3) Minimum positive sum subarray (contest ques)

24 November 2024 23:19

You are given an integer array nums and **two** integers l and r. Your task is to find the **minimum** sum of a **subarray** whose size is between l and r (inclusive) and whose sum is greater than 0.
Return the **minimum** sum of such a subarray. If no such subarray exists, return -1.
A **subarray** is a contiguous **non-empty** sequence of elements within an array.

From <https://leetcode.com/problems/minimum-positive-sum-subarray/description/>

```
int mini=INT_MAX;
void solve(vector<int>&nums,int k){
    deque<int>q;
    for(int i=0;i<k;i++){
        q.push_back(nums[i]);
    }
    int sum = accumulate(q.begin(),q.end(),0);
    if(sum>0){
        mini=min(mini,sum);
    }
    int i=k;
    while(i<nums.size()){
        sum-=q.front();
        q.pop_front();
        sum+=nums[i];
        q.push_back(nums[i]);
        i++;
        if(sum>0){
            mini = min(mini, sum);
        }
    }
    return;
}
int minimumSumSubarray(vector<int>& nums, int l, int r) {

    for(int i=l;i<=r;i++){
        solve(nums,i);
    }
    if(mini==INT_MAX){
        return -1;
    }
    return mini;
}
```

4) Replace elements with greatest element of right side

29 November 2024 19:47

Given an array arr, replace every element in that array with the greatest element among the elements to its right, and replace the last element with -1.
After doing so, return the array.

From <https://leetcode.com/problems/replace-elements-with-greatest-element-on-right-side/description/>

```
vector<int> replaceElements(vector<int>& arr) {
    int n = arr.size();
    stack<int> st;
    vector<int> ans(n, 0);
    st.push(-1);
    for(int i = arr.size()-1; i >= 0; i--){
        if(arr[i] > st.top()){
            ans[i] = st.top();
            st.pop();
            st.push(arr[i]);
        }
        else{
            ans[i] = st.top();
        }
    }
    return ans;
}
```

5) Replace every element with the least greater element on its right (special stl used)

29 November 2024 20:19

Given an array **arr[]** of **N** integers and replace every element with the least greater element on its right side in the array. If there are no greater elements on the right side, replace it with **-1.**

From <https://www.geeksforgeeks.org/problems/replace-every-element-with-the-least-greater-element-on-its-right/0>

```
vector<int> findLeastGreater(vector<int>& arr, int n) {
      set<int>s;
      vector<int>ans(n,-1);
      for(int i=n-1;i>=0;i--) {
         s.insert(arr[i]);
         auto it = s.upper_bound(arr[i]);
         if(it!=s.end()) {
            ans[i] = *it;
         }
      }
      return ans;
   }
```

5) Kth largest

30 November 2024 15:51

Given an integer array nums and an integer k, return *the* k^{th} *largest element in the array*.
Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.
Can you solve it without sorting?

From <https://leetcode.com/problems/kth-largest-element-in-an-array/description/>

```
int findKthLargest(vector<int>& nums, int k) {
    int n = nums.size();
    sort(nums.begin(), nums.end());
    while(k>1){
        nums.pop_back();
        k--;
    }
    int m = nums.size();
    return nums[m-1];
}
```

From <https://leetcode.com/problems/kth-largest-element-in-an-array/submissions/1439292162/>

6) Top k frequent elements

30 November 2024 17:01

Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

From <https://leetcode.com/problems/top-k-frequent-elements/description/>

```
static bool mycomp(const pair<int, int> &a, const pair<int, int> &b) {
    return a.second > b.second;
}
vector<int> topKFrequent(vector<int>& nums, int k) {
    unordered_map<int, int> freqMap;
    for (int num : nums) {
        freqMap[num]++;
    }
    vector<pair<int, int>> freqVector(freqMap.begin(), freqMap.end());

    // Sort by frequency in descending order
    sort(freqVector.begin(), freqVector.end(), mycomp);
    // Collect the top k elements
    vector<int> result;
    for (int i = 0; i < k && i < freqVector.size(); i++) {
        result.push_back(freqVector[i].first);
    }
    return result;
}
```

7) Best time to buy and sell stocks

09 December 2024 02:43

You are given an array prices where prices[i] is the price of a given stock on the i^{th} 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.

