Question 1:

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Solved by myself

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

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

nums1[i]=nums2[i-m];

}

sort(nums1.begin(),nums1.end());

}

};

Question 2 :   
  
Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.

Return k.

Input: nums = [3,2,2,3], val = 3

Output: 2, nums = [2,2,\_,\_]

Solution solved by me

class Solution {

public:

    int removeElement(vector<int>& nums, int val) {

        int k;

        auto it = nums.begin();

        while(it != nums.end()){

            if(\*it==val){

                it=nums.erase(it);

            }

            else{

                it++;

            }

        }

        k=nums.size();

        return k;

    }

};

Question 3

Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.

Return k

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2,\_]

Solved by me

class Solution {

public:

    int removeDuplicates(vector<int>& nums) {

        int j=1;

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

            if(nums[i]!=nums[i-1]){

                nums[j]=nums[i];

                j++;

            }

        }

        return j   ;

    }

};

Question 4   
Remove Duplicates from Sorted Array II

Given an integer array nums sorted in non-decreasing order, remove some duplicates in-place such that each unique element appears at most twice. The relative order of the elements should be kept the same.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Do not allocate extra space for another array. You must do this by modifying the input array in-place with O(1) extra memory.

**Input:** nums = [1,1,1,2,2,3]

**Output:** 5, nums = [1,1,2,2,3,\_]

Not solved by me

class Solution {

public:

    int removeDuplicates(vector<int>& nums) {

        int j=1;

        int c=1;

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

        {

            if(nums[i]==nums[i-1] && c!=2){

                c++;

                nums[j]=nums[i];

                j++;

            }

            else if (nums[i]!=nums[i-1])

            {

                c=1;

                nums[j]=nums[i];

                j++;

            }

        }

        return j;

    }

};

Question 5

Majority Element

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Input: nums = [3,2,3]

Output: 3

Not solved by me

class Solution {

public:

    int majorityElement(vector<int>& nums) {

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

        return nums[(nums.size())/2];

    }

};

Question 6

Rotate Array

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative

**Input:** nums = [1,2,3,4,5,6,7], k = 3

**Output:** [5,6,7,1,2,3,4]

Not solved by me

class Solution {

public:

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

        k= k%nums.size();

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

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

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

    }

};

Question 7

Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Input: prices = [7,1,5,3,6,4]

Output: 5

Not solved by me

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int lowest= prices[0];

        int margin=0;

        for(auto i:prices){

            if(i<lowest){

                lowest=i;

            }

           if (i-lowest>margin)

            {

                margin=i-lowest;

            }

        }

        return margin;

    }

};

Question 8

Best Time to Buy and Sell Stock II

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.

Find and return the maximum profit you can achieve.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [1,2,3,4,5]

Output: 4

Solved by me   
  
class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int lowest= prices[0]; //7

        int margin=0;

        for(int i=0 ;i <prices.size();i++){

            if(lowest>prices[i]){     //l=1

                lowest=prices[i];

            }

            if(prices[i]-lowest>0){

                margin = margin + (prices[i]-lowest);

                lowest = prices[i];

            }

        }

        return margin ;

    }

};

//{7,1,5,3,6,4}

Question 9

Jump Game

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the last index, or false otherwise.

Example 1:

Input: nums = [2,3,1,1,4]

Output: true

Bnot solved by me

class Solution {

public:

    bool canJump(vector<int>& nums) {

        int reachable =0 ;

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

            if(reachable<i){

                return false;

            }

            reachable = max(reachable,i+nums[i]);

        }

        return true;

    }

};

Question 11

H Index

Given an array of integers citations where citations[i] is the number of citations a researcher received for their ith paper, return the researcher's h-index.

According to the definition of h-index on Wikipedia: The h-index is defined as the maximum value of h such that the given researcher has published at least h papers that have each been cited at least h times.

Example 1:

Input: citations = [3,0,6,1,5]

Output: 3

Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3.

Example 2:

Input: citations = [1,3,1]

Output: 1

Not Solved by me :   
  
class Solution {

public:

    int hIndex(vector<int>& citations) {

        sort(citations.begin(),citations.end(),greater<int>());

        int h =0 ;

        while(h<citations.size() && citations[h]>h){

            h++;

        }

        return h;

    }

};

Question13

Product of Array except self

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

Example 1:

Input: nums = [1,2,3,4]

Output: [24,12,8,6]

Example 2:

Input: nums = [-1,1,0,-3,3]

Output: [0,0,9,0,0]

Not solved by me

class Solution {

public:

    vector<int> productExceptSelf(vector<int>& nums) {

        int n = nums.size();

        vector<int> prefix (n,1);

        vector<int> suffix(n,1);

        for(int i=1 ; i<n ; i++){

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

        }

        for(int i=n-2 ; i>=0 ; i--){

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

        }

        for(int i=0 ; i<n ; i++){

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

        }

        return nums;

    }

};

Question 14

Gas Station

There are n gas stations along a circular route, where the amount of gas at the ith station is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from the ith station to its next (i + 1)th station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays gas and cost, return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1. If there exists a solution, it is guaranteed to be unique

Example 1:

Input: gas = [1,2,3,4,5], cost = [3,4,5,1,2]

Output: 3

Explanation:

Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank = 0 + 4 = 4

Travel to station 4. Your tank = 4 - 1 + 5 = 8

Travel to station 0. Your tank = 8 - 2 + 1 = 7

Travel to station 1. Your tank = 7 - 3 + 2 = 6

Travel to station 2. Your tank = 6 - 4 + 3 = 5

Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3.

Therefore, return 3 as the starting index.

Not solved by me

class Solution {

public:

    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {

        int n = gas.size();

        int totalsurplus=0;

        int surplus =0;

        int start=0;

        for(int i=0;i<n;i++){

            totalsurplus+=gas[i]-cost[i];

            surplus+=gas[i]-cost[i];

            if(surplus<0){

                surplus=0;

                start=i+1;

            }

        }

        if(totalsurplus<0){

            return -1;

        }

        else{

            return start;

        }

    }

};

// gas [5,1,2,3,4]

// cost[4,4,1,5,1]