Day 1 (01/06)

2nd largest number

```
int getSecondLargest (vector<int> & arr) {
    int largest = -1 , second_largest = -1;
    int size = arr.size();
    for (int i=0 ; i<size ; i++) {
        if ( arr[i] > largest ) {
            largest = arr[i];
        }
    }
    for (int i=0; i<size ; i++) {
        if ( arr[i] > second_largest &&
             arr[i] != largest) {
            second_largest = arr[i];
        }
    }
    return second_largest
}
```

OUTPUT
22

→ second_largest = INT_MIN // if negatives are there

```
int main() {
    vector<int> arr = {10, 20, 15, 5, 37, 22};
    cout << getSecondLargest (arr);
    return 0;
}
```

Push zeros to End (BRUTE APP)

```
void pushZeroesToEnd (vector<int> & arr){
    vector<int> array ;
    int size = arr.size();
    for (int i=0 ; i<size ; i++) {
        if (arr[i] != 0) {
            array.push_back(arr[i]);
        }
    }
    for (int j = array.size() ; j<size ;
        j++) {
        array.push_back(0);
    }
    for (int i=0 ; i<size ; i++) {
        arr[i] = array[i];
    }
}
int main () {
    vector<int> arr = {1, 2, 0, 4, 3, 0}
    pushZeroesToEnd (arr);
    return 0;
}
```

OUTPUT
1 2 4 3 0 0

Two sum

```
vector<int> twoSum (vector<int> & nums,
                    int target) {
    for(int i=0 ; i<nums.size() ; i++) {
        for (int j=i+1 ; j<nums.size() ; j++)
            if ( nums[i]+nums[j] == target ) {
                return {i, j};
            }
    }
    return { };
}
int main() {
    vector<int> nums = {2, 7, 11, 15};
    int target = 9;
    vector<int> result = twoSum (nums, target);
    for ( int i : result) {
        cout << i << " ";
    }
    return 0;
}
```

OUTPUT
{ 0, 1 }

Remove Duplicates from an sorted array

{ 1, 1, 2, 2, 2, 3 }

BRUTE

```
set<int> st        // only contains unique elements
for (i=0 ; i<n ; i++)
    st.insert (arr[i])        // NlogN
index = 0 ;
for ( auto it : st ) {        // iterator in set
    arr[index] = it ;
    index ++;
}                             // → N
```

3
2
1

output = { 1, 2, 3, -, -, - }

T(c) = NlogN + N

S(c) = O(N)

OPTIMAL T(c) = O(N) , S(c) = O(1)

```
int removeDuplicates (vector<int> & nums){
    int j = 0
    int size = nums.size() ;
    for (int i = 1 ; i<size ; i++) {
        if ( nums[j] != nums[i]){
            j++;
            nums[j] = nums[i] ;
        }
    }
    return j+1 ;
}
```

Reverse array

```
void reverseArray (vector<int> & arr) {
    vector <int> brr;
    int n = arr.size();
    for (int i = n-1 ; i >= 0 ; i++) {
        brr.push_back (arr[i]);
    }
    for (int i = 0 ; i > n ; i++) {
        arr [i] = brr [i];
    }
}
```

Approach → Brute → Better → Optimal

Q) Left rotate Array by one place :

OPTIMAL:

```
vector<int> rotateArray (vector <int> & arr,
                         int n ) {
    int temp = arr [0];
    for (int i = 1 ; i < n ; i++) {
        arr [i-1] = arr [i];
    }
    arr [n-1] = temp;
    return arr;
}
```

Left rotate Array by D places

$\boxed{D = D \% N}$

Eg : arr = { 1, 2, 3 } 5 }

D = 1	=> { 2, 3, 1 }
D = 2	=> { 3, 1, 2 }
D = 3	→ { 1, 2, 3 } => D = 0
D = 4	=> D = 1

Shifting

D = 2

	index	i.e (i - D)
1 →	0	
2 →	1	
3 →	0	0 - 2 = 2
4 →	1	1 - 2 = 1
5 →	2	
6 →		

```
for (i = d ; i < N ; i++) {
    a [ i-d ] = a[i];
}
```

Put back temp

```
j = 0
for (i = n-k ; i < n ; i++) {
    a [i] = temp [ j ] ;
    j++ ;
}
```

Brute Code:

```
void leftRotate (int arr[], int n, int d) {
    d = d % n ; // repeats after n
    int temp [d] ;
    for (int i = 0, i < d ; i++)
    {
        temp [i] = arr[i] ;
    }
    for (int i = d ; i < n , i++) {
        arr [i-d] = arr [i] ;
    }
    for (int i = n-d ; i < n ; i++) {
        arr [i] = temp [i-(n-d)] ;
    }
}
int main() {

    leftRotate (arr, n , d) ;
        cout << arr [i] << " " ;
    return 0 ;
}
```

T.C → O(d) + O(n - d) + O(d) = O(n + d)

S.C → O (d) .