From <https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/>

```
int maxProfit(vector<int>& prices) {
    int profit = 0;
    int curr = prices[0];
    for(int i = 1; i < prices.size(); i++){
        curr = min(curr, prices[i]);
        profit = max(profit, prices[i] - curr);
    }
    return profit;
}
```

9) Print all subarray (also in recursion)

14 February 2025 11:44

Given an array, arr[], generate all possible subarrays using recursion and return them as a vector of vectors.
The subarrays must be returned in the following order:

1. Subarrays starting from the first element, followed by subarrays starting from the second element, and so on.
2. For each starting index, subarrays should be in increasing length.

From <https://www.geeksforgeeks.org/problems/generating-all-subarrays/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
vector<vector<int> > getSubArrays(vector<int>& arr) {
    int n = arr.size();
    vector<vector<int>> ans;
    for(int i = 0; i < n; i++){
        for (int j = i; j < n; j++){
            vector<int>temp;
            for (int k = i; k <= j; k++){
                temp.push_back(arr[k]);
            }
            ans.push_back(temp);
        }
    }
    return ans;
}
```

10) Trapping Rainwater (not done using stack but arrays)

06 November 2024 02:21

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

From <https://leetcode.com/problems/trapping-rain-water/description/>

```
vector<int> getRightMax(vector<int>& height, int n){
    vector<int> rightMax(n) ;
    rightMax[n-1] = height[n-1];
    for (int i = n-2; i >= 0; i--){
        rightMax[i] = max(rightMax[i+1], height[i]);
    }
    return rightMax;
}
vector<int> getLeftMax(vector<int>& height, int n){
    vector<int> leftMax(n) ;
    leftMax[0] = height[0];
    for (int i = 1; i < n; i++){
        leftMax[i] = max(leftMax[i-1], height[i]);
    }
    return leftMax;
}
int trap(vector<int>& height) {
    int n = height.size();
    vector<int> leftMax = getLeftMax(height, n);
    vector<int> rightMax = getRightMax(height, n);
    int sum = 0;
    for(int i = 0; i < n; i++){
        sum += min(leftMax[i], rightMax[i]) - height[i];
    }
    return sum;
}
```

11) Largest Element in an array

26 June 2025 10:42

Given an array **arr[].** The task is to find the **largest** element and return it

From <https://www.geeksforgeeks.org/problems/largest-element-in-array4009/1>

```
int largest(vector<int> &arr) {
    // code here

    int Large = INT_MIN;

    for (int i = 0; i < arr.size(); i++){
        if(arr[i] > Large){
            Large = arr[i];
        }
    }
    return Large;
}
```

From <https://www.geeksforgeeks.org/problems/largest-element-in-array4009/1>

12) Second Largest in an array

26 June 2025 11:50

Given an array of **positive** integers **arr[]**, return the **second largest** element from the array. If the second largest element doesn't exist then return **-1**.
Note: The second largest element should not be equal to the largest element.

From <https://www.geeksforgeeks.org/problems/second-largest3735/1>

```
int getSecondLargest(vector<int> &arr) {
    int large = INT_MIN;
    for(auto it : arr){
        if(it >= large){
            large = it;
        }
    }
    arr.erase(remove(arr.begin(), arr.end(), large), arr.end());
    int secLarge = INT_MIN;
    for(auto it : arr){
        if(it >= secLarge){
            secLarge = it;
        }
    }
    return secLarge == INT_MIN? -1 : secLarge;
}
```

13) Check if array is sorted and rotated

26 June 2025 12:08

Given an array nums, return true *if the array was originally sorted in non-decreasing order, then rotated **some** number of positions (including zero)*. Otherwise, return false.
There may be **duplicates** in the original array.
Note: An array A rotated by x positions results in an array B of the same length such that B[i] == A[(i+x) % A.length] for every valid index i.

From <https://leetcode.com/problems/check-if-array-is-sorted-and-rotated/description/>

```
bool check(vector<int>& nums) {
    int count = 0;
    for (int i = 1; i < nums.size(); i++){
        if(nums[i-1] > nums[i]) count++;
    }
    if(nums[nums.size()-1] > nums[0]) count++;
    return count <= 1;
}
```

14) Remove duplicates from sorted array

26 June 2025 12:20

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.

From <https://leetcode.com/problems/remove-duplicates-from-sorted-array/description/>

