Created by CodersRound
Mr. Pramod Bagali
Visit -
www.codersround.online
Youtube - Link

Q. Two Sum →

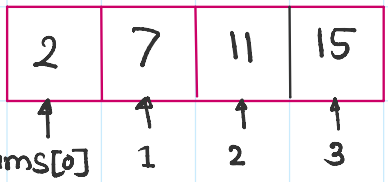
Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.
You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.
You can return the answer in any order.

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

Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Solution →

nums [4] =



nums is taken as a
n = size of array
target = 9

Brute Force →

We have two nested for loops for i & j

```
for(i=0; i<n; i++){
    for(j=i; j<n; j++){
        if(a[i] + a[j] == target){
            return{i,j};
        }
    }
}
```

```
vector <int> targetSum(vector <int> &a, int &target){
    int array_size = a.size();
    for(int i=0; i<array_size-1; i++){
        for(int j=i+1; j<array_size;j++){
            if(a[i] + a[j] == target){
                return{i, j};
            }else{return {-1, -1};}
        }
    }
    return{};
}
int main(){
    vector <int> a = {2,7,11,15};
    int target = 9;
    for(int &x: targetSum(a, target)){
        cout << x;
    }
    cout << endl;
}
```

How this brute force approach works ?

→

step 1 :- a[i]=0 {2}. a[j]=1 {7}

Step 1 :- a[i]=0 {2 ... a[j]=1 [7]

2+7 == 9 (Returns (0,1))

Time Complexity ⇒ $O(n^2)$

Space Complexity ⇒ $O(1)$

Optimized Approach ⇒ Binary Search / Two pointer

left = 0, right = n-1, tempSum = 0;

```
while (left < right) {
    tempSum = a[left] + a[right];
    if (tempSum == target){
        return {left, right};
    }
    else if (tempSum > target){
        right--;
    }
    else{
        left++;
    }
}
```

```
vector <int> targetSum(vector <int> &a, int &target){
    int left = 0, right = a.size() - 1, tempSum;
    while(left < right){
        tempSum = a[left] + a[right];
        if(tempSum == target){
            return{left, right};
        }
        else if(tempSum > target){
            right--;
        }
        else{
            left++;
        }
    }
    return {};
}
int main(){
    vector <int> a = {2,7,11,15};
    int target = 9;
    for(int &x: targetSum(a, target)){
        cout << x;
    }
    cout << endl;
}
```

How this Optimized Solution works ?

⇒

0	1	2	3
2	7	11	15

left ↑ ↑ ↑ ↑

0	1	2	3
2	7	11	15

↑ ↑ ↑ ↑ right

target = 9

Step 1: a[0] = 2 ... a[3] = 15 right--

2 + 15 = 17 > 9

2 + 15 = 17 > 9

Step 2 : a[0] = 2 a[2] = 11

7 + 11 = 18 > 9 right--

Step 3:- a[0] = 2 a[1] = 7

2 + 7 = 9 == 9 return {left, right}

0 , 1

Time Complexity :- O(n)

Space Complexity :- O(1)

Q. Best Time to Buy & 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.

Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

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

Example 2:
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

Solution ⇒

	0	1	2	3	4	5
prices =	7	1	5	6	3	4

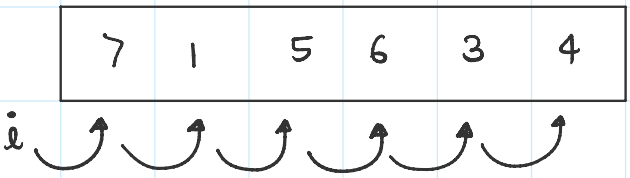
Brute Force Approach :-

{ int n is a size of prices array

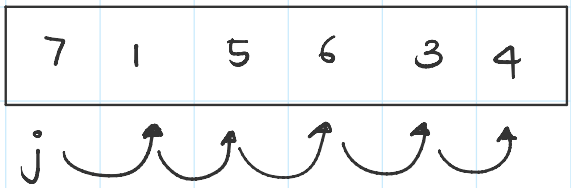
```
int answer = 0
for (int i=0; i<n ; i++) {
    for(int j=i+1;j<n ; i++){
        answer = max(answer, prices[j] - prices[i]);
    }
}
return answer;
}
```

How this Brute Force approach works ?