⇒ change S.C [O(d) → O(1)]

OPTIMAL

D = 3

{ 1, 2, 3, 4, 5, 6, 7 }

Reverse Reverse

Reverse ⟶ { 3, 2, 1, 7, 6, 5, 4 }

{ 4, 5, 6, 7, 1, 2, 3 } ⟶ Req'd. soln

```
void leftRotate (int arr[], int n, int d) {
    reverse (a, a+d)
    reverse ( a+d, a+n)
    reverse ( a, a+n)
}
```

i.e,

TC → O(2n)

SC → O(1)

Q) Push zeros to End (OPTIMAL) → 2 pointers

TC → O(n)
SC → O(1)

```
int j = -1;
for (int i=0 ; i<n ; i++) {
    if (a[i] == 0) {
        j = i;
        break;
    }
}
if (j == -1) return a;
for (int i = j+1 ; i<n ; i++) {
    if (a[i] != 0) {
        swap (a[i], a[j]);
        j++;
    }
}
```

j will always be at 0

[1, 0, 2, 3, 2, 0, 4, 4, 5, 1, 0, 0, 0]

Q) union of two sorted arrays

⇒ Add elements of both arrays without duplicates.

unique : map, set

Brute ~~vector~~ :

Step 1:

```
Set <int> st
for (i=0 ; i<n1 ; i++) {
    st.insert (a1[i])
}
for (i=0 ; i<n2 ; i++) {
    st.insert (a2[i]).
}
```

$n_1 = a_1.size()$;
$n_2 = a_2.size()$;

→ $O(n_1 \log n)$

→ $(n_2 \log n)$

Step 2:

```
union [st.size()]
i=0
for (auto it : st)
    union [i++] = it;
```

→ $O(n_1+n_2)$

→ for vector : union.add(it)
union.push_back(it)

SC → $O(n_1+n_2) + O(n_1+n_2)$

TC → $O(n_1 \log n + n_2 \log n) + O(n_1+n_2)$

```
vector<int> sortedArray (vector<int> a,
                         vector<int> b)
{
    int n1, n2;
    [Step 1]
    vector <int> temp;
    for (auto it : st) {
        temp.push_back (it);
    }
    return temp
```

Optimal (2 - pointer)

compare smallest element from both array and insert into resultant.

→ same as Addition of polynomials

Code

```
vector<int> sortedArray (vector<int> a,
                         vector<int> b)
{
    int n1 = a.size(), n2 = b.size();
    int i=0, j=0;
    vector<int> union Arr;
    while (i<n1 && j<n2) {
        if (a[i] <= b[j]) {
            union Arr.size() == 0 ||
            if (union Arr.back() != a[i]) {
                union Arr.push_back (a[i]);
            }
            i++;
        }
        else {
            a[i] → b[j]
            i++ → j++
        }
    }
    // one array iterated
    while (j<n2) {
        if (union Arr.size() == 0 ||
            union Arr.back() != b[j]) {
            union Arr.push_back(b[j]);
        }
        j++;
    }
    while (i<n1) {
        // Paste
    }
```

copy

PASTE

T.C : $O(n_1+n_2)$

S.C : $O(n_1+n_2)$ if worst case

↓

all elements unique

Intersection of 2 sorted arrays

(present in both arrays / unique)

```
vector<int> findArrayIntersection(vector<int>& A,
    vector<int>& B, int m)
{
    int i=0, j=0;
    vector<int> ans;
    while (i<n && j<m) {
        if (A[i] < B[j]) {
            i++;
        }
        else if (B[j] < A[i]) {
            j++;
        }
        else {
            ans.push_back(A[i]);
        }
    }
    return ans;
}
```

T.C : $O(n_1+n_2)$
S.C : $O(1)$

Q) Remove element

```
int removeElement(vector<int>& nums,
                  int val)
{
    int size = nums.size();
    int k=0;
    for(int i=0; i<size; i++) {
        if (nums[i] != val) {
            nums[k] = nums[i];
            k++;
        }
    }
    return k;
}
```

