***Maximum Sum SubArray***

public int maxSubArray(int[] nums) {

int currentSum =0;

int maxSum =Integer.MIN\_VALUE;

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

currentSum = currentSum + nums[i];

maxSum = Math.max(currentSum,maxSum);

if(currentSum<0){

currentSum =0;

}

}

return maxSum;

}

***Sort Colors 0,1,2***

public void sortColors(int[] nums) {

int left =0; //slow pointer

int i=0; //fast pointer

int right = nums.length -1; //pointer to shift to last

while (i<=right){

switch(nums[i]){

case 0:

int temp = nums[i];

nums[i] = nums[left];

nums[left]=temp;

i++;

left++;

break;

case 1:

i++;

break;

case 2 :

temp = nums[i];

nums[i] =nums[right];

nums[right] =temp;

right --;

break;

}

}

}

***Find Minimum in Rotated Sorted Array* Input: *nums = [3,4,5,1,2] Output: 1***

public int findMin(int[] nums) {

if(nums.length==0) return -1;

if(nums.length==1) return nums[0];

int l =0;

int r = nums.length -1;

while (l<r){

int mid = l + (r-l)/2; //jis point pe not sorted hai should be the answer

if(mid>0 && nums[mid] < nums[mid-1]){ //mid >0 to handle mid-1 : index out of bound exception

return nums[mid];

}

else if (nums[l]<=nums[mid]&& nums[mid]>nums[r] ){ //means that left part of array is sorted and you have to go to unsorted part to get miniumu element

l = mid +1;

}

else{

r =mid-1;

}

}

return nums[l] ;

}

}

//4,5,6,7,0,1,2

[***Best Time to Buy and Sell Stock***](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)

public int maxProfit(int[] prices) {

int maxProfit =0;

int buy = prices[0];

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

{

if(prices[i]<buy){

buy =prices[i];

}

else{

maxProfit = Math.max(prices[i]-buy, maxProfit);

}

}

return maxProfit;

}

**Move Zeroes**

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements. Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]

public void moveZeroes(int[] nums) {

int j=0;

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

if (nums[i]!=0){

int temp = nums[i];

nums[i] = nums[j];

nums[j]=temp;

j++;

}

}

}

[Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/)

int[] result = new int[nums.length];

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

int product =1;

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

product \*=nums[j];

}

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

product \*=nums[j];

}

result[i] = product;

}

return result;

}

public int[] productExceptSelf(int[] nums) {

int n = nums.length;

int[] output = new int[n];

int [] prefix = new int[n];

int [] suffix = new int[n];

//pre compute prexif

//Prefix[i] = product of all elements to left of i

prefix[0]=1;

for(int i=1;i<n;i++){

prefix[i] = prefix[i-1] \* nums[i-1];

}

//pre compute suffix

//Suffix[i] = product of all elements to right of i

suffix[n-1] =1;

for (int i=n-2;i>=0;i--)

{

suffix[i] =suffix[i+1] \* nums[i+1];

}

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

output[i] = prefix[i]\* suffix[i];

}

return output;

}

#### [Majority Element](https://leetcode.com/problems/majority-element/)

public int majorityElement(int[] nums) {

int result=0;

Map<Integer,Integer> map = new HashMap <>();

for(int i:nums){

if(map.containsKey(i)){

map.put (i,map.get(i)+1);

}

else{

map.put(i,1);

}

}

for (Map.Entry<Integer,Integer> hm : map.entrySet()) {

if (hm.getValue() >(nums.length/2))

result = hm.getKey();

}

return result;

}

***Check if Array Is Sorted and Rotated Input***: nums = [3,4,5,1,2] Output: true

public boolean check(int[] nums) {

int count = 0;

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

if (nums[i - 1] > nums[i]) {

count++;

}

}

if (nums[nums.length - 1] > nums[0]) {

count++;

}

return count <= 1;

}

[***Find Peak Element***](https://leetcode.com/problems/find-peak-element/) (A peak element is an element that is strictly greater than its neighbours) **Input:** nums = [1,2,1,3,5,6,4] **Output:** 5

public int findPeakElement(int[] nums) {

int l =0;

int h =nums.length-1;

while(l<h){

int mid =l+(h-l)/2;

if(nums[mid]>nums[mid+1]){

h =mid;

}

else{

l =mid+1;

}

}

return l;

}

**Rotate Array right by k steps**

public void rotate(int[] nums, int k) {

int n= nums.length;

int temp[] = new int[nums.length];

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

temp[(i+k)%n] = nums[i];

}

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

nums[i]= temp[i];

}

}

public void rotate(int[] nums, int k) {

k = k%nums.length;

reverse(nums,0, nums.length-1);

reverse(nums,0,k-1);

reverse(nums,k, nums.length-1);

}

public void reverse(int nums[],int start ,int end){

while(start<end){

int temp = nums[start];

nums[start]=nums[end];

nums[end]=temp;

start++;

end--;

