, long long k){
    int left = 0, right = 0;
    long long sum = A[0]
    int maxlen = 0;

    while(right <n){
        while(left<= right && sum > k){              Shrink if Sum>k
            sum= sum - A[left]
            left ++;
        }

        if (sum==k){
            int len= right - left+1;                 Check if sum==k
            if(len > maxlen){
                maxlen=len;
            }
        }

        right ++;                                    Move right pointer
        if (right <n) {
            sum= sum + A[right];
        }
    }
    return maxlen;
}
```

{1, 2, 3, 1, 1, 1, 1, 3, 3}
$\boxed{k=6}$

Sum= ~~1~~ ~~3~~ ~~6~~ ~~7~~ ~~6~~ 7
~~5~~ ~~6~~ ~~7~~
~~4~~ ~~7~~ ~~6~~ ~~7~~
~~3~~ ~~7~~ 6

len= ~~0~~ ~~3~~ 4

Note: See the better Soln from striver bcz it is also a optimal Soln for problem where sum k(positive + negative)

3.2 Medium level

Q- Two Sum problem

arr[] = {2,6,5,8,11}
target = 14

(i) ANS in YES/NO for possible target

(ii) return index of both element

Brute:

```
A = {2,6,5,8,11}
n = sizeof(A)/size of(A[0]);
target = 14;

bool found = false;

for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
        if (A[i] + A[j] == target){
            found = true;
            break;
        }
    }
    if(found) break;
}

if(found) cout <<"YES";
else cout << "No";
```

T.C ⇝ $O(n^2)$
S.C ⇝ $O(1)$

Use hashing

Better:

here book → array given

```
String read(int n, vector<int> book, int target){
    map<int, int> mpp;
    for(int i = 0; i<n; i++){
        int a = book[i]
        int more = target - a;
        if(mpp.find(more)!= mpp.end()){
            return "YES";
        }
        mpp[a] = i;
    }
    return "No";
}
```

→ for index return{mpp[more], i};

T.C → O(NlogN)
S.C → O(N)

2 pointer

Optimal:

- first sort the array (Quicksort)

$$A = \{\underset{\overset{\uparrow}{\text{left}}}{2},5,6,8,\underset{\overset{\uparrow}{\text{right}}}{11}\}$$

ONLY for 1st ~~variety~~, optimal exist

```
String read(int n, vector<int> book, int target){
    int left = 0, right = n-1;
    sort(book.begin(), book.end());
    while(left < right){
        int sum = book[left] + book[right];
        if(sum == target) return "YES";
        else if (sum < target) left++;
        else right--;
    }
    return "No";
}
```

T.C → O(NlogN)+O(N)
S.C → O(1)

Q- Sort an array of 0's, 1's and 2's

arr[] = {0,1,2,0,1,2,1,2,0,0,0,1}

Brute: use any of sorting technique like Merge sort

note: here are only three distinct value and Quick sort ignores property and interviewer expect you to exploit constraints ·use merge sort

T·C ⇝ O(NlogN)

S·C ⇝ O(N)

Better:

```
cnt0 = 0 , cnt1 = 0 , cnt2 = 0
for(i = 0 → n){
  if (arr[i] == 0) cnt0 ++;
  elseif (arr[i]==1) cnt1 ++;
  else  cnt2 ++;
}

int idx = 0;
while (cnt0){
      arr[idx ++] = 0;
}
while (cnt1){
      arr[idx ++] =1;
}
while (cnt2){
        arr[idx ++] = 2;
}

for (i=0→n)
   cout << a[i] << " ";
```

use (cnt 0 --)

T·C ⇝ O(N)

S·C ⇝ O(1)

Optimal: Dutch National Flag algorithm

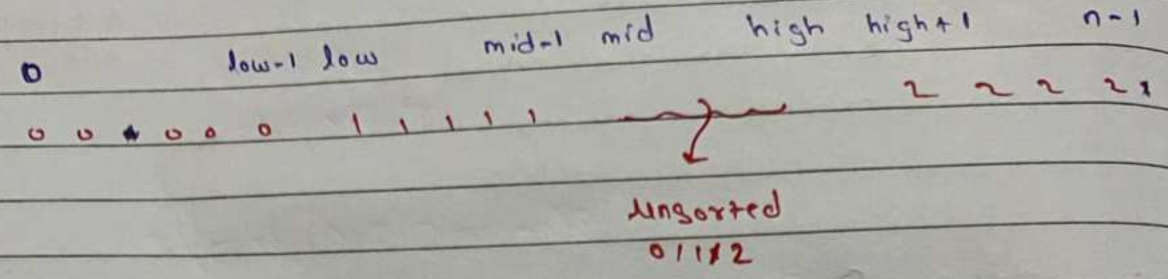
A = [0,1,2,0,1,2,1,2,0,0,0,1]

↑ mid ↑ high

[0.... low-1] ⟶ 0's

[low.... mid-1] ⟶ 1's

[high+1, n-1] ⟶ 2's