If val = Last element

L.C

Q) Remove Duplicates

T.C : $O(\log n)$

```
int removeDuplicates(vector<int>& nums){
    int l=0, r=0;
    int n = nums.size();
    while (r<n) {
        int count = 1;
        while ((r+1) < n &&
            nums[r] == nums[r+1]) {
            r++,
            count,
        }
        int cnum = min(count, 2);
        for (int i=0; i<cnum; i++) {
            nums[l] = nums[r];
            l++;
        }
        r++;
    }
    return l;
}
```

for resultant array index posn

min = 1
if cnum=2
then duplicate found

S.C : $O(1)$

Q)

T.C : $O(\log n)$

```
int searchInsert(vector<int>& nums, int
                 target)
{
    int n = nums.size();
    int low = 0;
    while (low<n) {
        int mid = (low+n)/2;
        if (nums[mid] == target) {
            return mid;
        }
        else if (nums[mid] < target) {
            low = mid + 1;
        }
        else {
            n = mid;
        }
    }
    return low;
}
```

BINARY

// index posn

Remove Duplicates

BRUTE :	TC	SC
Set :	$O(N\log N)+O(N)$	$O(N)$

Q) Find Missing numbers in array

Given : N = 5 and (N-1) numbers

arr[] = {1, 2, 4, 5}

3 ?

Brute Force Sol^n

```
for (i=1 ; i<=N ; i++) {
    flag = 0
    for ( j=0 ; j<n-1 , j++) {
        if (arr[j] == i) {
            flag = 1
            break
        }
    }
    if (flag == 0) {
        return i;
    }
```

T.C : $O(N^2)$

Better Sol^n (Hashing)

0	1	1	0	1	1	3
0	1	2	3	4	5	6

→ sol^n

```
hash{n+1} = {0} ;
for (i=0 ; i<n ; i++) {
    hash{arr[i]} = i ;
}
for (i = 1 → n) {
    if (hash[i] == 0) {
        return i ;
    }
}
```

T.C → O(2N)
S.C → O(N)

Optimal Sol^n → Sum → XOR

① Sum = (n*(n+1)) / 2

```
S2 = 0
for (i = 0 → n)
    S2 + = arr[i]
return (sum - S2)
```

T.C : O(N)
S.C : O(1)

② Concept : $x \wedge x = 0$, $x \wedge 0 = x$

```
int XOR1 = 0, XOR2 = 0 ;
for (i = 1 → N)
    XOR1 = XOR1 ^ i ;
    XOR2 = 0
for (i = 0 ; i < N-1)
    XOR2 = XOR2 ^ arr[i]
return XOR1 ^ XOR2
```

O(2N) ✗

O(N)

→ XOR2 = XOR1 ^ (i+1)

→ XOR1 = XOR1 ^ n

Q) Consecutive ones (Max^m) [L-0 48]

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

2 3 2

ans = 3

```
int findMaxConsecutiveOnes (vector<int>
    & nums) {
    int maxi = 0 ;
    int cnt = 0 ;
    for (int i = 0 ; i<nums.size() ; i++){
        if (nums[i] == 1) {
            cnt++ ;
            maxi = max(maxi, cnt) ;
        }
        else {
            cnt = 0 ;
        }
    }
    return maxi ;
}
```

21:42

Q) Find the no. that appears once

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

Brute : T.C : $O(n^2)$; S.C : O(1)

```
for (i = 0 → n) {
    num = arr[i]
    count = 0
    for (j = 0 → n) {
        if (arr[j] == num) {
            count++
        }
    }
    if (count == 1) return num;
}
```

Better : Hashing

T.C : O(3n)
S.C : $O(n \wedge n^2)$

0	1	2	3	4
0	2	1	2	2

```
maxi = arr[0]
for (i = 0 ; i<n ; i++)
    maxi = max(maxi, arr[i]) ;
hash[maxi] = [0]
for (i = 0 → n)
    hash{arr[i]}++ ;
for (i = 0 → n)
    if (hash{arr[i]} == 1)
```

SORTING

① Selⁿ Sort

13	46	24	52	20	9

In ascending order

9, 13, 20, 24, 46, 52

→ Get the minimum at the first Here (swap (13, min))

→ First element sorted 9, 46, 24, 52, 20, 13

Get minimum at 1st index pos n

9, 13, 24, 52, 20, 46

→ 9, 13, 20, 52, 24, 46

→ 9, 13, 20, 24, 46, 52 ✓

(6 element → 5 steps)

