***24-02-2021***

1. **Reverse the array LC**

**void reverseString(vector<char>& s) {**

**int i=0,j=s.size()-1;**

**while(i<=j)**

**{**

**swap(s[i++],s[j--]);**

**}**

**}**

***25-02-2021***

1. **Maximum & Minimum of an array GFG**

**vector<int> getMinMax(vector<int>&arr , int n)**

**{**

**vector<int> result(2); //result array, 0-> max | 1->min**

**int i; //index**

**// If array has even number of elements**

**// then initialize the first two elements**

**// as minimum and maximum**

**if(n%2 == 0)**

**{**

**if(arr[0] > arr[1])**

**{**

**result[0] = arr[0];**

**result[1] = arr[1];**

**}**

**else**

**{**

**result[0] = arr[1];**

**result[1] = arr[0];**

**}**

**//set starting index for loop**

**i = 2;**

**}**

**// If array has odd number of elements**

**// then initialize the first element as**

**// minimum and maximum**

**else**

**{**

**result[0] = result[1] = arr[0];**

**//set starting index for loop**

**i = 1;**

**}**

**while(i < n-1)**

**{**

**if(arr[i] > arr[i+1])**

**{**

**result[0] = max(arr[i],result[0]);**

**result[1] = min(arr[i+1],result[1]);**

**}**

**else**

**{**

**result[0] = max(arr[i+1],result[0]);**

**result[1] = min(arr[i],result[1]);**

**}**

**//Increment the index by 2 as**

**//two elements are processed in loop**

**i += 2;**

**}**

**return result;**

**}**

1. **Kth Largest Element LC**

**3.1-> Sorting**

**int findKthLargest(vector<int>& nums, int k) {**

**sort(nums.rbegin() , nums.rend());**

**return nums[k-1];**

**}**

**3.2-> Partial Sorting**

**int findKthLargest(vector<int>& nums, int k) {**

**partial\_sort(nums.begin(), nums.begin() + k, nums.end(), greater<int>());**

**return nums[k - 1];**

**}**

**3.3-> Max Heap**

**int findKthLargest(vector<int>& nums, int k) {**

**priority\_queue<int> pq(nums.begin(), nums.end());**

**for (int i = 0; i < k - 1; i++) {**

**pq.pop();**

**}**

**return pq.top();**

**}**

**3.4-> Min heap**

**3.5-> Partioning**

1. **Given an array which consists of only 0, 1 ,2. Sort the array without using any sorting algo LC**

**void sortColors(vector<int>& nums) {**

**if(nums.size()==0 || nums.size()==1) return;**

**int start=0,end=nums.size()-1,index=0;**

**while(index<=end)**

**{**

**if(nums[index] == 0)**

**{**

**swap(nums[start],nums[index]);**

**index++;**

**start++;**

**}**

**else if(nums[index] == 2)**

**{**

**swap(nums[index],nums[end]);**

**end--;**

**}**

**else**

**{**

**index++;**

**}**

**}**

**}**

1. **Move all the negative elements to one side of the array**

**void shiftall(int arr[], int left, int right)**

**{**

**// Loop to iterate over the**

**// array from left to the right**

**while (left<=right)**

**{**

**// Condition to check if the left**

**// and the right elements are**

**// negative**

**if (arr[left] < 0 && arr[right] < 0)**

**left+=1;**

**// Condition to check if the left**

**// pointer element is positive and**

**// the right pointer element is negative**

**else if (arr[left]>0 && arr[right]<0)**

**{**

**int temp=arr[left];**

**arr[left]=arr[right];**

**arr[right]=temp;**

**left+=1;**

**right-=1;**

**}**

**// Condition to check if both the**

**// elements are positive**

**else if (arr[left]>0 && arr[right] >0)**

**right-=1;**

**else{**

**left += 1;**

**right -= 1;**

**}**

**}**

**}**

1. **Find Union**

**int doUnion(int a[], int n, int b[], int m) {**

**map<int,int> freq;**

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

**freq[a[i]]++;**

**for(int j=0;j<m;j++)**

**freq[b[j]]++;**

**return freq.size();**

**}**

1. **Clockwise rotate array**

**void rotate(int arr[] , int n)**

**{**

**int last = arr[n-1];**

**for(int i=n-1;i>=1;i--)**

**{**

**arr[i] = arr[i-1];**

**}**

**arr[0] = last;**

**}**

***26-02-2021***

1. **Maximum subarray sum**

**int maxSubArray(vector<int>& nums) {**

**int max\_local=nums[0] , max\_global=nums[0];**

**for(int i=1;i<nums.size();i++)**

**{**

**max\_local = max(nums[i] , nums[i]+max\_local);**

**max\_global = max(max\_global , max\_local);**

**}**

**return max\_global;**

**}**

1. **Shortest unsorted Continuous Subarray**

**9.1-> TC-O(N log N) SC-O(N);**

**int findUnsortedSubarray(vector<int>& nums) {**

**vector<int> arr(nums);**

**sort(arr.begin() , arr.end());**

**int start = 0, end = nums.size()-1;**

**for(;start < nums.size(); start++) {**

**if(nums[start] != arr[start]) break;**

**}**

**if(start >= nums.size()-1) return 0;**

**for(; end>=0; end--){**

**if(nums[end] != arr[end]) break;**

**}**

**return end-start+1;**

**}**

**9.2-> public int findUnsortedSubarray(int[] nums) {**

**int min = Integer.MAX\_VALUE, max = Integer.MIN\_VALUE;**

**int n = nums.length;**

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

**//Decreasing Value**

**if(nums[i] < nums[i-1]) {**

**min = Math.min(min,nums[i]);**

**}**

**}**

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

**//Increasing value**

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

**max = Math.max(max,nums[i]);**

**}**

**}**

**if(min == Integer.MAX\_VALUE && max == Integer.MIN\_VALUE) return 0;**

**int start = 0, end = n-1;**

**for(; start < n; start++) {**

**if(nums[start] > min) break;**

**}**

**for(; end >= 0; end--) {**

**if(nums[end] < max) break;**

**}**

**return end-start+1;**

**}**

***27-02-2021***

1. **Valid stack sequence**

**bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {**

**stack<int> stack;**

**int i = 0;**

**for (int x : pushed) {**

**stack.push(x);**

**while (stack.size() && stack.top() == popped[i]) {**

**stack.pop();**

**i++;**

**}**

**}**

**return stack.size() == 0;**

**}**

**31/07/2021**

1. **Given an array, rotate the array to the right by k steps, where k is non-negative.**

class Solution {

public:

    void rotate(vector<int>& nums, int k) {

        reverse(nums.begin() , nums.end());

        reverse(nums.begin(),nums.begin()+k);

        reverse(nums.begin()+k,nums.end());

    }

};