⇒



i = 7



j = 1

answer = max (answer, prices[j] - prices[i]); (answer max)

i = 7	j = 1	⇒	1 - 7 = -6	(0, -6) ⇒ 0
i = 7	j = 5	⇒	5 - 7 = -2	(0, -2) ⇒ 0
i = 7	j = 6	⇒	6 - 7 = -1	(0, -1) ⇒ 0
i = 7	j = 3	⇒	3 - 7 = -4	(0, -4) ⇒ 0
i = 7	j = 4	⇒	4 - 7 = -3	(0, -3) ⇒ 0
i = 1	j = 5	⇒	5 - 1 = 4	(0, 4) ⇒ 4
	j = 6	⇒	6 - 1 = 5	(4, 5) ⇒ 5
	j = 3	⇒	3 - 1 =	
	j = 4	⇒	4 - 1 =	

j=4 ⇒ 4-1

This process goes on upto (i=4, j=4) but it stored max answer = 5

Time Complexity ⇒ $O(n^2)$

Space Complexity ⇒ $O(1)$

Optimized Solution :-

```
int answer = 0;
int mi = INT_MAX;
for(auto ele: prices){
answer = max(answer, ele-mi);
mi = min(mi, ele);
}
return answer;
}
```

```
int BesttimeStock(vector<int> &prices){
    int answer = 0;
    int mi = INT_MAX;
    for(auto ele: prices){
        answer = max(answer, ele-mi);
        mi = min(mi, ele);
    }

    return answer;
}
int main(){
    vector <int> prices = {7, 1, 5, 3, 6, 4};
    cout << BesttimeStock(prices)<<endl;
}
```

prices =

0	1	2	3	4	5
7	1	5	3	6	4

How optimized solution works ?

mi = INT_MAX; = 7
{answer = max(answer, ele - mi);
mi = min(ele, mi);}

ele = 7 → answer = max(0, 7-7) = 0
min = mi(7, 7) = 7

ele = 1 → answer = max(0, 1-7) = 0
min = mi(1, 7) = 1

ele = 5 → answer = max(0, 5-1) = 4
min = mi(5, 1) = 1

ele = 3 → ans = max(4, 3-1) = 4
min = mi(3, 1) = 1

ele = 6 → answer = max(4, 6-1) = 5 | ele = 4 → ans = max(5, 4-1) = 5

min = mi(6,1) = 1 | min = mi(4,1) = 1

Maximum profit is 5 returned.

Time Complexity :- O(n)

Space Complexity :- O(1)

Q. Merge two Sorted Array

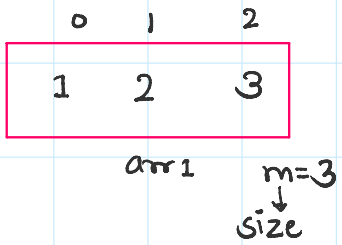
You are given two integer arrays nums1 and nums2, sorted in **non-decreasing order**, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.
Merge nums1 and nums2 into a single array sorted in **non-decreasing order**.
The final sorted array should not be returned 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.

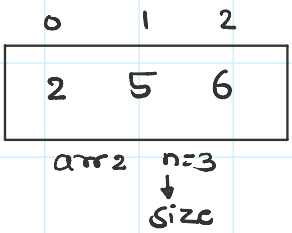
Example 1:
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

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

Note :- Two arrays must be sorted.

Solution:-





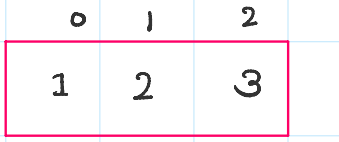

```
void merge(vector<int> &a, int m, vector<int> &b, int n){
    int i=m-1, j=n-1, idx = m+n-1;
    while(i>=0 && j>=0){
        if(a[i] >= b[j]){
            a[idx] = a[i];
            i--;
            idx--;
        }else{
            a[idx] = b[j];
            j--;
            idx--;
        }
    }
    while(i>=0){
        a[idx--] = a[i--];
    }
```

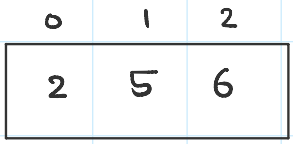
Time Complexity → O(m+n)

Space Complexity → O(n)

```
    while(j>=0){
        a[idx--] = b[j--];
    }
}
int main(){
    vector<int> a = {1,2,3};
    vector<int> b = {2,6,7};
    int m=3, n=3;
    a.resize(m+n);
    merge(a,m,b,n);
    for(int &i: a){
        cout << i<< " ";
    }
}
```

How this solution works?





i = m-1, j = n-1, idx = m+n-1

i = 3-1 j = 3-1 idx = 3+3-1

i = 2 j = 2 idx = 5

idx = m+n-1 = 5

a[] = 1, 2, 3, 0, 0, 0

0 1 2 3 4 5

a[] 1 2 3 ↑ i=2

b[] 2 5 6 ↑ j=2

```
if(a[i] >= b[j]){
    a[idx] = a[i]
    i--;
    idx--;
}
else{
    a[idx] = b[j];
    j--;
    idx--;
}
```

a[5] = 0 ≥ b[2] = 6 (not works)

So else starts

a[5] = 6

1, 2, 3, _, _, 6 ↑ a[5]

This type whole process continues.....