Swapping takes place b/w

Index	
0	min
1	min
n-1	min

```
for (i = 0 ; i <= n-2 ; i++) {
    min = i; //whoever is the first guy
    for ( j = i+1 ; j <= n-1 ; j++) {
        if ( arr [j] < arr [min] ) {
            min = j ;
        }
    }
    swap ( arr [min], arr [i] );
    ↳ using temp
}
```

TC : $\frac{n + n-1 + n-2 + \dots 2 + 1}{} = \frac{n(n+1)}{2} = O(n^2)$

SC :

→ Best / Avg. / Worst

② Bubble Sort

→ 13, 46, 24, 52, 20, 9

→ 13, 24, 46, 20, 9, 52 ↳ max at last

✓ ✓ ✗

13, 24, 20, 46, 9, 52

✗

→ 13, 24, 20, 9, 46, 52 ↳ 2 max at last

✓ ✗

13, 20, 24, 9, 46, 52

✗

→ 13, 20, 9, 24, 46, 52

✓ ✗ ✓

13, 9, 20, 24, 46, 52

→ 9, 13, 20, 24, 46, 52

(5 steps → 6 element)

```
for (int i = n-1 ; i >= 0 ; i--) {
    for (int j = 0 ; j <= i-1 ; j++) {
        if ( arr[j] > arr [j+1] ) {
            int temp = arr [j+1] ;
            arr[j+1] = arr [j] ;
            arr [j] = temp ;
        }  did swap = 1 ;
    } if (did == 0)
        break ;
}
```

TC : $O(n^2)$ → worst → Best $O(N)$

SC :

Insertion Sort

```
for (int i = 0 ; i <= n ; i++) {
    int j = i ;
    ~~for (int~~
    while ( j >= 0 && arr[j-1] > arr[j])
        int temp = arr[j-1] ;
        arr[j-1] = arr[j] ;
        arr[j] = temp ;

        j - - ;
    }
}
```

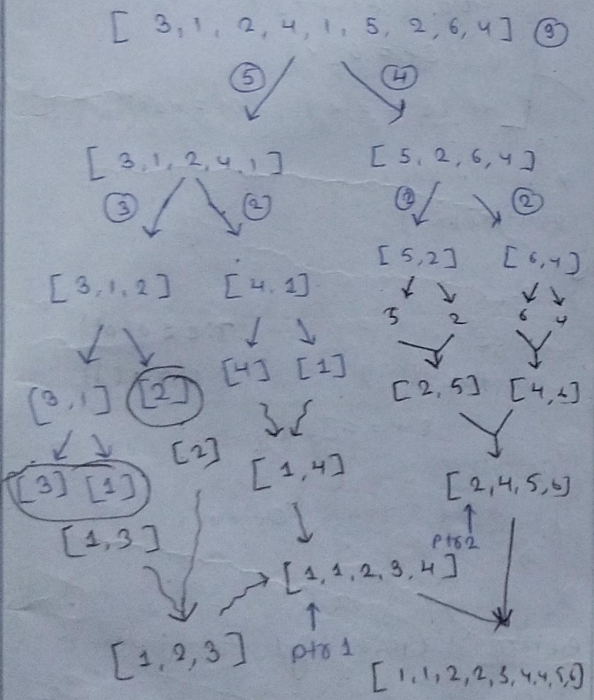
TC : $O(n^2)$ → worst $O(n)$ → Best

SC :

MERGE SORT ?? (Much more optimised)

↓

DIVIDE & MERGE



TC : $O(N \times \log_2 n)$ → Best / worst / avg.

SC : $O(N)$ → worst & temp variable

```
void merge (vector<int> & arr, int low,
    int mid, int high) {
    vector<int> temp ;
    int left = low ;
    int right = mid + 1 ;
    while ( left <= mid && right <= high)
    {
     if (arr[left] <= arr[right]) {
        temp.push_back (arr[left]);
        left ++ ;
     }
     else {
        temp.push_back (arr[right]);
        right ++ ;
     }
    }
    while (left <= mid ) {
        temp.push_back (arr[left]);
        left ++ ;
    }
    while (right <= high ) {
        temp.push_back (arr[right]);
        right ++ ;
    }
    for (int i = low ; i <= high ; i++) {
        arr[i] = temp [i - low] ;
    }
}

void ms (vector<int> & arr, int low,
    int high) {
    if ( low = = high) return ;
    int mid = (low + high) / 2 ;
    ms ( arr, low, mid) ;
    ms (arr, mid+1, high) ;
    merge (arr, low, mid, high) ;
}

void mergesort (vector<int> & arr, int n) {
    ms (arr, 0, n-1) ;
}
```

