**Sheet :-** https://takeuforward.org/strivers-a2z-dsa-course/strivers-a2z-dsa-course-sheet-2/

**ARRAYS - EASY**

1. **Largest Element in Array**

**Approach 1** :- Sort array ascending, print last ele. TC:- O(N*log(N)) SC: O(n)

**Approach 2** :- Use max variable TC:- O(n)  SC:- O(1)

```
class Compute {

    public int largest(int arr[], int n)
    {
        int max=arr[0];
        for(int i=0;i<n;i++){
            if(arr[i]>max){
                max=arr[i];
            }
        }
        return max;
    }
}
```

2. **Second Largest Element in Array**

   **Approach 1** :- Sort it, print second last TC:- O(NlogN) SC:- O(1)

   **Approach 2** :- Find max ele in one traversal, then find the number just smaller in next traversal  TC :- O(2N) SC:- O(1)

   Approach 3 :- Traverse array, If the current element is larger than ‘large’ then update second_large and large variables Else if the current element is larger than ‘second_large’ then we update the variable second_large. Once we traverse the entire array, we would find the second largest element in the variable second_large.

```
class Solution {
    int print2largest(int arr[], int n) {
        // code here
        int max=-1;
        int secmax=-1;
        for(int i=0;i<n;i++){
            if(arr[i]>max){
                secmax=max;
                max=arr[i];
            }
            else if(arr[i]>secmax && arr[i]!=max){
                secmax=arr[i];
            }
        }
        return secmax;
    }
}
```

   Tc :- O(N) Sc:- O(1)

3. **Check if array is sorted**

   **Approach 1 :-** Take elements one by one then traverse the array completely in front of it.  TC :- O(n2)

   **Approach 2 :-** Check if previous element is smaller than previous one, if false then it is not sorted. TC :- O(N) SC:- O(1)

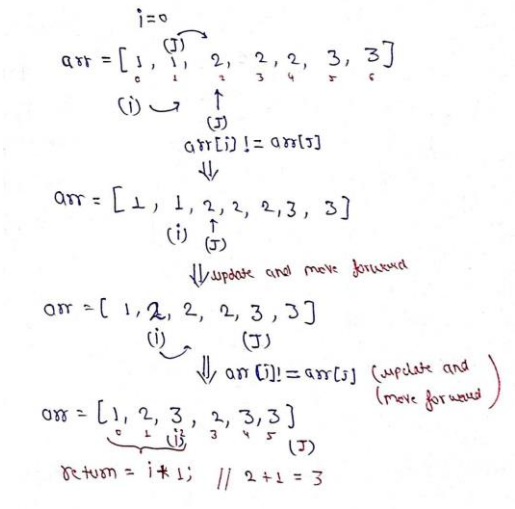
```
static boolean isSorted(int arr[], int n) {
  for (int i = 1; i < n; i++) {
    if (arr[i] < arr[i - 1])
      return false;
  }

  return true;
}
```

4. **Remove duplicated in place from sorted array**

**Approach 1** :- Declare Hashset, put array elements in set, put all elements of set to a new array TC:-O(NlogN)+O(n) SC:-O(n)

```
static int removeDuplicates(int[] arr) {
    HashSet < Integer > set = new HashSet < > ()
    for (int i = 0; i < arr.length; i++) {
        set.add(arr[i]);
    }
    int k = set.size();
    int j = 0;
    for (int x: set) {
        arr[j++] = x;
    }
    return k;
}
```

**Approach 2** :- Two Pointers – have two pointers i and j . check if arr[i]!=arr[j] then arr[i] will be arr[j], basically iterate over same elements using j without doing anything , when find unique elements swap with element at i. TC :- O(N) SC:- O(1)




```
class Solution {
    public int removeDuplicates(int[] arr) {
        int j=0;
        for(int i=1;i<arr.length;i++){
           if(arr[i]!=arr[j]){
            j++;
            arr[j]=arr[i];
           }
        }
        return j+1;
    }
}
```

