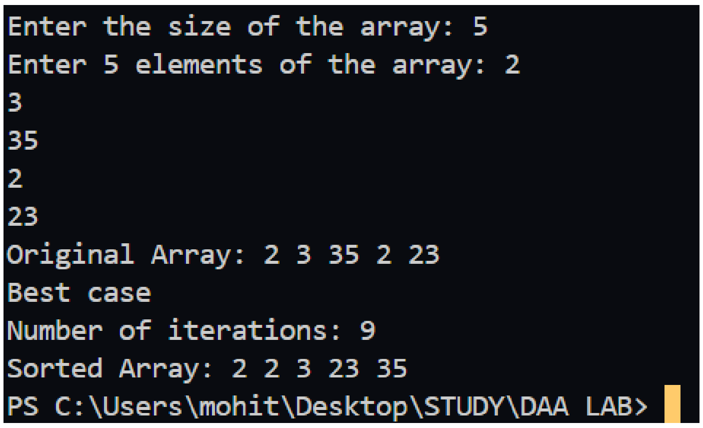
LAB 1

Bubble Sort

```cpp
#include <iostream>
using namespace std;

void printCount(int count) {
    cout << "Number of iterations: " << count << endl;
}

void bubbleSort(int arr[], int n) {
    int iterationCount = 0;
    bool swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            iterationCount++;
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            cout << "Best case" << endl;
            break;
        }
    }
    printCount(iterationCount);
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    int n;
    cout << "Enter the size of the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    cout << "Original Array: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}

```

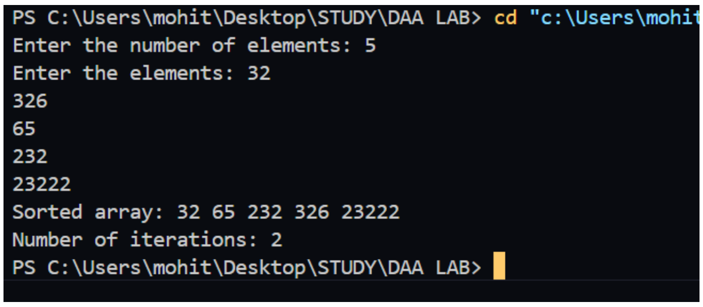


```
Enter the size of the array: 5
Enter 5 elements of the array: 2
3
35
2
23
Original Array: 2 3 35 2 23
Best case
Number of iterations: 9
Sorted Array: 2 2 3 23 35
PS C:\Users\mohit\Desktop\STUDY\DAA LAB>
```

	LOOPS	COMP
BEST	1	4
WORSE	4	10
AVG	3	9

Insertion Sort

```cpp
#include <iostream>
#include <vector>

using namespace std;

void insertionSort(vector<int> & arr , int& iterations) {
  int n = arr.size();
  for (int i = 1; i < n; ++i) {
    int key = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      --j;
      ++iterations;
    }
    arr[j + 1] = key;
  }
}

int main(){
  int n;
  cout << "Enter the number of elements: ";
  cin >> n;

  vector<int> arr(n);
  cout << "Enter the elements: ";
  for (int i = 0; i < n; ++i) {
    cin >> arr[i];
  }

  int iterations = 0;
  insertionSort(arr, iterations);

  cout << "Sorted array: ";
  for (int i = 0; i < arr.size(); ++i) {
    cout << arr[i] << " ";
  }
  cout << endl;

  cout << "Number of iterations: " << iterations << endl;
  return 0;
}

```



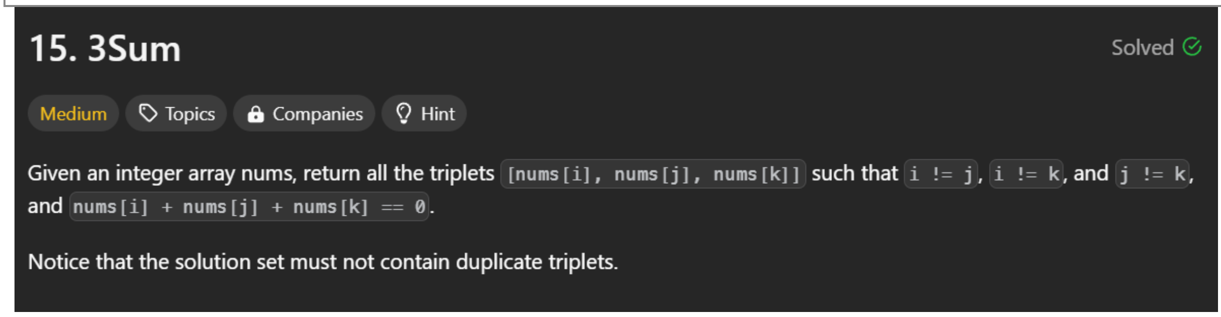
```
PS C:\Users\mohit\Desktop\STUDY\DAA LAB> cd "c:\Users\mohit
Enter the number of elements: 5
Enter the elements: 32
326
65
232
23222
Sorted array: 32 65 232 326 23222
Number of iterations: 2
PS C:\Users\mohit\Desktop\STUDY\DAA LAB> 
```

	LOOPS	COMP
BEST	10	10
WORSE	10	10
AVG	10	10

LEETCODE PROBLEMS1.

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 retum the answer in any order.

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

    }
};

```



15. 3Sum

Solved ✓

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.

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int> &nums) {
        sort(begin(nums), end(nums));
        vector<vector<int>> result;
        for (int i = size(nums) - 1; i >= 2; --i) {
            if (i + 1 < size(nums) && nums[i] == nums[i + 1]) {
                continue;
            }
            const auto& target = -nums[i];
            int left = 0, right = i - 1;
            while (left < right) {
                if (nums[left] + nums[right] < target) {
                    ++left;
                } else if (nums[left] + nums[right] > target) {
                    --right;
                } else {
                    result.push_back({nums[left], nums[right], nums[i]});
                    ++left; --right;
                    while (left < right && nums[left] == nums[left - 1]) {
                        ++left;
                    }
                    while (left < right && nums[right] == nums[right + 1]) {
                        --right;
                    }
```