QUICK SORT

ascending order

arr[] = [4 6 2 5 7 9 1 3]

1. Pick a (pivot) → any random element & place it in its correct place in the sorted array
2. Smaller on left, larger on right

```
int partition(vector<int>& arr, int low,
              int high) {
    int pivot = arr[low];
    int i = low, j = high;
    while (i < j) {
        while (arr[i] <= pivot && i <= high - 1) {
            i++;
        }
        while (arr[j] > pivot && j <= low+1) {
            j--;
        }
        if (i<j)  swap(arr[i], arr[j]);
    }
    swap(arr[low], arr[j]);
    return j;
}
```

T.C : O(N logn)

S.C → O(1)

```
void qs (vector<int> & arr, int low,
int high) {
    if (low < high) {
        int pIndex = partition(arr, low, high);
        qs = (arr, low, pIndex - 1);
        qs = (arr, pIndex+1, high);
    }
}
```

```
vector<int> quickSort (vector<int> arr)
{
    qs (arr, 0, arr.size() - 1);
    return arr;
}
```

Q) Find longest subarray with sum K (tve)

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

K = 3
↳ sum of subarray

Subarray : contiguous part of array

Brute :

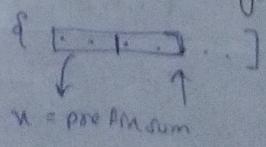
Generate all subarray, calculate sum, longest sub-arr is answer

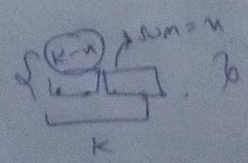
```
for ( i = 0 → n ) {
    for ( j = i → n ) {
        s = 0                          T.C ≈ O(n^3)
        for (k = i → j)
            s += a[k]
        if (s == k)  len = max(len, j-i+1)
    }
}
print(len);
```

```
for (   )
    s = 0                              T.C ≈ O(n^2)
    for (   )                          S.C ≈ O(1)
        s += a[j];

    if (s == k)  len = max(   )
}
}
```

Better : (Hashing)





Optimal for +ve / -ve

```
int longestSubarrayWithSumK (vector<int>a,
long long K) {
    map<long long, int> preSumMap;
    long long sum = 0;
    int maxlen = 0;
    for (int i = 0; i < a.size(); i++) {
        sum += a[i];
        if (sum == K) {
            maxlen = max(maxLen, i+1);
        }
        long long ~~int~~ rem = sum - K;
        if (preSumMap.find(rem) !=
            preSumMap.end())
            int len = i - preSumMap[rem];
            maxLen = max(maxLen, len);
        }
        if (preSumMap.find(sum) ==
            preSumMap.end())
            preSumMap[sum] = i;
        }
    }
    return maxLen;
}
```

TC: O(N log N)
↓ ordered map

O(N×1)
↳ unordered

O(N²) → worst

SC: O(N)

OPTIMIZE (2 pointer approach)

Reverse Engineering method

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

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

ans = 5 (K = 3)

(8, 2)
(3, 1)
(1, 0)

maxlen = 2

~~sum = 1, index = 0~~

count = 0

rem = 3 ✓ len = 2 - (1)

sum = ~~1~~ 3

index = ~~0~~ 1

(8, 7)
(7, 3)
(6, 2)
(3, 1)
(1, 0)

i	sum	rem	maxlen
0	1	-2	0
1	3	0	2
2	6	3	2
3	7	4	2
4	8	5	2
5	9	6	3
6	10	7	3
7	14	10	

-2 ✗
0 ✗
3 ✓ len = 2 - 1 = 1
maxlen = 2
4 ✗
6 ✓
len = 5 - 2 = 3
9 ✓
len = 3

3

Better: (Hashing)

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

sum = 0 len = 0 2

1 2 3 1

2 3

sum = 9

9 - 6 = 3 = k

↳ Is 6 present yes

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

sum index

1, 1, 1, 3 3 2

2

Q) NEXT PERMUTATION

Given: arr[ ] = [3 1 2]

→ can be repetitive integers

→ all permut^n in sorted array

[1 2 3]
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

permutation

Brute

1. Generate all sorted
2. Linear search
3. Next Index = reqd. soln

Interview

Better (STL)

dip

T.C : (N! × N)

OPTIMAL

arr[ ] = [2, 1, 5, 4, 3, 0, 0]

↳ concept; words in dictionary

1. longest prefix match

a[i] < a[i+1]