}

}

#### [Container With Most Water](https://leetcode.com/problems/container-with-most-water/)

#### Bar chart Description automatically generated with low confidence

#### public int maxArea(int[] height) {

int maxarea =0;

int left = 0;

int right = height.length-1;

while(left<right){

int w = right -left;

int h = Math.min(height[right],height[left]);

maxarea = Math.max(maxarea,w\*h);

if(height[left]<height[right]) {

left++;

}

else{

right--;

}

}

return maxarea;

}

#### [Two Sum II - Input Array Is Sorted](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/)

public int[] twoSum(int[] numbers, int target) {

int left =0;

int right = numbers.length -1;

while(left<right){

if(numbers[left]+ numbers[right]>target){

right--;

}

if(numbers[left]+ numbers[right]<target){

left++;

}

if(numbers[left]+ numbers[right]==target){

left++; right++;

break;

}

}

return new int[]{left,right};

}

#### [3Sum](https://leetcode.com/problems/3sum/)

public List<List<Integer>> threeSum(int[] nums) {

Arrays.sort(nums);

List<List<Integer>> output = new LinkedList();

for(int i=0;i<nums.length-2;i++){

if(i==0||i>0 && nums[i]!=nums[i-1]){

int low = i+1;

int high = nums.length -1;

int sum = 0 -nums[i];

while(low<high){

if(nums[low]+ nums[high]==sum){

output.add(Arrays.asList(nums[i],nums[low],nums[high]));

while(low<high && nums[low]==nums[low+1]) low++;

while(low<high && nums[high]==nums[high-1])high--;

low++;

high--;

}

else if(nums[low]+ nums[high]>sum){

high --;

}

else{

low++;

}

}

}

}

return output;

}

**Max Sum Subarray of size K**

N = 4, K = 2 Arr = [100, 200, 300, 400] **Output:**700

static long maximumSumSubarray(int K, ArrayList<Integer> Arr,int N){

long maxSum =0;

long sum =0;

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

{

sum += Arr.get(i);

}

maxSum = sum;

for(int i=K;i<N;i++){

maxSum = maxSum + Arr.get(i)-Arr.get(i-K);

sum = Math.max(sum,maxSum);

}

return sum;

}

***Reverse Array in size of k***

Input: arr[] = [1, 2, 3, 4, 5, 6, 7, 8, 9], K = 3 Output: 3, 2, 1, 6, 5, 4, 9, 8, 7

static void reverse(int[] arr, int grpsize){

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

int left = i;

int right =Math.min(i + grpsize - 1, arr.length - 1);

int temp;

while(left<right){

temp =arr[left];

arr[left] =arr[right];

arr[right]=temp;

left+=1;

right-=1;

}

}

}

***Subarray Sum Equals K***

public int subarraySum(int[] nums, int k) {

int count = 0, sum = 0;

HashMap < Integer, Integer > map = new HashMap < > ();

map.put(0, 1);

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

sum += nums[i];

if (map.containsKey(sum - k))

count += map.get(sum - k);

map.put(sum, map.getOrDefault(sum, 0) + 1);

}

return count;

}

public int subarraySum(int[] nums, int k) {

int count = 0, sum = 0;

HashMap < Integer, Integer > map = new HashMap < > ();

map.put(0, 1);

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

sum += nums[i];

if (map.containsKey(sum - k))

count += map.get(sum - k);

map.put(sum, map.getOrDefault(sum, 0) + 1);

}

return count;

}

***Java Program to Find Common Elements Between Two Arrays***

int arr1[]={1,2,3,4,4,5,5} int arr2[]={6,6,7,7,8,1,2,5} output: {1,2,4,5,6,7}

**Approach 1 :**

private static void FindCommonElemet(String[] arr1,

String[] arr2)

{

Set<String> set = new HashSet<>();

for (int i = 0; i < arr1.length; i++) {

for (int j = 0; j < arr2.length; j++) {

if (arr1[i] == arr2[j]) {

// add common elements

set.add(arr1[i]);

break;

}

}

}

for (String i : set) {

System.out.print(i + " ");

}

}

**Approach 2 :**

// their common element

public static void FindCommonElements(int[] arr1,

int[] arr2)

{

// create hashsets

Set<Integer> set1 = new HashSet<>();

Set<Integer> set2 = new HashSet<>();

// Adding elements from array1

for (int i : arr1) {

set1.add(i);

}

// Adding elements from array2

for (int i : arr2) {

set2.add(i);

}

// use retainAll() method to

// find common elements

set1.retainAll(set2);

System.out.println("Common elements- " + set1);

}

***Given an integer array nums, return true if the given array is monotonic, or false otherwise.***

An array is monotonic if it is either monotone increasing or monotone decreasing.

Monotonic Array [1,2,2,3] - true [1,3,2] -false

public boolean isMonotonic(int[] nums) {

boolean increasing = true;

boolean decreasing = true;

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

if(nums[i]>nums[i+1]){

increasing =false;

}

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

decreasing =false;

}

}

return increasing || decreasing ;

}

<https://leetcode.com/problems/largest-number/>