```
int removeDuplicates(vector<int>& nums) {
    int i = 0, j = 1;
    while(j < nums.size()){
        if(nums[i] == nums[j]) j++;
        else{
            i++;
            nums[i] = nums[j];
            j++;
        }
    }
    return i+1;
}
```

15) Rotate array

26 June 2025 12:42

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

From <https://leetcode.com/problems/rotate-array/description/>

```
void rotate(vector<int>& nums, int k) {
    int n = nums.size();
    k %= n;
    reverse(nums.begin(), nums.end());
    reverse(nums.begin(), nums.begin() + k);
    reverse(nums.begin() + k, nums.end());
}
```

16) Move Zeroes

26 June 2025 13:12

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
Note that you must do this in-place without making a copy of the array.

From <https://leetcode.com/problems/move-zeroes/description/>

```
void moveZeroes(vector<int>& nums) {
    int i = 0, j = 1;
    int n = nums.size();
    while(j < n){
        if(nums[i] != 0 && nums[j] != 0){
            i++; j++;
        }
        else if(nums[j] == 0 && nums[i] != 0){
            i++; j++;
        }
        else if(nums[i] == 0 && nums[j] != 0){
            swap(nums[i], nums[j]);
            i++; j++;
        }
        else if(nums[i] == 0 && nums[j] == 0) j++;

        if (i >= n - 1) break;
        if (j <= i) j = i + 1;
    }
}
```

17) Longest consecutive Sequence

26 June 2025 15:25

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*
You must write an algorithm that runs in O(n) time.

From <https://leetcode.com/problems/longest-consecutive-sequence/description/>

```
int longestConsecutive(vector<int>& nums) {
    if(nums.size() == 0) return 0;
    set<int>st;
    vector<int>temp;
    for(auto it : nums) st.insert(it);
    for(auto it : st) temp.push_back(it);
    int count  = 0;
    int ans = 0;
    for(int i = 1; i < temp.size(); i++){
        if(temp[i] - temp[i-1] == 1) count++;
        else{
            count = 0;
        }
        ans = max(count, ans);
    }
    return ans+1;
}
```

18) Missing Number

27 June 2025 11:19

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

From <https://leetcode.com/problems/missing-number/description/>

```
int missingNumber(vector<int>& nums) {
    int oldSum = 0;
    int n = nums.size();
    for(auto it : nums){
        oldSum += it;
    }
    int newSum = 0;
    for(int i = 0; i <= nums.size(); i++){
        newSum += i;
    }
    return newSum - oldSum;
}
```

19) Max Consecutive Ones

27 June 2025 11:31

Given a binary array nums, return *the maximum number of consecutive* 1*'s in the array*.

From <https://leetcode.com/problems/max-consecutive-ones/description/>

```
int findMaxConsecutiveOnes(vector<int>& nums) {
    int count = 0;
    int ans = 0;
    for(auto it: nums){
        if(it == 1){
            count++;
        }
        else count = 0;
        ans = max(ans, count);
    }
    return ans;
}
```

20) Single Number

27 June 2025 11:32

Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.
You must implement a solution with a linear runtime complexity and use only constant extra space.

From <https://leetcode.com/problems/single-number/description/>

```
int singleNumber(vector<int>& nums) {
    int a = 0;
    for (int i = 0; i  <nums.size(); i++){
        a = a^nums[i];
    }
    return a;
}
```

21) Subarray Sum equals K

27 June 2025 17:16

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to* k.
A subarray is a contiguous **non-empty** sequence of elements within an array.

From <https://leetcode.com/problems/subarray-sum-equals-k/description/>

```
int subarraySum(vector<int>& nums, int k) {
    int count = 0;
    for (int i = 0; i < nums.size(); i++){
        int sum = 0;
        sum = nums[i];
        if(sum == k) count++;
        for(int j = i+1; j < nums.size(); j++){
            sum += nums[j];
            if(sum == k) count++;
        }
    }
    return count;
}
```

22) Subarray sum equals k

27 June 2025 17:17

Given an array arr[] containing integers and an integer k, your task is to find the length of the longest subarray where the sum of its elements is equal to the given value k. If there is no subarray with sum equal to k, return 0.

From <https://www.geeksforgeeks.org/problems/longest-sub-array-with-sum-k0809/1>

