

Q1. 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: Input: `nums = [2,7,11,15]`, `target = 9` Output `[0,1]`

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

SOLUTION:

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

        vector<int> res;

        // Create a pair of original indices and values
        vector<pair<int, int>> numPairs;

        for (int i = 0; i < nums.size(); i++) {
            numPairs.push_back({nums[i], i});
        }

        // Sort the pairs based on their values
        sort(numPairs.begin(), numPairs.end());

        int left = 0;

        int right = numPairs.size() - 1;
```

```

while (left < right) {

    int sum = numPairs[left].first + numPairs[right].first;

    if (sum == target) {

        res.push_back(numPairs[left].second);

        res.push_back(numPairs[right].second);

        break;

    } else if (sum < target) {

        left++;

    } else {

        right--;

    }

}

return res;

}

};

```

Time complexity: $O(n)$

💡 ****Q2.**** 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.

Example :

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

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

Explanation: Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores)[

SOLUTION:

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {

        nums.erase(remove(nums.begin(), nums.end(), val), nums.end());

        return nums.size();

    }
};
```

Time complexity: O(n)

💡 **Q3.** Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1: Input: `nums = [1,3,5,6]`, `target = 5`

Output: 2

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        auto it = lower_bound(nums.begin(), nums.end(), target);
        return (it - nums.begin());
    }
};
```

Time complexity: $O(\log n)$

💡 **Q4.** You are given a large integer represented as an integer array `digits`, where each `digits[i]` is the *i*th digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Example 1: Input: `digits = [1,2,3]` Output: `[1,2,4]`

Explanation: The array represents the integer 123.

Incrementing by one gives $123 + 1 = 124$. Thus, the result should be `[1,2,4]`.

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int s = digits.size();
        int n = digits.size();
        vector<int> result;
        int carry = 1;

        for (int i = n - 1; i >= 0; i--) {
            int sum = digits[i] + carry;

            result.push_back(sum % 10);

            carry = sum / 10;
        }

        if (carry > 0) {
            result.push_back(carry);
        }

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

        return result;
    }
};
```

💡 **Q5.** 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`.

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        nums1.erase(remove(nums1.begin(), nums1.end(), 0), nums1.end());

        nums2.erase(remove(nums2.begin(), nums2.end(), 0), nums2.end());

        vector<int> merged(nums1.size() + nums2.size());

        std::merge(nums1.begin(), nums1.end(), nums2.begin(), nums2.end(),
merged.begin());

        nums1 = merged;
    }
};
```

💡 **Q6.** 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`

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        set<int> unique;
        for(int i=0;i<nums.size();i++)
            unique.insert(nums[i]);

        if(unique.size()==nums.size())
            return false;
        else
            return true;
    }
};
```

💡 **Q7.** Given an integer array `nums`, move all 0's to the end of it while maintaining the relative order of the nonzero elements.

Note that you must do this in-place without making a copy of the array.

Example 1: Input: `nums = [0,1,0,3,12]` Output: `[1,3,12,0,0]`

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {

        int s= nums.size();

        nums.erase(remove(nums.begin(),nums.end(),0),nums.end());

        int w=nums.size();

        int n=s-w;

        for(int i=0;i<n;i++){
            nums.push_back(0);
        }

    }
};
```


💡 **Q8.** You have a set of integers s , which originally contains all the numbers from 1 to n . Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array `nums` representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

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

```
class Solution {
public:
    vector<int> findErrorNums(vector<int>& nums) {
        vector<int> freq(nums.size(), 0);

        int miss, hit;

        for(int i=0; i<nums.size(); i++) freq[nums[i]-1]++;

        for(int i=0; i<nums.size(); i++){
            if(freq[i] == 0) miss = i+1;

            if(freq[i] == 2) hit = i+1;
        }

        nums.clear();

        nums.push_back(hit);

        nums.push_back(miss);

        return nums;
    }
};
```