```
(a[mid]==0){
    swap(a[low], a[mid]);
    low++, mid++

(a[mid]==1){
        mid++;
}
(a[mid]==2){
        swap(a[mid], a[high]);
        high--;
```

see code of this from github/GPT

Q- Majority element (> N/2 times)

A = [2 2 3 3 1 2 2]

Brute:

```
for(i=0→n)
    cnt=0;
    for(j=0→n)
        if(A[j] == A[i])
            cnt++;
    }
  if (cnt > N/2) return arr[i];
}
```

T.C → O(N2)
S.C → O(1)

Better:

```
int maxi = A[0];
for (i=1→n){
    if(A[i] > maxi) maxi = A[i];
}
```

also use hashing for better soln

```
int visi[maxi+1] = {0}

for (i=0→n){
    visi[A[i]]++;
}

target= n/2 ;

for (i=0; i<=maxi; i++){
    if(visi[i] > target){
        cout << i
        break;
    }
}
```

T-C → O(N)
S.C → O(N)

Optimal: Moore's voting algorithm

```
int majElem(vector <int> v){
    int cnt = 0;
    int el;
    for(int i = 0; i < v.size(); i++){
        if(cnt == 0){
            cnt==1;
            el = v[i];
        }
        else if(v[i] == el){
            cnt++;
        }
        else{
            cnt--;
        }
    }
    int cnt1 = 0;
    for(int i = 0; i < v.size(); i++){
        if(v[i] == el) cnt1 ++;
    }
    if(cnt1 > v.Size()/2)){
        return el;
    }
    return -1;
}
```

T.C → O(N)

S.C → O(1)

Q- Maximum Subarray Sum

A[ ] = {-2, -3, 4, -1, -2, 1, 5, -3]

ANS→ 7

Note: For the Brute soln just copy paste the brute soln of longest subarray with sum K · only remove the "if" part and rest same

~~↓~~ ~~↓~~ ~~↓~~ ~~↓~~ ~~↓~~ ↓

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

maxi = ~~-2~~ ~~4~~ 7

Sum = ~~0~~ ~~-2~~ 0 ~~-3~~ 0 ~~4~~ ~~3~~ ~~1~~ ~~2~~ ~~7~~ 4

optimal: Kadane's algorithm

```
long long maxSubSum(int arr[ ], int n){
    long long sum=0, maxi = LONG_MIN;
    for(int i=0; i<n; i++){
        Sum = Sum + arr[i];

        if(sum > maxi){
            maxi = Sum;
        }


        if(Sum < 0){
            Sum = 0;
        }
    }

    return maxi;
}
```

Note: if all element in array are -ve and you need to find and compute maxi to be a -ve value, alu answer a emp Subarray bez it has sum=0 wh is greater than -

Do it later

Q- Best Time to buy & Sell Stocks
arr[] ⇝ [7, 1, 5, 3, 6, 4]
Maximise the profit

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

T.C ⇝ O(N)
S.C ⇝ O(1)

Q- Rearrange array elements by sign
A[] = {3, 1, -2, -5, 2, -4}
output {3, -2, 1, -5, 2, -4}
there will always be equal positive & equal negative

order ⤵
(+, -, +, -, +, -)
OR
(-, +, -, +, -, +)

Note : order of occurence will be same even in output

for ex: input ⇝ 3, 1, -2
output ⇝ 3 - 1 - 2

Brute:

```
int p = 0, q = 0
int pos[n/2];
int neg [n/2];
```

T.C ⇝ O(2N)
S.C ⇝ O(N)

Step 1: inserting positive/negative separately

```
for(i = 0 -> n){
    if(arr[i] > 0){
        pos[p] = arr[i];
        p++;
    }
    else{
        neg [q] = arr[i];
        q++;
    }
}
```

```
int i=0, j=0, k=0 ;
while(i<p && j<q){
    arr[k++] = pos[i++];
    arr[k++]= neg[j++];
}
```

Optimal: We know that all positive are at even index (0,2,4,...)
all negative are at odd index (1,3,5,...)

We iterate and check element and place on its respective index

```
n = nums.size();
vector <int> ans;
int posIndex=0; negIndex=1;
for (int i=0; i<n; i++){
    if(nums[i] <0){
        ans [negIndex] = nums[i];
        negIndex = negIndex +2;
    }else{
        ans [posIndex] = nums[i];
        posIndex +=2;
    }
}
return ans;
```

Variety 1 is done where we have equal negative and positive element variety 2 : almost same like 1 but if both are not equal, after inserting alternately. insert the rest in any order

~~we can't use optimal sol^n now~~

```
int p=0, q=0
int pos[n]; neg[n]
```

Brute: step 1 will remain same

collect all positive & negative (till pos == neg)

don't collect all at once because we still can apply the same for (pos = neg) case.