At last, a[] = 1,2,2,3,5,6 merged array.

Q. Occurs Once

You are given an array A of length N, where N is always an odd integer. There are (N-1)/2 elements in the array that occur twice and one element which occurs once. Your task is to find the only element that occurs once in the array.
Note: There are (N-1)/2+1 elements that are unique in the array.

From <https://www.codingninjas.com/codestudio/problems/occurs-once_1214969?leftPanelTab=0>

Input - 7 3 5 4 5 3 4
Output - 7

⇒ 7 occured only once

Solution ⇒

7	3	5	4	5	3	4

Using XOR approach
(if two operands are different then only returns true)

```
int occursonce(vector<int> &a, int n)
{
    int answer = 0;
    for (int i = 0; i < n; i++)
    {
        answer = answer ^ a[i];
    }
    return answer;

}
int main(){
    vector <int> a = {7,3,5,4,5,3,4};
    int n = 8;
    cout << occursonce(a,n);
}
```

Time Complexity ⇒ O(n)
Space Complexity ⇒ O(1)

Q. Contains Duplicate

Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true
Example 2:
Input: nums = [1,2,3,4]
Output: false

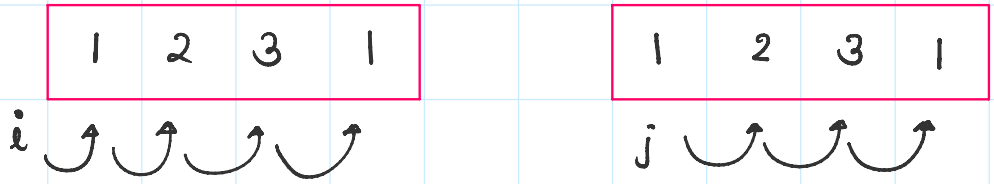
From <https://leetcode.com/problems/contains-duplicate/>

→ Solution :-

1) Brute Force →

We are going to run two loops comparing i^{th} & j^{th} element if any

We are going to run two loops compare i^{th} & j^{th} element if any same element finds just returns true otherwise false.



➡ How this approach works?

```
bool containsDuplicate_bruteforce(int *nums, int l) {
   for (int i=0; i<l; i++) {
      for (int j=i+1; j<l; j++) {
         if (nums[i] == nums[j]) {
            return true;
         }
      }
   }
   return false;
}
int main(){
  int nums[] = {1,2,3,1};
  int l = sizeof(nums)/sizeof(nums[0]);
  cout << containsDuplicate_bruteforce(nums,l);
}
```

i = 0, j = 1 (1) (2) ⟹ nothing returned

i = 0, j = 2 (1) (3) ⟹ "

i = 0, j = 3 (1) (1) ⟹ returns true, i = 1, same element found.

Time Complexity → $O(n^2)$ Space Complexity → $O(1)$

Normal Solution :-

This approach is like to sort all elements and then see that there any 2 (↓ may more) same elements are present then returns true.

How this approach works?

```
bool containsDuplicate(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        for(int i=0;i<nums.size()-1;i++)
            if(nums[i]==nums[i+1]){
                return true;
            }
        return false;
    }
int main(){
    vector <int> a = {1,2,3,1};
    cout << containsDuplicate(a);
}
```

1, 2, 3, 1

⟹ sort() → works

[Note:- sort() function contains mixture of insertion sort, heap sort & Quick Sort] called as Intro Sort

⟹ 1, 1, 2, 3 → returns true

→ if (nums[i] == nums[i+1]){

Time Complexity → $O(n \log n)$

Time Complexity → O(nlogn)

⇒ 1, 1, 2, 3 → returns true

⇒ if (nums[i] == nums[i+1]){
return true
}

Q. Majority Element.

You have been given an array/list 'ARR' consisting of 'N' integers. Your task is to find the majority element in the array. If there is no majority element present, print -1.

From <https://www.codingninjas.com/codestudio/problems/majority-element_842495?leftPanelTab=0>

Input -
1, 1, 2, 1, 3, 5, 1
Output - 1

Solution ⟹

Brute Force :-

```
void findMajority(int arr[], int n)
{
    int maxCount = 0;
    int index = -1;
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[i] == arr[j])
                count++;
        }
        if (count > maxCount) {
            maxCount = count;
            index = i;
        }
    }
    if (maxCount > n / 2)
        cout << arr[index] << endl;
    else
        cout << "No Majority Element" << endl;
}
int main()
{
    int arr[] = { 1, 1, 2, 1, 3, 5, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    findMajority(arr, n);
    return 0;
}
```

arr = 1, 1, 2, 1, 3, 5, 1

How this approach works ?