2. find > A, but the smallest one so that you can stay close
3. Try to place rest in sorted array

Ex: • [1 2 3 4 5] dip

last index that can have dip is n-2

• [5 4 3 2 1]

No dip [∵ it is last permut^n]

Algo

last dip index

for (i = n-2 ; i >= 0 ; i--) {
ind = -1
if (a[i] < a[i+1]) {
checking dip
ind = i ;
break ;

if (ind == -1)
rev (arr)

Step 2 for (i = n-1 ; i >= ind ; i++)
if (arr[i] > arr[ind])
swap (arr[i], arr[ind])
break ;

Step 3 : rev (arr, ind+1, n-1)

CODE :

~~ind~~
int ind = -1 ;
int n = A.size() ;

for (int i = n-2 ; i >= 0 ; i--) {
if (A[i] < A[i+1]) {
ind = i ;
break ;
}
}

if (ind == -1) {
reverse (A.begin(), A.end()) ;
return A ;
}

for (int i = n-1 ; i > ind ; i--) {
if (A[i] > A(ind)) {
swap (A[i], A[ind]) ;
break ;
}
}

reverse (A.begin() + ind + 1, A.end()) ;
return A ;
}

TC : O(3N)

SC : ~~O(N)~~ → O(1)

Eg: [2 4 1 7 5 0]
0 1 2 3 4 5

index = 2 : step 1 ✓

step 2 : for (i = 5 → 3)
if (A(i) > A(ind))
swap

increase

Q) Kadane's Algo

• Maximum (SUBarray) sum → {4, -1, 5} ✗ {4, -1} ✓ {4, -1, -2} ✓

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

sum = 7

Brute

```
maxi = INT_MIN;
for (i = 0 → n) {
    for (j = i → n) {
        sum = 0
        for (k = i → j) {
            sum += arr[k]
        }
        maxi = max(sum, maxi)
    }
}
```

T.C : O(n³)
S.C : O(1)

every subarray genaration

Better

```
for (i = 0 → n)
    sum = 0
    for (j = i → n)
        sum += arr[j]
    maxi = max(      )
```

O(n²)
O(1)

OPTIMAL Kadane's Algo

maxi = INT_MIN; | -2 ✗

It won't make sense to carry -2 forward as it only decreases the sum

sum = 0 - 0

↳ if (sum < 0) sum = 0

```
int maxSubarraySum(int arr[], int n)
{
    if (sum == 0) start = i;
    int sum = 0, maxi = INT_MIN;
    for (int i → n) {
        sum += arr[i];
        if (sum > maxi) {
            maxi = sum;
            ans start = start,
            ans end = i
        }
        if (sum < 0) {
            sum = 0;
        }
    }
    return maxi
```

ans start = -1
ans end = -1

TC : O(n²)
SC : O(1)

MAJORITY ELEMENT (> N/2 times)

arr[] = [2 2 3 3 1 2 2]

n = 7 $\frac{n}{2} = 3$ 2 → 4 times

Brute

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

T.C : O(n²)
S.C = O(1)

Better (either NlogN or N)

Hashing

```
int fun (vector<int> v) {
    ordered map<int, int> mpp;
    → for (i = 0 → n) {
        mpp[v[i]]++;
    }
    for (auto it : mpp) {
        if (it.second > (n/2)) {
            return it.first;
        }
    }
    return -1;
}
```

(2, 4)	(2,4)
(3, 2)	(3,2)
(1, 1)	(1,1)

(ele, count)
↓ Key ↓ value

NlogN

TC : O(NlogN) + O(N)

SC : O(N) ↓ unique ele

unordered map O(n)

OPTIMAL (Moore's Voting algorithm)

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

ele = 7

Count = ✗ ✗ ✗ ✗ ✗ 0 [7 definitely not majority element]

Till here N = 6 ; N/2 = 3

count = 3 not > 3 that is why

∴ Count = 0

new ele = 5

Count = 0 1 0 1 2 1 0 1 2 3 4

```
int majorityelement (vector<int> v) {
    int cnt = 0;
    int el = ;
    for ( i = 0 → n ) {
        if ( cnt = = 0) {
            cnt = 1;
            el = v [i];
        }
        else if (v[i] = = el) {
            cnt ++ ;
        }
        else {
            cnt - - ;
        }
    int cnt1 = 0;
    for (i = 0 → n)
        if (v[i] = = el ) cnt1++;

    if (cnt1 > (v.size()/2) {
        return el;
    }
    return -1 ;
```

TC: O(N)
SC: O(1)

Q) STOCKS

(1)

arr[] = [7, 1, 5, 3, 6, 4] → Price of Stock on each day

n = 6 //no. of days.