```
int longestSubarray(vector<int>& arr, int k) {
    int ans = 0;
    for(int i = 0; i < arr.size(); i++){
        int sum = 0;
        sum  = arr[i];
        if(sum == k) ans = 1;
        for(int j = i+1; j < arr.size(); j++){
            sum += arr[j];
            if(sum == k){
                ans = max(ans, j+1-i);
            }
        }
    }
    return ans;
}
```

23) Longest Consecutive Sequence

27 June 2025 17:18

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*
You must write an algorithm that runs in O(n) time.

From <https://leetcode.com/problems/longest-consecutive-sequence/description/>

```
int longestConsecutive(vector<int>& nums) {
    if(nums.size() == 0) return 0;
    set<int>st;
    vector<int>temp;
    for(auto it : nums) st.insert(it);
    for(auto it : st) temp.push_back(it);
    int count  = 0;
    int ans = 0;
    for(int i = 1; i < temp.size(); i++){
        if(temp[i] - temp[i-1] == 1) count++;
        else{
            count = 0;
        }
        ans = max(count, ans);
    }
    return ans+1;
}
```

24) Two sum

29 June 2025 19:59

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.
You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.
You can return the answer in any order.

From <https://leetcode.com/problems/two-sum/description/>

```
vector<int> twoSum(vector<int>& nums, int target) {
    int n = nums.size();
    int i = 0; int j = n-1;
    vector<int>ans;
    while(i < j){
        if(nums[i] + nums[j] == target){
            ans.push_back(i);
            ans.push_back(j);
        }
        else if(nums[i] + nums[j] > target)
    }
}
```

25) Sort color

29 June 2025 20:00

Given an array nums with n objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.
We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.
You must solve this problem without using the library's sort function.

From <https://leetcode.com/problems/sort-colors/description/>

```
void sortColors(vector<int>& nums) {
    int i = 0; int j = i+1;
    while(i < nums.size() && j < nums.size()){
        if(nums[i] < nums[j]){
            swap(nums[i], nums[j]);
        }
        i++;j++;
    }
    reverse(nums.begin(), nums.end());
}
```

26) Majority Element

29 June 2025 20:01

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.

From <https://leetcode.com/problems/majority-element/description/>

```
int majorityElement(vector<int>& nums) {
    int n = nums.size();
    unordered_map<int, int>mpp;
    for(int i = 0; i < n; i++){
        mpp[nums[i]]++;
    }
    for(auto i: mpp){
        if(i.second > n/2){
            return i.first;
        }
    }
    return -1;
}
```

27) Rearrange array elements by sign

29 June 2025 20:02

You are given a **0-indexed** integer array nums of **even** length consisting of an **equal** number of positive and negative integers.
You should return the array of nums such that the the array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in nums is **preserved**.
3. The rearranged array begins with a positive integer.

Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*.

From <https://leetcode.com/problems/rearrange-array-elements-by-sign/description/>

```
vector<int> rearrangeArray(vector<int>& nums) {
    vector<int>temp1;
    vector<int>temp2;
    vector<int>temp;
    for (int i = 0; i < nums.size(); i++){
        if(nums[i] < 0){
            temp2.push_back(nums[i]);
        }
        else{
            temp1.push_back(nums[i]);
        }
    }
    int n  = max(temp1.size(), temp2.size());
    for (int i = 0; i < n; i++){
        temp.push_back(temp1[i]);
        temp.push_back(temp2[i]);
    }
    return temp;
}
```

28) Next Permutations

29 June 2025 20:03

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

From <https://leetcode.com/problems/next-permutation/description/>

```
void nextPermutation(vector<int>& nums) {
    next_permutation(nums.begin(), nums.end());
}
```

From <https://leetcode.com/problems/next-permutation/submissions/1237098124/>

29) Array leaders

29 June 2025 20:05

You are given an array arr of positive integers. Your task is to find all the leaders in the array. An element is considered a leader if it is greater than or equal to all elements to its right. The rightmost element is always a leader.

From <https://www.geeksforgeeks.org/problems/leaders-in-an-array-1587115620/1>

```
// vector<int> leaders(vector<int>& arr) {
//     vector<int>ans;
//     int n = arr.size();
//     ans.push_back(arr[n-1]);
//     for(int i = n-2; i >= 0; i--){
//         int j = i+1;
//         while(j < n && arr[j] <= arr[i]){
//             j++;
//         }
//         if(j == n) ans.push_back(arr[i]);
//     }
//     reverse(ans.begin(), ans.end());
//     return ans;
// }
```

30) Set Matrix Zero

29 June 2025 20:05

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.
You must do it in place.

From <https://leetcode.com/problems/set-matrix-zeroes/description/>