5. **Right Rotate Array by k places (includes 1 place)**

**Approach 1** :- Copy k elements to temporary array, shift elements to empty positions now. Put k elements in the front again from the temp array. TC :- O(n) SC:- O(k)

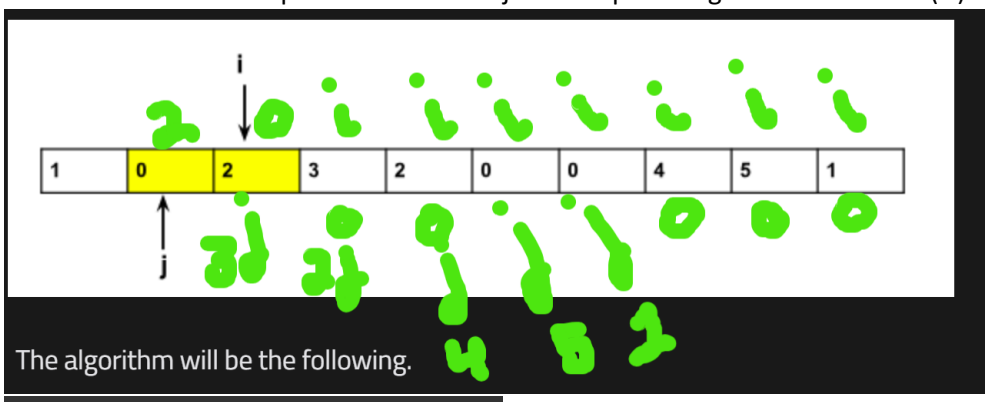
**Approach 2** :- Reverse last k ele and (n-k) ele, reverse whole array now TC:- O(N) SC:- O(1)

```java
class Solution {
    public void rotate(int[] nums, int k) {
        k = k%nums.length;
        reverseArray(nums, 0, nums.length-1);
        reverseArray(nums, 0, k-1);
        reverseArray(nums, k, nums.length-1);
    }

    public void reverseArray(int[] a, int l, int r){
        while(l <= r){
            int t = a[l];
            a[l] = a[r];
            a[r] = t;
            l++; r--;
        }
    }
}
```

6. **Move all zeros to the end of the array**

   **Approach 1** :- Make temp array, copy all non-zero elements to it then fill the rest with zeroes. Tc :- O(2N) SC:-O(N)

   **Approach 2** :- Use 2 pointers , move i when encounter non-zero element, move j after swap. In case of non zero swap element at i and j else keep moving i forward. TC :- O(N) SC:- O(1)




```java
class Solution {
    public void moveZeroes(int[] nums) {
        int i = 0;
        int j = 0;

        while (j < nums.length) {
            if (nums[j] != 0) {
                int temp = nums[j];
                nums[j] = nums[i];
                nums[i] = temp;
                i++;
            }
            j++;
        }
    }
}
```

7. **Linear Search**

   **Approach 1:** Search element in array, return the index. TC:- O(n) SC:- O(1)

```java
class Solution{

    static int search(int arr[], int N, int X)
    {

        // Your code here
        for(int i=0;i<N;i++){
            if(arr[i]==X){
                return i;
            }
        }
        return -1;
    }

}
```

8. **Union of two arrays**

**Approach 1** :- Use Hashset, put all elements in it to find distinct elements. TC:- O(n+m(log(n+m)) SC:- O(n+m)

```java
class Solution{
    public static int doUnion(int a[], int n, int b[], int m)
    {
        //Your code here
        HashSet<Integer> hs= new HashSet<>();
        for(int i=0;i<n;i++){
            hs.add(a[i]);
        }
        for(int i=0;i<m;i++){
            hs.add(b[i]);
        }

        return hs.size();
    }
}
```