Buy → 1

Sell → 6

profit → 5

// If you are selling on ith day.
you buy on the min price from
1st (i-1)

```
int mini = arr[0];
int maxprofit = 0;
int n = arr.size();
for (i = 0 → n)
    int cost = arr[i] - mini;
    maxprofit = max(maxpro, cost)
    mini = min(mini, arr[i])
return maxprofit;
```

Q) Sort an array of 0's, 1's 2's

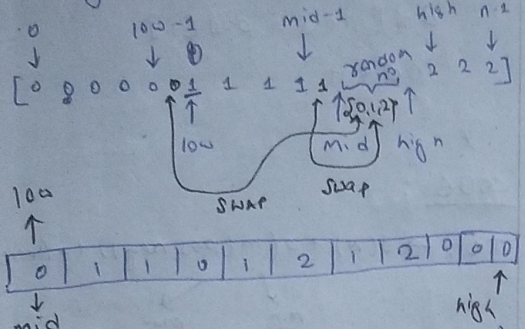
Dutch National Flag algorithm
↳ OPTIMAL

low, mid, high

[0, ..., low-1] → 0 extreme left

[low, ..., mid-1] → 1

[high+1, n-1] → 2 extreme right



0	1	1	0	1	2	1	2	0	0	0

```
void Sort Array (vector<int>& arr, int n)
{
    int low=0, mid =0, 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--;
        }
    }
}
```

TC : O(N)
SC : O(1)

BRUTE

```
int count0 = 0;
int count1 = 0;
int count2 = 0;
int n = arr.size();
for (int i = 0 → n) {
    if (arr[i] == 0) {
        count0 ++;
    }
    else if (arr[i] == 1) {
        count1 ++;
    }
    else {
        count2 ++;
    }
}
for (int i = 0 → (count0) - 1 ) {
    arr [i] = 0;
}
for (int i = count0 → (count1) - 1){
    arr [i] = 1;
}
for (int i = count1 → n-1 ) {
    arr [i] = 2;
}
```

TC : O(NlogN)
SC : O(N)

Q) Rearrange array elements by sign

→ n always even

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

BRUTE

TC : O(N) + O(N/2)
SC : O(N)

pos [] = [3 1 2]

Neg [] = [-2 -5 -4]

```
for (i = 0 → n/2)
    arr[i * 2] = pos[i]
    arr[(i * 2) + 1] = Neg[i]
```

OPTIMAL [case 1 : pos = neg]

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

ans[] = {

0 1 2 3 4 5

↑ pos ↑ neg

}

```
vector<int> rearrangeArray (vector<int>
& nums) {
  int n = nums.size();
  int pos_index=0 , neg_index = 1;
  vector <int> ans (n, 0);
  for (int i =0 → n) {
    if (nums[i] < 0) {
      ans [~~pos~~ neg_index ] = nums[i]
      neg_index + = 2;
    }
    else {
      ans [pos_index] = nums[i];
      pos_index + = 2;
    }
  }
  return ans;
}
```

// Here no. of '+ve' and '-ve' elements were same

② Not Same (case 2) [Pos != Neg]

pos > Neg ✓ Neg > pos

arr[] = { 1 2 -4 -5 3 6]

ans[] = { 1 -4 2 -5 3 6}

BACK TO BRUTE

1 3 5 6

pos[] → [~~2 3 4 1~~]

2 4

neg[] → [-4 -5]

pos > neg

First 4 elements in order
(2 pos ; 2 neg)

then put rest 2 pos

```
for ( i=0 ; i<2 ; i++) {
  arr[ i*2 ] = pos[i]
  arr[ i*2 + 1 ] = neg[i]
}
ind = 4
for ( i = 2 ; i < pos.size(), i++)
  arr[ind] = pos[i]
  ind + +;
}
```

TC : O(2N)
SC : O(N)

CODE :

TC : → O(N) + O(m(pos, neg)) + O(left)

min: O(0) + O(N) ; max: O(N/2) + O(0)

= O(N) + O(N) = O(2N)

→

```
vector<int> pos, neg;
int n = a.size();
for ( i = 0 → n) {
  ~~pos~~ if (a[i] > 0) {
    pos.pushback
  }
  else
    neg.pushback
}
if (pos.size > neg.size) {
  for ( i = 0 → neg.size)
    a[ 2*i ] = pos[i]
    a[ 2*i+1 ] = neg[i]
}
int index = neg.size*2
for (int i = neg.size → pos.size)
  a[index] = pos[i]
  index++;
}
else { pos size < neg size }
```