```
void setZeroes(vector<vector<int>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size();
    vector<vector<int>> temp = matrix;
    for (int i = 0; i < n; i++){
        for (int j= 0; j < m; j++){
            if(matrix[i][j] == 0){
                for(int k = 0; k < m; k++){
                    temp[i][k] = 0;
                }
                for(int k = 0; k < n; k++){
                    temp[k][j] = 0;
                }
            }
        }
    }
    matrix = temp;
}
```

31) Rotate Image

29 June 2025 20:29

You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).
You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

From <https://leetcode.com/problems/rotate-image/description/>

```
void rotate(vector<vector<int>>& matrix) {
    for(int i = 0; i < matrix.size(); i++){
        for(int j = i; j < matrix[0].size(); j++){
            swap(matrix[i][j], matrix[j][i]);
        }
    }
    for(int i = 0; i < matrix.size(); i++){
        reverse(matrix[i].begin(), matrix[i].end());
    }
}
```

32) Spiral Matrix

29 June 2025 20:57

Given an m x n matrix, return all elements of the matrix in spiral order.

From <https://leetcode.com/problems/spiral-matrix/description/>

```
vector<int> spiralOrder(vector<vector<int>>& matrix) {
    int m = matrix.size();
    int n = matrix[0].size();
    vector<int>ans;
    int top = 0;
    int left = 0;
    int bottom = m-1;
    int right = n-1;
    while(left <= right && top <= bottom){
        for(int i = left; i <= right; i++){
            ans.push_back(matrix[top][i]);
        }
        top++;
        for(int i = top; i <= bottom; i++){
            ans.push_back(matrix[i][right]);
        }
        right--;
        if(top <= bottom){
            for(int i = right; i >= left; i--){
                ans.push_back(matrix[bottom][i]);
            }
            bottom--;
        }
        if(left <= right){
            for(int i = bottom; i >= top; i--){
                ans.push_back(matrix[i][left]);
            }
            left++;
        }
    }
    return ans;
}
```

33)Pascal Triangle

05 July 2025 11:15

Given an integer numRows, return the first numRows of **Pascal's triangle**.
In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:

From <https://leetcode.com/problems/pascals-triangle/description/>

```
vector<int> helper(int row){
    long long ans = 1;
    vector<int>temp;
    temp.push_back(1);
    for(int col = 1; col < row; col++){
        ans = ans*(row-col);
        ans = ans/col;
        temp.push_back(ans);
    }
    return temp;
}
vector<vector<int>> generate(int numRows) {
    vector<vector<int>>ans;
    for(int i = 1; i <= numRows; i++){
        ans.push_back(helper(i));
    }
    return ans;
}
```

From <https://leetcode.com/problems/pascals-triangle/submissions/1682294704/>

34) Majority Element II

05 July 2025 11:16

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

From <https://leetcode.com/problems/majority-element-ii/description/>

```
vector<int> majorityElement(vector<int>& nums) {
    int n = nums.size();
    vector<int>ans;
    unordered_map<int, int>mpp;
    for(int i = 0; i < n; i++){
        mpp[nums[i]]++;
    }
    for(auto i: mpp){
        if(i.second > n/3){
            ans.push_back(i.first);
        }
    }
    return ans;
}
```

From <https://leetcode.com/problems/majority-element-ii/submissions/1682339610/>

35) 3-sum

05 July 2025 11:17

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.
Notice that the solution set must not contain duplicate triplets.

From <https://leetcode.com/problems/3sum/description/>

```
vector<vector<int>> threeSum(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    int n = nums.size();
    vector<vector<int>>ans;
    for(int i = 0; i < nums.size(); i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        int j = i+1;
        int k = n-1;
        while(j < k){
            int sum = nums[i] + nums[j] + nums[k];
            if(sum < 0) j++;
            else if(sum > 0) k--;
            else {
                vector<int>temp = {nums[i], nums[j], nums[k]};
                ans.push_back(temp);
                j++; k--;
                while(j < k && nums[j] == nums[j-1]) j++;
                while(j < k && nums[k] == nums[k+1]) k--;
            }
        }
    }
    return ans;
}
```

From <https://leetcode.com/problems/3sum/submissions/1682361893/>

36) 4 sum

05 July 2025 11:18

Given an array nums of n integers, return *an array of all the **unique** quadruplets* [nums[a], nums[b], nums[c], nums[d]] such that:

- 0 <= a, b, c, d < n
- a, b, c, and d are **distinct**.
- nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in **any order**.

From <https://leetcode.com/problems/4sum/description/>

