**Max Consecutive Ones**

class Solution {

public int findMaxConsecutiveOnes(int[] nums) {

int count=0;

int ans=0;

for(int i:nums){

if(i==1)

count++;

else{

if(count>ans){

ans=count;

}

count=0;

}

}

if(count>ans)

ans=count;

return ans;

}

}

**Find Numbers with Even Number of Digits**

class Solution {

public int findNumbers(int[] nums) {

int answer=0;

int check =0;

for(int i=0;i<nums.length;i++){

check=containEven(nums[i]);

if(check%2==0){

answer++;

}

}

return answer;

}

public int containEven(int num){

int val=1;

int a=num;

while(a/10!=0){

a=a/10;

val++;

}

return val;

}

}

**Squares of a Sorted Array**

class Solution {

public int[] sortedSquares(int[] A) {

for (int i=0;i<A.length;i++){

A[i]=A[i]\*A[i];

}

Arrays.sort(A);

return A;

}

}

**Duplicate Zeros**

class Solution {

public void duplicateZeros(int[] arr) {

for(int i=0;i<arr.length-1;i++){

if(arr[i]==0){

shiftRight(arr,i);

arr[i+1]=arr[i];

++i;

}

}

}

public void shiftRight(int[] arr,int index){

for(int i=arr.length-2;i>index;i--){

arr[i+1]=arr[i];

}

}

}

**Merge Sorted Array**

class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

boolean shift=false;

if(m==0){

for(int i=0;i<n;i++)

nums1[i]=nums2[i];

}

else{

for(int i=0;i<n;i++){

int n2=nums2[i];

shift=false;

int j=0;

for(j=0;j<m;j++){

if(n2<=nums1[j]){

m= shiftRight(nums1,j,m);

shift=true;

break;

}

}

if(shift==true)

nums1[j]=n2;

else {

nums1[j]=n2;

m++;

}

System.out.println(Arrays.toString(nums1));

}

}

}

public int shiftRight(int[] nums1, int index,int m){

for(int i=m-1;i>=index;i--){

nums1[i+1]=nums1[i];

}

m++;

return m;

}

}

**Remove Element**

class Solution {

public int removeElement(int[] nums, int val) {

int ans=nums.length;

int length=nums.length;

for(int i=0;i<length;i++){

if(nums[i]==val){

shiftLeft(nums,i);

length--;

--ans;

i--;

}

}

return ans;

}

public void shiftLeft(int[] nums,int index){

int temp=nums[index];

int i;

for(i=index;i<nums.length-1;i++){

nums[i]=nums[i+1];

}

nums[i]=temp;

}

}

**Remove Duplicates from Sorted Array**

class Solution {

public int removeDuplicates(int[] nums) {

int j=1;

int length=0;

if(nums.length>=1){

length=1;

int value=nums[0];

for(int i=1;i<nums.length;i++){

if(nums[i]!=value){

value=nums[i];

nums[j]=nums[i];

j++;

length++;

}

}

}

return length;

}

}

**Check If N and Its Double Exist**

import java.lang.Math;

class Solution {

public boolean checkIfExist(int[] arr) {

HashSet<Integer> map=new HashSet<Integer>();

boolean answer=false;

int t,d;

for(int i:arr){

if(i%2==0){

t=2\*i;

d=(i/2);

}

else{

t=2\*i;

d=2\*i;

}

System.out.println(t+" "+d);

if(map.contains(t)==true || map.contains(d)==true){

answer=true;

break;

}

else

map.add(i);

}

return answer;

}

}

**Valid Mountain Array**

class Solution {

public boolean validMountainArray(int[] A) {

int i=0;

int j=0;

boolean increase=false;

boolean decrease=false;

int max=0;

int max\_index=0;

for(int k=0;k<A.length;k++){

if(max<A[k]){

max=A[k];

max\_index=k;

}

}

while(i<max\_index){

if(A[i]<A[i+1]){

increase=true;

}

else{

increase=false;

break;

}

i++;

}

j=max\_index;

while(j<A.length-1){

if(A[j]>A[j+1]){

decrease=true;

}

else{

decrease=false;

break;

}

j++;

}

if(increase==true && decrease==true)

return true;

else

return false;

}

}

**Replace Elements with Greatest Element on Right Side**

class Solution {

public int[] replaceElements(int[] arr) {

int i=0;

int j=1;

int max=0;

while(i<arr.length-1){

if(j<arr.length){

if(arr[j]>max){

max=arr[j];

}

}

else{

arr[i]=max;

i++;

j=i;

max=0;

}

j++;

}

arr[arr.length-1]=-1;

return arr;

}

}

**Remove Duplicates from Sorted Array**

class Solution {

public int removeDuplicates(int[] nums) {

int j=1;

int length=0;

if(nums.length>=1){

length=1;

int value=nums[0];

for(int i=1;i<nums.length;i++){

if(nums[i]!=value){

value=nums[i];

nums[j]=nums[i];

j++;

length++;

}

}

}

return length;

}

}

**Move Zeroes**

class Solution {

public void moveZeroes(int[] nums) {

int zero=0;

boolean is\_zero=false;

int i=0;

int n=nums.length;

while(i<n){

if(nums[i]==0){

if(is\_zero==false){

zero=i;

is\_zero=true;

}

}

else if(is\_zero==true){

int temp=nums[zero];

nums[zero]=nums[i];

nums[i]=temp;

is\_zero=false;

i=zero;

}

i++;

}

}

}

**Sort Array By Parity**

class Solution {

public int[] sortArrayByParity(int[] A) {

int i=0;

int even=0;

int odd=0;

boolean is\_odd=false;

while(i<A.length){

if(A[i]%2==0){

even=i;

}

else{

if(is\_odd==false){

odd=i;

is\_odd=true;

}

}

if(odd<even && is\_odd==true){

int temp=A[even];

A[even]=A[odd];

A[odd]=temp;

is\_odd=false;

i=odd;

odd=0;even=0;

}

i++;

}

return A;

}

}