Q) LEADERS IN AN ARRAY

→ everything on the right should be smaller

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

6 is bigger

ans → [22, 12, 6]

same

same

To right of 22 every number is smaller than 22

BRUTE

```
for (i = 0 → n)
    leader = true
    for (j = i+1 → n)
        if (a[j] > a[i]
            leader = false
            break
    if (leader == true)
        ans.add(a[i]);
```

TC ≈ O(N²)

SC : O(N)

worst case

OPTIMAL

Start iterating from the back

arr[] = [10 22 12 3 0 6]

| 6 | 12 | 22 |

~~if num > max~~

maxi = ~~INT_MIN~~ : 6

0 > 6 ✗ 0 max ✗ max = 6

3 > 6 ✗ 3 max ✗ max = 6

12 > 6 ✓ max = 12

22 > 12 ✓ max = 22

10 > 22 ✗ max = 22

Code :

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

TC : O(N) → Avg, O(NlogN) → worst

Q) Longest Consecutive Seq.

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

BRUTE :

```
largest = 1
for (i = 0 → n)
    x = arr[i]
    count = 1
    for (j = 0 → n) {
        if (arr[j] == x) {
            x = x + 1;
            count++;
        }
    }
```

TC : O(N²)

SC : O(1)

BETTER :

```
sort(arr.begin(), arr.end())
largest = 1, cntcur = 0,
left small = INT_MIN
for (i = 0 → n) {
    if (arr[i] - 1 == leftsmaller)
        cnt++;
        left smaller = arr[i]
    else if (arr[i] != leftsmaller)
        cnt = 1
        left.smaller = arr[i]
    longest = max(longest, cnt)
```

TC : nlogn + n

SC : O(1)

OPTIMAL

```
int longest = 1;
unordered_set<int> st;
for (i = 0 → n) {
    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++;
        }
        longest = max(longest, cnt)
    }
}
return longest
```

unordered set

102
4
100
1
101
3

TC : O(1) ↔ O(N)

SC :

Q) MAXM PRODUCT SUBARRAY

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

ans = 6

BRUTE: (3 loops)

```
for (i = 0 → n) {
    for (j = i → n) {
        for (k = i → j) {
            product *= arr[k]
            ~~maxi~~
        }
        maxi = max(maxi, product)
    }
}
```

TC ≈ O(N³)
SC: O(1)

BETTER (2 loops)

```
for (i = 0 → n) {
    prod = 1
    for (j = i → n)
        prod *= arr[j]
        maxi = max(maxi, prod)
    }
}
```

TC ≈ O(N²)
SC: O(1)

OPTIMAL (2 soln) → observn (✓) → Kadane's (✗)

Observn:

1. all +ve ele : Multiply all ✓
2. even -ve, +ve : Multiply all ✓
3. odd -ve, +ve :

→ [. . . . .]
prefix suffix

4. it has zeros : '0' breaking point. arrays breaks into → sub-arrays. Maxm is the answer.

CODE:

```
max = INT_MIN
pre = 1, suff = 1
for (i = 0 → n)
    if (pre == 0)  pre = 1;
    if (suff == 0)  suff = 1;
    prefix = prefix × arr[i]
    suff = suff × arr[n-i-1]
    maxi = max(maxi, max(pre, suff))
```

Q) Maxm circular subarray sum

arr = [8 -8 9 -9 10 -11 12]

BRUTE:

```
int res = arr[0]
for (i = 0 → n)
    cursum = 0
    for (j = 0 → n)
        ind = (i+j) % n
        cursum += arr[ind]
        res = max(res, cursum)
return res
```

TC: ≈ O(N²)
SC: O(1)

CIRCULAR INDEX

BETTER

(prefix & suffix)

```
suffixsum = arr[n-1]
maxsuffix = (n, 0)
maxsuffix[n-1] = arr[n-1]
for (i = n-2, i >= 0)
    suffixsum += arr[i]
    maxsuffix[i] = max(maxsuffix[i+1], suffixsum)

circularsum = arr[0]
normalsum = arr[0]
cursum = 0
prefix = 0
for (i = 0 → n)
    cursum = max(cursum + arr[i], arr[i])
    normalsum = max(normalsum, cursum)
    prefix += arr[i]
    circularsum = max(circularsum, prefix + maxsuffix[i+1])

return max(circularsum, normalsum)
```