```
vector<vector<int>> fourSum(vector<int>& nums, int target) {
    sort(nums.begin(), nums.end());
    int n = nums.size();
    vector<vector<int>> ans;
    for (int i = 0; i < nums.size(); i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        for (int j = i+1; j < nums.size(); j++){
            if(j != (i+1) && nums[j] == nums[j-1]) continue;
            int k = j+1;
            int l = n-1;
            while(k < l){
                long long sum = nums[i];
                sum += nums[j];
                sum += nums[k];
                sum += nums[l];
                if(sum < target){
                    k++;
                }
                else if(sum > target){
                    l--;
                }
                else{
                    vector<int> temp = {nums[i], nums[j], nums[k], nums[l]};
                    ans.push_back(temp);
                    k++;
                    l--;
                    while(k < l && nums[k] == nums[k-1]) k++;
                    while(k < l && nums[l] == nums[l+1]) l--;
                }
            }
        }
    }
    return ans;

}
```

From <https://leetcode.com/problems/4sum/submissions/1411661488/>

37) Largest subarray with 0 sum

05 July 2025 11:19

Given an array **arr[]** containing both positive and negative integers, the task is to find the **length** of the **longest subarray** with a sum equals to **0.**
Note: A subarray is a contiguous part of an array, formed by selecting one or more consecutive elements while maintaining their original order.

From <https://www.geeksforgeeks.org/problems/largest-subarray-with-0-sum/1>

```
  // int maxLen(vector<int>& arr) {
  //     int ans = 0;
  //     for(int i = 0; i < arr.size(); i++){
  //         int sum = 0;
  //         for(int j = i; j < arr.size(); j++){
  //             sum += arr[j];
  //             if(sum == 0){
  //                 ans = max(ans, j+1-i);
  //             }
  //         }
  //     }
  //     return ans;
  // }

int maxLen(vector<int>& arr) {
    unordered_map<int, int> prefixMap;
    int sum = 0, maxLength = 0;

    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];

        if (sum == 0) {
            maxLength = i + 1;
        }

        // Only store the first occurrence of the prefix sum
        if (prefixMap.find(sum) != prefixMap.end()) {
            maxLength = max(maxLength, i - prefixMap[sum]);
        } else {
            prefixMap[sum] = i;
        }
    }

    return maxLength;
}
```

38) Merge Intervals

05 July 2025 11:25

Given an array of intervals where intervals[i] = [start, end], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

From <https://leetcode.com/problems/merge-intervals/description/>

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
    sort(intervals.begin(), intervals.end());
    vector<vector<int>>ans;
    for(int i = 0; i < intervals.size(); i++){
        if(ans.empty() || intervals[i][0] > ans.back()[1]){
            ans.push_back(intervals[i]);
        }
        else{
            ans.back()[1] = max(intervals[i][1], ans.back()[1]);
        }
    }
    return ans;
}
```

From <https://leetcode.com/problems/merge-intervals/submissions/1684587690/>

39) Merge Sorted Array

05 July 2025 11:26

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.
Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.
The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

From <https://leetcode.com/problems/merge-sorted-array/description/>

```
void inplace(vector<int>& nums, int start, /*int mid*/int end){
    int len = end-start+1;
    int gap = len/2 + len%2;
    while(gap > 0){
        int i = start; int j = gap+start;
        while(j <= end){
            if(nums[i] > nums[j]){
                swap(nums[i], nums[j]);
            }
            i++; j++;
        }
        gap = gap <= 1?0: gap = gap/2 + gap%2;
    }
}
void merge(vector<int>& arr1, int m, vector<int>& arr2, int n) {
    int j = 0;
    int k = m;
    while(j < n){
        arr1[k] = arr2[j];
        j++;k++;
    }
    inplace(arr1, 0, m+n-1);
}
```

From <https://leetcode.com/problems/merge-sorted-array/submissions/1684629767/>

40) Missing and repeating Elements

05 July 2025 11:27

Given an unsorted array ***arr*** of positive integers. One number **a** from the set **[1, 2,....,n]** is missing and one number **b** occurs twice in the array. Find numbers **a** and **b**.
Note: The test cases are generated such that there always exists one missing and one repeating number within the range **[1,n]**.