```
arr[] = {-1,2,3,4,-3,1}
  for(i=0; i<2; i++){
        // collect
      - - -
      - - -
  }
 i=0,j=0,k=0
 while(i<p && j<q){
     arr[k++] = pos[i++];
     arr[k++] = neg[i++];
 }
index=4;
for remaining element
for(i=2; i<p; i++){
  arr[index] = pos[i]
  index++;
}
```

worst $\rightsquigarrow$ O(N/2)

O(N/2

T·C $\rightsquigarrow$ O(N) + O(min(p,q)) + O(leftov

$\rightsquigarrow$ O(2N)

S·C $\rightsquigarrow$ O(N)

Q- Next Permutation

A[ ] = {3,1,2}

output : {3,2,1}

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

Brute:

a) Generate all permutation (in sorted order)
b) Linear search to find given permutation
c) return next index

it will be a lot of time taking

Note: in C++ STL, we have funcⁿ for next permutation

```
next_permutation(A.begin(), A.end());
return A;
```

optimal:

```
idx = -1;
for(i = n-2 ; i >= 0 ; i--){
    if(a[i] < a[i+1]){
        idx = i;
        break;
    }
}
```

~> find longest prefix match

```
if(idx == -1){
    reverse(A.begin(), A.end());
```

```
for(i = n-1 ; i >= 0; i--){
    if(a[i] > a[idx]){
        swap(a[i], a[idx]);
    }
}
rev(a, idx+1, n-1);
```

~> find > A[idx] but the smallest one so that you can stay close

T.C ~> O(3N)
S.C ~> O(1) → extra space

everything on the right should be smaller

Q- Leaders in an array

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

output ⤳ {22, 12, 6}

observation :
last element will
always be the
leader

Brute:

```
vector<int> leaders
for(i= 0 → n){
  bool isLeader = true;
   for(j = i+1 → n ){
      if ( A[j] >= A[i] ){
          isLeader = false ;
          break ;
      }
  }
  if (isLeader){
      leaders.push_back(A[i]);
  }
```

T.C ⤳ O(n²)

S.C ⤳ O(N)

optimal:

```
vector<int> ans ;
int maxi = INT_MIN ;
for (int i = n-1 ; i >=0 ; i--){
    if(a[i] > maxi){
        ans.push_back(a[i]);
    }
    maxi = max(maxi , a[i]);
}
```

Start from last of array

idea 1) Stay at any index
2) find maxi element from right
3) if current > maxi ⤳ leader

T.C ⤳ O(N)

S.C ⤳ O(1) → extra space

Q- Longest consecutive sequence

arr[] = {102, 4, 100, 1, 101, 3, 2, 1, 1}

output ⤳ 4 (len=4) {1, 2, 3, 4}

Brute:

```
longest = 1;
for(i = 0; i<n; i++){
    x = arr[i]
    cnt = 1;
    while(ls(arr, x+1) == true){
        x = x+1;
        cnt = cnt+1;
    }
}
```

T.C ⤳ O(n²)
S.C ⤳ O(1)

Better:

```
sort(nums.begin(), nums.end());
int n = nums.size();
int lastsmaller = INT_MIN;
int cnt = 0;
int longest = 1;
for(int i=0; i<n; i++){
    if(nums[i]-1 == lastsmaller){
        cnt = cnt+1;
        lastsmaller = nums[i];
    }
    elseif{ lastsmaller != mums[i]){
        cnt=1;
        lastsmaller = nums[i];
    }
    longest = max(longest, cnt);
}
return longest;
```

T.C ⤳ O(NlogN + N)

Optimal:

```
int longest = 1;
unorderd_set <int> st;
for(int i = 0; i<n; i++){
    st.insert(a[i]);
}
for (auto it : st){
    if (st.find(it-1) == st.end()){
        int cnt = 1;
        int x = it;
        while (st.find(x+1) != st.end()){
            x = x+1;
            cnt = cnt+1;
        }
        longest = max(longest, cnt);
    }
}
return longest;
```

T.C ⇝ O(3N)
S.C ⇝ O(N)

Q- Set Matrix zeros (nxM)

row Column

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

Whenever you find a zero. make its complete row & column to zero

Output

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

Note: To traverse in a 2D array you need min T.c ⇝ $O(n^2)$ so you cannot make it less than n^2

Brute:

```
for(int i=0; i<n; i++){
    for(int j=0; j<m; j++){
        if (arr[i][j]==0){
            markRow(i);
            markRow(j);
        }
    }
}

for(i=0→n){
    for(j=0→n){
        if(arr[i][j]==-1){
            arr[i][j]==0;
        }
    }
}
```

```
markRow(i){
    for(j=0; j<m; j++){
        if (arr[i][j]==0){
            arr[i][j]=-1;
        }
    }
}

markRow(j){
    for(i=0; i<n; i++){
        if(arr[i][j]==0){
            arr[i][j]=-1;
        }
    }
}
```

T.C ⇝ O(n×m×(n+m)) + O(n×M)
S.C ⇝ O(1)

Better

```
col[m] = {0}, row[n] = {0};
for(i=0→n){
    for(j=0→m){
        if(arr[i][j]==0){
            row[i]=1;
            col[j]=1;
        }
    }
}

for(i=0→n){
    for(j=0→m){
        if(row[i] || col[j]){
        arr[i][j]=0;
        }
    }
}
```

T.C ⇝ O(n×M) + O(n×M)
S.C ⇝ O(n) + O(m)

for optimal one ⇝
you just need to check
space bcz time is
already optimised

Note: watch optimal from striver bcz it asked directly in interviews many times

Q- Rotate Matrix / Image by 90° (n×n)

$$\begin{matrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\\9 & 10 & 11 & 12\\13 & 14 & 15 & 16\end{matrix} \xrightarrow{\text{rotate}} \begin{matrix}13 & 9 & 5 & 1\\14 & 10 & 6 & 2\\15 & 11 & 7 & 3\\10 & 12 & 8 & 4\end{matrix}$$

initially at	finally at
1st row →	last column
2nd row →	2nd last column
3rd row →	2nd column
4th row →	1st column

Brute:

i j → j (n-1)-i

{0} {0} → {0}{3}
{0} {1} → {1}{3}
{0} {2} → {2}{3}
{0} {3} → {3}{3}

j (n-1)-i

{1} {0} → {0}{2}
{1} {1} → {1}{2}
{1} {2} → {2}{2}
{1} {3} → {3}{2}

```
ans[n][n];
for(i=0→n){
    for(j=0→n){
        ans[j][n-1-i] = matrix[i][j]
    }
}
```

T.C → O(n^2)
S.C → O(n^2)

Optimal: Take transpose → reverse all row

How to take transpose :

matrix: 1) The diagonal will stay same even after doing transpose

2) The upper & lower triangular element swap

{0}{1} → {1}{0}　{1}{2} → {2}{1}
{0}{2} → {2}{0}　{1}{3} → {3}{1}
{0}{3} → {3}{0}　{2}{3} → {3}{2}

{i}{j} → {j}{i}

Observation: first column becomes the first row but in a reverse order

Second column becomes ~~the second row but~~ in a reverse order

this is what we call

Transpose → change column to row or vice versa

```
for(i=0 → n-1){
  for(j = i+1 → n){
     swap (a[i][j] , a[j][i]);

  }
}
```

T-C ⇝ O(n²)
S·C ⇝ O(1)

```
for(i=0→n){
  reverse(a[i].begin() , a[i].end());
}
```

Q- Spiral matrix

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

- it has only one solution
- interviewer will ask this to test your implementation and how clean you can write the code

top = 0 bottom = 5
left = 0 right = 5

right → bottom → left → top

watch the code from github/striver

left/right
a[,][ ]
top/bottom

whatever is constant put it there and another one will always be i

Q- Number of subarrays with sum k

a[] = {1,2,3,-3,1,1,1,4,2,-3} k=3

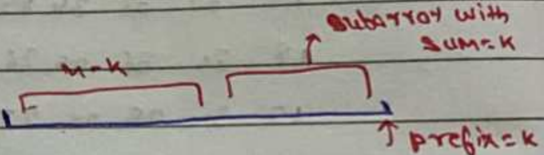
Brute: Generate all subarray and check for which sum=k

```
cnt = 0;
for(i=0→n)
    sum = 0;
    for(j=i→n){
        sum= sum + arr[j];
        if(sum==k){
            cnt ++;
        }
    }
}
```

T.C ⇝ $O(n^2)$

S.C ⇝ $O(1)$

Optimal: using prefix sum



no. of x-k = no. of subarray with k

Hashmap(preSum, cnt)

Key Value

```
mpp[0]=1;
int presum=0, cnt=0;
for(i=0→n){
    presum= presum + arr[i];
    int remove = presum-k;
    cnt= cnt + mpp[remove];
    mpp[presum]= mpp[presum]+1;
}
return cnt;
```

T.C ⇝ O(nlogn)

S.C ⇝ O(n)