## Two Sum

Problem link:

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

<https://tutorialcup.com/leetcode-solutions/two-sum-leetcode-solution.htm>

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.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

**Input:** nums = [3,2,4], target = 6

**Output:** [1,2]

**Example 3:**

**Input:** nums = [3,3], target = 6

**Output:** [0,1]

**Constraints:**

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

C++

* class Solution {
* public:
* vector<int> twoSum(vector<int>& nums, int target) {
* int n = nums.size();
* for(int i = 0 ; i < n - 1 ; i++)
* for(int j = i + 1 ; j < n ; j++)
* {
* if(nums[i] + nums[j] == target)
* return {i,j};
* }
* return {};
* }
* };

#include <bits/stdc++.h>

using *namespace* std;

vector <*int*> targetSum(vector <*int*> &a , *int* &target)

{

*int* left = 0 , right = *int*(a.size()) - 1 , tempSum;

    while(left < right)

    {

        tempSum = a[left] + a[right];

        if(tempSum == target)

            return {left + 1 , right + 1};

        if(tempSum > target)

            right--;

        else

            left++;

    }

    return {-1 , -1};

}

*int* main()

{

    vector <*int*> a = {1 , 4 , 5 , 11 , 12};

*int* target = 9;

    for(*int* &x : targetSum(a , target))

        cout << x << " ";

    cout << '\n';

}

## 15. 3Sum

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.

**Example 1:**

**Input:** nums = [-1,0,1,2,-1,-4]

**Output:** [[-1,-1,2],[-1,0,1]]

**Explanation:**

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

The distinct triplets are [-1,0,1] and [-1,-1,2].

Notice that the order of the output and the order of the triplets does not matter.

**Example 2:**

**Input:** nums = [0,1,1]

**Output:** []

**Explanation:** The only possible triplet does not sum up to 0.

**Example 3:**

**Input:** nums = [0,0,0]

**Output:** [[0,0,0]]

**Explanation:** The only possible triplet sums up to 0.

**Constraints:**

* 3 <= nums.length <= 3000
* -105 <= nums[i] <= 105
* class Solution {
* public:
* vector<vector<int>> threeSum(vector<int>& nums) {
* int n=nums.size();
* set< vector<int> >s;
* //as no duplicate combination is allowed
* for(int i=0;i<n-2;i++){
* for(int j=i+1;j<n-1;j++){
* for(int k=j+1;k<n;k++){
* if(nums[i]+nums[j]+nums[k] == 0){
* vector<int> v(3);
* v[0]=nums[i];
* v[1]=nums[j];
* v[2]=nums[k];
* sort(v.begin(),v.end());
* s.insert(v);//insert in set
* }
* }
* }
* }
* vector<vector<int>>v;
* for(auto i:s)
* {
* v.push\_back(i);//copy vector to set
* }
* return v;
* }
* };

Time Limit Exceed.

### Two Pointer approach

The best method to get 3 SUM [LeetCode Solution](https://codewithgeeks.com/merge-2-sorted-linked-lists-leetcode/) would be using two pointers approach.

Here the first step would be to sort the given input array. We would also get rid of the extra space that we were using.

We know that a+b+c=0. If we keep ‘a’ constant we will get b+c=-a. We can now get b and c using the two pointer approach. Let i denote the present index of a so we will start from i+1 (low pointer) to search b and n-1(high pointer) to search for c where n is the size of nums array. If the sum of b+c is giving less than the required sum so we will just increase the low pointer one step towards right.

*class* Solution {

*public:*

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

        vector< vector<*int*> > Vec;//<<,>> are operators.Hence use space for '< <' (vector inside vector)

        if(nums.size() < 3) return Vec;//if there is only 2 numbers return Empty vector

        sort(nums.begin(), nums.end());//sort given vector to apply 2 pointers method

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

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

                //if i-1 and i th element are same-> if taken in loop may produce duplicate ans.So for same value only take it once

*int* low = i+1, high = nums.size()-1, sum = -nums[i];

                while(low < high){//search like binary search

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

                        Vec.push\_back({nums[i], nums[low], nums[high]});

                        //avoid duplicate value.Take them once

                        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) low++;

                    else high--;//nums[low] + nums[high] > sum

                  }

              }

          }

          return Vec;

      }

  };