From <https://www.geeksforgeeks.org/problems/find-missing-and-repeating2512/1>

```
vector<int> findTwoElement(vector<int>& arr) {
    vector<int> ans;
    int n = arr.size();
    int sum1 = 0;
    for(int i = 1; i <= n; i++){
        sum1 += i;
    }
    int sum2 = 0;
    sort(arr.begin(), arr.end());
    for(int i = 0; i < arr.size(); i++){
        if(i > 0 && arr[i] == arr[i-1]){
            ans.push_back(arr[i]);
        }
        sum2 += arr[i];
    }
    sum2 -= ans[0];
    ans.push_back(sum1-sum2);
    return ans;
}
```

From <https://www.geeksforgeeks.org/problems/find-missing-and-repeating2512/1>

41) Maximum Product Subarray

05 July 2025 11:33

Given an integer array nums, find a subarray that has the largest product, and return *the product*.
The test cases are generated so that the answer will fit in a **32-bit** integer.

From <https://leetcode.com/problems/maximum-product-subarray/description/>

```
int maxProduct(vector<int>& nums) {
        int maxi = nums[0];
        for (int i = 0; i < nums.size(); i++){
            int product = nums[i];
            for (int j = i+1; j < nums.size(); j++){
                maxi = max(maxi, product);
                product = product * nums[j];
            }
            maxi = max(maxi, product);
        }
        return maxi;
    }
```

From <https://leetcode.com/problems/maximum-product-subarray/submissions/1412529655/>

42) Count Inversion

05 July 2025 12:04

Given an array of integers **arr[]**. Find the **Inversion Count** in the array.
Two elements arr[i] and arr[j] form an inversion if **arr[i] > arr[j]** and **i < j**.

***Inversion Count**:* For an array, inversion count indicates how far (or close) the array is from being sorted. If the array is already sorted then the inversion count is 0.
If an array is sorted in the reverse order then the inversion count is the maximum.

From <https://www.geeksforgeeks.org/problems/inversion-of-array-1587115620/1>

```
int count = 0;

void merge(vector<int>& arr, int low, int mid, int high){
   vector<int> temp;
   int left = low;
   int right = mid + 1;

   while(left <= mid && right <= high){
      if(arr[left] <= arr[right]) {
         temp.push_back(arr[left]);
         left++;
      } else {
         temp.push_back(arr[right]);
         count += (mid - left + 1); // Inversions here
         right++;
      }
   }

   while(left <= mid) {
      temp.push_back(arr[left]);
      left++;
   }

   while(right <= high) {
      temp.push_back(arr[right]);
      right++;
   }

   for(int i = low; i <= high; i++) {
      arr[i] = temp[i - low];
   }
}

void mergeSort(vector<int>& arr, int low, int high){
   if(low >= high) return;
   int mid = (low + high) / 2;
   mergeSort(arr, low, mid);
   mergeSort(arr, mid + 1, high);
   merge(arr, low, mid, high);
}
```

```
int inversionCount(vector<int>& arr){
   count = 0; // ☑ Always reset before use
   mergeSort(arr, 0, arr.size() - 1);
   return count;
}
```

43) Reverse piars

05 July 2025 12:26

Given an integer array nums, return *the number of* ***reverse pairs*** *in the array*.
A **reverse pair** is a pair (i, j) where:

- 0 <= i < j < nums.length and
- nums[i] > 2 * nums[j].

From <https://leetcode.com/problems/reverse-pairs/description/>

```
int count = 0;
void merge(vector<int>& arr, int low, int mid, int high){
    // Standard merge process
    vector<int> temp;
    int left = low, right = mid + 1;
    while(left <= mid && right <= high) {
        if(arr[left] <= arr[right]) {
            temp.push_back(arr[left++]);
        } else {
            temp.push_back(arr[right++]);
        }
    }
    while(left <= mid) temp.push_back(arr[left++]);
    while(right <= high) temp.push_back(arr[right++]);
    for(int i = low; i <= high; i++) {
        arr[i] = temp[i - low];
    }
}
void countPairs(vector<int>& arr, int low, int mid, int high){
    int right = mid + 1;
    for(int i = low; i <= mid; i++) {
        while(right <= high && (long long)arr[i] > 2LL * arr[right]) {
            right++;
        }
        count += (right - (mid + 1));
    }
}
void mergeSort(vector<int>& arr, int low, int high){
    if(low >= high) return;
    int mid = (low + high) / 2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid + 1, high);
    countPairs(arr, low, mid, high);
    merge(arr, low, mid, high);
}
int reversePairs(vector<int>& nums) {
    mergeSort(nums, 0, nums.size()-1);
    return count;
}
```

44) Count Subarrays with given XOR

05 July 2025 13:53

Given an array of integers **arr[]** and a number **k**, count the number of subarrays having **XOR** of their elements as **k**.