**In case that the two given arrays are sorted**

**Approach 1 :-** Two Pointers approach, create a union arraylist, move one pointer in each array in case :- if a[i]==b[j] add a[i], if a[i]<b[j] add a[i] first then i++, ], if b[j]<a[i] add b[j] first then j++. TC:-O(n+m) SC:-O(n+m)

```
{
    public static ArrayList<Integer> findUnion(int arr1[], int arr2[], int n, int m)
    {   int i=0,j=0;
        ArrayList<Integer> Union= new ArrayList<>();
        while(i<n && j<m){
            if(arr1[i]<=arr2[j]){
                if (Union.size() == 0 || Union.get(Union.size()-1) != arr1[i])
                    Union.add(arr1[i]);
                i++;}
            else{
                if (Union.size() == 0 || Union.get(Union.size()-1) != arr2[j])
                  Union.add(arr2[j]);
                j++;}
        }
        while (i<n)
            {
                if (Union.get(Union.size()-1) != arr1[i])
                    Union.add(arr1[i]);
                i++;}
        while (j<m)
            {
                if (Union.get(Union.size()-1) != arr2[j])
                    Union.add(arr2[j]);
                j++;}
        return Union;
        }

    }
```

9. **Find the missing number in an array**

**Approach 1 :-** Do linear search for each ele in [0..n]. any ele not found is ans. TC:-O(N2) SC:-O(1)

**Approach 2:-** Put ele [0…n] in hash array then count frequency, the one with zero freq is ans. TC:- O(2N) SC:-O(N)

**Approach 3:-** Calculate the sum of natural nos [0..n] using formula, calulate summation of array, subtract and find which one is missing TC:- O(N) SC:-O(1)

```
class Solution {
    public int missingNumber(int[] nums) {
        int n=nums.length;
        int sum=(n*(n+1))/2;
        int sumarr=0;
        for(int i=0;i<n;i++){
              sumarr+=nums[i];
        }
        return sum-sumarr;
    }
}
```

**Approach 4:- XOR**

```
XOR of two same numbers is always 0 i.e. a ^ a = 0. ←Property 1.
XOR of a number with 0 will result in the number itself i.e. 0 ^ a = a. ←Property 2
```

```
Assume the given array is: {1, 2, 4, 5} and N = 5.
XOR of (1 to 5) i.e. xor1 = (1^2^3^4^5)
XOR of array elements i.e. xor2 = (1^2^4^5)
XOR of xor1 and xor2 = (1^2^3^4^5) ^ (1^2^4^5)
            = (1^1)^(2^2)^(3)^(4^4)^(5^5)
            = 0^0^3^0^0 = 0^3 = 3.
The missing number is 3.
```

```java
public static int missingNumber(int []a, int N) {

    int xor1 = 0, xor2 = 0;

    for (int i = 0; i < N - 1; i++) {
        xor2 = xor2 ^ a[i]; // XOR of array elements
        xor1 = xor1 ^ (i + 1); //XOR up to [1...N-1]
    }
    xor1 = xor1 ^ N; //XOR up to [1...N]

    return (xor1 ^ xor2); // the missing number
}
```

10. **Find maximum consecutive ones**

    **Approach :-** keep count variable keep doing count++ till we find 1 and update maxcount with every iteration of counting ones when hit 0, reset count to 0. TC:- O(N) SC:-O(1)

```java
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int ct=0;
        int max=0;
        for(int i=0;i<nums.length;i++){
            if(nums[i]==1){
                ct++;
                max=Math.max(max,ct);
            }
            else{
                ct=0;
            }
        }
        return max;

    }
}
```

11. **Subarray with Sum K**

    **Approach 1** :- Generate all subarrays then check how many have the desired sum K.

**Approach 2** :- Save Prefix Sums in Hash Map. Check if the required prefix sum is available to make the desired sum. If yes increase the count (same can be for length if yes then check if it's the maximum length). Put the updated sum in map.
TC :- O(nlogn) Sc:- O(n)

```java
class Solution {
    public int subarraySum(int[] nums, int k) {
        Map<Integer, Integer> mp = new HashMap<>();
        int currsum=0;
        int ct=0;
        mp.put(0, 1);
        for(int i=0;i<nums.length;i++){
            currsum+=nums[i];
            int rem= currsum-k;
            if (mp.containsKey(rem)) {
                ct += mp.get(rem);
            }
            mp.put(currsum, mp.getOrDefault(currsum, 0) + 1);

        }
        return ct;
    }
}
```