Topic

3.1

Q- largest element in array

Brute ⇝ sort array and return $(n-1)^{th}$ index T.C ⇝ O(nlogn)

Optimal ⇝

```
largest = a[0]
for(i=0;i<n;i++){
    if(arr[i] > largest){
        largest = a[i];
    }
    print(largest)
}
```

T.C ⇝ O(n)

Q- Second largest element in array

Brute ⇝ Sort array and return index before $(n-1)^{th}$

their maybe a case like ⇝ {1,2,4,5,7,7}

```
for(i=n-2; i>=0; i--){
    if(arr[i] != largest){
        Sec_lar = arr[i];
        break;
    }
}
```

Sorting → finding 2nd largest

T.C ⇝ O(nlogn + n)

Better →

```
largest = arr[0]
for(i=0;i<n;i++){
  if(arr[i] > largest){
    largest = arr[i]
  }
}
```

T.C → O(n)

```
sec_lar = -1
for(i=0;i<n;i++){
  if(arr[i] > sec_lar && arr[i] != largest){
    Sec_lar = arr[i]
  }
}
print(sec_lar
```

T.C → O(n+n) = O(2n)

Optimal

```
largest = arr[0] , sec_lar = -1
for(i=0;i<n;i++){
  if(a[i] > largest){
    sec_lar = largest;
    largest = a[i];
  }
  else if (a[i] < largest && a[i] > sec_lar){
    sec_lar = a[i];
  }
}
return sec_lar;
```

(assuming array doesn't have any -ve value. but if it had take any min. integer)

} don't change order of writing

T.C → O(N)

Page No.

Q- Remove duplicates from sorted array

Brute:

```
set<int> st
for(i=0;i<n;i++){
    st.insert(arr[i])
}
index=0
for(auto it : st){
    arr[index] = it;
    index++;
}
```

first we insert element into set set doesn't have any duplicates then inserting the set elements into array

T.C ~> O(NlogN) + O(N)

T.C ~> O(N + NlogN)

S.C ~> O(N)

Optimal 2 pointer approach

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

1st element of array is always unique. keep 1 pointer on 1st index

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

T.C ~> O(N)

S.C ~> O(1)

Q- Left rotate an array by one place

arr[] = {1,2,3,4,5}
↳ output {2,3,4,5,1}

Optimal:

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

T.C ~> O(n)
S.C ~> O(1) ← extra space
space ~> O(N)

Q- Left rotate an array by D places

Brute:

d = d % n
temp[] = {1,2,3}

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

T.C ~> O(d) + O(n-d) + O(d)
T.C = O(n+d)
S.C = O(d)

To decrease extra space)

3 2 1 7 6 5 4
1 2 3 4 5 6 7

Complete reverse = 4 5 6 7 1 2 3

Optimal:

```
void Reverse(int arr[], int start, int end){
    while(start <= end){
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}

void leftRotate(int arr[], int n, int d){
    Reverse(arr, arr+d);
    Reverse(arr+d, arr+n);
    Reverse(arr, arr+n);
}
```

T.C ~> O(2n)
S.C ~> O(1)

Q- Move all zeroes to the end of the array

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

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

Brute

temp[ ] = { }

```
for(i=0;i<n;i++){
  if(arr[i]!=0){
    temp[i]=arr[i];
  }
```

~~size_array = temp.size~~

```
for(i=0; i<temp.size(); i++){
    arr[i] = temp[i];
}

size_array = temp.size();

for(i=size_array; i<n; i++){
    arr[i] = 0;
}
```

T.C = O(N) + O(X) + O(N-X)

= O(2N)

S.C = O(N)

2 pointer approach

first 0 find kro and usse
j ko rkhdo and i ko usse
ek aage.

ab jese jese non-zero number
mile toh uske swap kro

```
j=-1
for(i=0;i<n;i++){
  if(arr[i]==0){        T.C ~> O(N)
    j=i
    break;
  }
}
for(i=j+1;i<n;i++){
  if(arr[i]!=0){
    swap(arr[i], arr[j]);
    j++;
  }
}
```

T.C ~> O(n)

S.C ~> O(1)

~> T.C = O(n-x)

Q- Union of two sorted arrays

arr1[] = {1,1,2,3,4,5}
arr2[] = {2,3,4,4,5}

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

Brute: Store it in set then place in an array called union

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

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

union.add(it)
```

T.C → O(n1 log n + n2 log n + O(n1+n2))
S.C → O(n1 + n2)

Optimal: arr1[] = {1,1,2,3,4,5}
arr2[] = {2,3,4,4,5,6}

2 pointer approach

```
let arr1 = a
let arr2 = b

Union[] = { }
n1 = a.size(), n2 = b.size();
while(i<n1 && j<n2){
    if(a[i] <= b[j]){
        if (union.size()==0 || union.back() != a[i]){
            union.push_back(a[i]);
        }
        i++;
    }
    else
```

Its not complete soln
watch striver soln

Q- A[] = {1,2,2,3,3,4,5,6}
B[] = {2,3,3,5,6,6,7}

find intersection of two array

Brute:

we will take a visited array and update it accordingly

```
vis[] = {0};
inter = { }
for(i → n1){
    for(j → n2){
        if(a[i] == b[j] && vis[j] == 0){
            inter.add(a[i])
            vis[j] = 1;
            break;
        }
        if(b[j] > a[i]) break;
    }
}
```

T.C ⇝ O(n1 × n2)
S.C ⇝ O(n2)

(n1 + n2)

Optimal

A[] = {1,2,2,3,3,4,5,6}
B[] = {2,3,3,5,6,6,7}

```
int i=0, j=0
vector<int> ans;
while(i<n1 && j<n2){
    if(A[i] < B[j]){
        i++;
    }
    else if(B[j] < A[i]){
        j++;
    }
    else{
        ans.push_back(A[i]);
        i++;
        j++;
    }
}
return ans;
```

T.C ⇝ O(n1 + n2)
S.C ⇝ O(1)

Q- Find the Missing number blw 1 to N in array
arr[] = {1,2,4,5} N=5 output=3

Brute:

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

T.C ~> O(N²)
S.C ~> O(1)

Better

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

T.C ~> O(2N)
S.C ~> O(N)

optimal:

** MY OWN Solution

```
int comb[100];
int k=0;
for(1->5){
    comb[k]=i;
    k++;
}
int i=0,j=0
while(i<n && j<5){
    if(arr[i]==comb[j]){
        i++;
        j++;
    }else{
        cout<<comb[j]<<" ";
    }
}
```

T.C ~> O(N)
S.C ~> O(1)

optimal

Optimal:

Sum = $\frac{n(n+1)}{2}$

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

T.C → O(N)
S.C → O(1)

Q- Find the Maximum consecutive ones

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

We will take a counter and whenever find 1, we increase it and whenever find 0, we change it value to 0

We also store the max of counter

cnt = 0 1 2
0 1 2 3
0 1 2

Max = 0 1 2 3

Max = 3

```
n = arr.size();
int maxi = 0;
int cnt = 0;

for (0 → n){
    if (Arr[i] == 1){
        cnt++;
        maxi = max(maxi, cnt);
    }
    else{
        cnt = 0
    }
}
cout << maxi
```