From <https://www.geeksforgeeks.org/problems/count-subarray-with-given-xor/1>

```
long subarrayXor(vector<int> &arr, int k) {
    unordered_map<int, int> freq;
    long count = 0, xorSum = 0;

    for (int num : arr) {
        xorSum ^= num;

        if (xorSum == k)
            count++;

        // Check if there is a prefix XOR that when XOR-ed with current gives k
        if (freq.find(xorSum ^ k) != freq.end())
            count += freq[xorSum ^ k];

        // Store current prefix XOR in map
        freq[xorSum]++;
    }

    return count;
}
```

45) Maximum Subarray Sum (Kadane's Algo)

05 July 2025 14:32

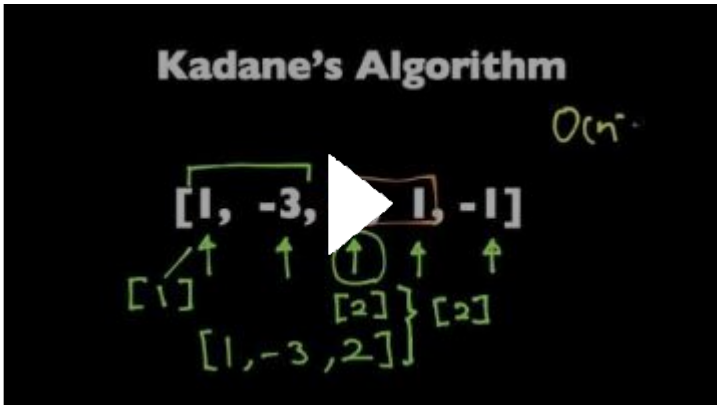
Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

From <https://leetcode.com/problems/maximum-subarray/description/>

```
int maxSubArray(vector<int>& nums) {
    int curr = nums[0];
    int global = nums[0];
    for(int i = 1; i < nums.size(); i++){
        curr = max(nums[i], curr + nums[i]);
        if(curr > global) global = curr;
    }
    return global;
}
```

//for reference, watch the tutorial

Kadane's Algorithm to Maximum Sum Subarray Problem



46) Print subarray with maximum subarray sum (extended version of above problem)

05 July 2025 14:40

```
int maxSubArray(vector<int>& nums) {
    int curr = nums[0], global = nums[0];
    int start = 0, end = 0, tempStart = 0;

    for (int i = 1; i < nums.size(); i++) {
        if (nums[i] > curr + nums[i]) {
            curr = nums[i];
            tempStart = i; // start new subarray
        } else {
            curr += nums[i];
        }

        if (curr > global) {
            global = curr;
            start = tempStart;
            end = i;
        }
    }

    // Print the subarray
    cout << "Maximum Subarray: ";
    for (int i = start; i <= end; i++) {
        cout << nums[i] << " ";
    }
    cout << endl;

    return global;
}
```

1) Find Missing and repeated values

05 July 2025 11:35

You are given a **0-indexed** 2D integer matrix grid of size n * n with values in the range [1, n^2]. Each integer appears **exactly once** except a which appears **twice** and b which is **missing**. The task is to find the repeating and missing numbers a and b.
Return *a **0-indexed** integer array* ans *of size* 2 *where* ans[0] *equals to* a *and* ans[1] *equals to* b.

From <https://leetcode.com/problems/find-missing-and-repeated-values/description/>

```
vector<int> findMissingAndRepeatedValues(vector<vector<int>>& grid) {
    int n = grid.size();
    int size = n*n;
    vector<int>count(size+1, 0);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            count[grid[i][j]]++;
        }
    }
    vector<int>ans;
    for(int i = 1; i <= size; i++){
        if(count[i] == 2) ans.push_back(i); // repeated number
    }
    for(int i = 1; i <= size; i++){
        if(count[i] == 0) ans.push_back(i); // missing number
    }
    return ans;
}
```

From <https://leetcode.com/problems/find-missing-and-repeated-values/submissions/1684665046/>