→ maxCount = 0 , index = -1

if (arr[i] == arr[j]){
count++;
}

i j
= 1 = 1 → count = 1, maxCount = 1, index = 0

i j
1 ≠ 2

1 = 1 → count = 2, maxCount = 2, index = 3

i j
1 ≠ 3

1 ≠ 5

1 = 1 → count = 3, maxCount = 3, index = 5

So last return a[i] = a[5] = 1

maximum/majority times element is **1**

Time Complexity :- O(n²) Space Complexity :- O(1)

Optimal Solution :-

(Moore's Voting Algorithm)

```
#include <bits/stdc++.h>
using namespace std;
int findCandidate(int a[], int size)
{
    int maj_index = 0, count = 1;
    for (int i = 1; i < size; i++) {
```

1. Loop through each element and maintains a count of majority element, and a majority index, *maj_index*
2. If the next element is same then increment the count if the next element is not same then decrement the count

```cpp
using namespace std;
int findCandidate(int a[], int size)
{
    int maj_index = 0, count = 1;
    for (int i = 1; i < size; i++) {
        if (a[maj_index] == a[i])
            count++;
        else
            count--;
        if (count == 0) {
            maj_index = i;
            count = 1;
        }
    }
    return a[maj_index];
}
bool isMajority(int a[], int size, int cand)
{
    int count = 0;
    for (int i = 0; i < size; i++)
        if (a[i] == cand)
            count++;
    if (count > size / 2)
        return 1;
    else
        return 0;
}
void printMajority(int a[], int size)
{
    int cand = findCandidate(a, size);
    if (isMajority(a, size, cand))
        cout << " " << cand << " ";
    else
        cout << "No Majority Element";
}
int main()
{
    int a[] = { 1, 3, 1, 1, 2 };
    int size = (sizeof(a)) / sizeof(a[0]);
    printMajority(a, size);
    return 0;
}
```

1. Loop through each element and maintains a count of majority element, and a majority index, *maj_index*
2. If the next element is same then increment the count if the next element is not same then decrement the count.
3. if the count reaches 0 then changes the maj_index to the current element and set the count again to 1.
4. Now again traverse through the array and find the count of majority element found.
5. If the count is greater than half the size of the array, print the element
6. Else print that there is no majority element

From <https://www.geeksforgeeks.org/majority-element/>

Time Complexity :- O(n)

Space Complexity :- O(1)

Q. Find Duplicates

Problem Statement
You are given an array ‘ARR’ of size ‘N’ containing each number between 1 and ‘N’ - 1 at least once. There is a single integer value that is present in the array twice. Your task is to find the duplicate integer value present in the array.
For example:
Consider ARR = [1, 2, 3, 4, 4], the duplicate integer value present in the array is 4. Hence, the answer is 4 in this case.

From <https://www.codingninjas.com/codestudio/problems/duplicate-in-array_893397>

Solution -

```cpp
#include<bits/stdc++.h>
using namespace std;
int FindDups(int *arr, int n){
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if(arr[i] == arr[j]){
                return j;
            }
        }
    }
    return {};
}
int main(){
    int arr[] = {1,2,2,3};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```cpp
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {
        vector<int> ans;
        int n=nums.size();
        for(auto x:nums){
            x= abs(x);
            if(nums[x-1]>0) nums[x-1] *= -1;
            else ans.push_back(x);
        }

        return ans;

    }
```

```
}
int main(){
    int arr[] = {1,2,2,3};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << FindDups(arr,n);
}
//Time Complexity - O(n^2)
//Space Complexity - O(1)
```

```
        return ans;
    }
};
```

Q. Move Zeros To Left

- **Problem Statement**
- **You are given an array 'ARR' of integers. Your task is to modify the array so that all the array elements having zero values get pushed to the left and all the array elements having non-zero value come after them while maintaining their relative order.**
- **For Example :**
- Consider the array { 1, 1, 0, 2, 0 }.
  For the given array the modified array should be {0,0,1,1,2} .
  Arrays { 0, 0, 1, 2, 1 } and { 0, 0, 2, 1, 1 } are not the correctly reorganized array even if they have all the zero values pushed to the left as in both the arrays the relative order of non-zero elements is not maintained.

- From <https://www.codingninjas.com/codestudio/problems/move-zeros-to-left_1094877>

Solution

```
#include<bits/stdc++.h>
using namespace std;
void moveZerosLeft(int *arr, int n){
    int count = n-1;
    for(int i=n-1; i>=0; i--){
        if(arr[i]!=0){
            swap(arr[i],arr[count]);
            count--;
        }
    }
}
int main(){
    int arr[] = {1,2,3,0,2,0};
    int n = sizeof(arr)/sizeof(arr[0]);
    for(int i=0;i<n;i++){
        cout << arr[i];
    }
    cout << endl;
    moveZerosLeft(arr,n);
    for(int i=0;i<n;i++){
        cout << arr[i];
    }
}
//Time Complexity - O(n)
// Space Complexity - O(1)
```