1. **Question Number and Name:**
2. [217. Contains Duplicate](https://leetcode.com/problems/contains-duplicate/)
3. **Topics:**

[Array](https://leetcode.com/tag/array/)[Hash Table](https://leetcode.com/tag/hash-table/) [Sorting](https://leetcode.com/tag/sorting/)

1. **Problem Statement:**

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

**Example 3:**

**Input:** nums = [1,1,1,3,3,4,3,2,4,2]

**Output:** true

**Constraints:**

* 1 <= nums.length <= 105
* -109 <= nums[i] <= 109

1. **Code:**
2. class *Solution* {
3. public:
4. bool containsDuplicate(vector<int>*&* nums) {
5. sort(nums.begin(),nums.end());
6. bool dupli\_exist=false;
7. for(int i=0;i<nums.size()-1;i++)
8. {
9. if(nums[i]==nums[i+1])
10. {
11. dupli\_exist=true;
12. break;
13. }
14. }
15. return dupli\_exist;
16. }
17. };
    1. **Notes:**

* One solution is O(N2) solution by running two for loops over each other.
* The other solution is sorting the array first and then running a for loop iterating over the array in O(N) time complexity. This is the solution I have used above and It has O(nlogn) time complexity and O(1) space complexity since it doesnot require additional space if you discount the space taken by the sorting algorithm
* An O(N) time complexity solution would by using a hashmap and checking if an element already exists before inserting it in the map, we use the std::map.count() function to find if the key already exists, if the count is >0 it means that the key already exists in the map and there are duplicates in the array. The downside of this method is that it also require O(N) space complexity but this is the best in terms of space and time complexity as we can get.

1. **Question Number and Name:**

## [242. Valid Anagram](https://leetcode.com/problems/valid-anagram/)

1. **Topics:**

[Hash Table](https://leetcode.com/tag/hash-table/)[String](https://leetcode.com/tag/string/) [Sorting](https://leetcode.com/tag/sorting/)

1. **Problem Statement:**

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Example 2:**

**Input:** s = "rat", t = "car"

**Output:** false

**Constraints:**

* 1 <= s.length, t.length <= 5 \* 104
* s and t consist of lowercase English letters.

1. **Code:**
2. class *Solution* {
3. public:
4. bool isAnagram(string s, string t) {
5. sort(s.begin(), s.end());
6. sort(t.begin(), t.end());
7. if(s==t)
8. return true;
9. else
10. return false;
11. }
12. };
    1. **Notes:**

* You can do this question by using hashmaps, in that case the time and space complexity will by O(S+T).
* Or you can sort the strings and check if both of them are equal or not after sorting both of them. This solution has O(nlogn) time complexity and O(1) space complexity since most interviewers assume that built-in library functions don’t consume extra memory while sorting.

1. **Question Number and Name:**

## [1. Two Sum](https://leetcode.com/problems/two-sum/)

1. **Topics:**

Array, hashtable

1. **Problem Statement:**

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.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

1. **Code:**
2. class Solution {
3. public:
4. vector<int> twoSum(vector<int>*&* nums, int target) {
5. int first=0,second=0;
6. for (int i=0; i<nums.size();i++)
7. {
8. for(int j=i+1;j<nums.size();j++)
9. {
10. if(nums[i]+nums[j]  ==  target)
11. {
12. first=i,second=j;
13. cout<<first<<endl;
14. cout<<second<<endl<<endl;
15. break;
16. }
17. }
18. }
19. vector<int> ans;
20. if(first==0 && second==0)
21. {
22. return ans;
23. }
24. ans.push\_back(first);
25. ans.push\_back(second);
26. return ans;
27. }
28. };
    1. **Notes:**
    * Pretty simple way of doing it in O(N2), Doesn’t really require any explanation.
29. **Question Number and Name:**

## [49. Group Anagrams](https://leetcode.com/problems/group-anagrams/)

1. **Topics:**

Array, Hash Table, String, Sorting

1. **Problem Statement:**

Medium

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Constraints:**

* 1 <= strs.length <= 104
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

1. **Code:**

class Solution {

public:

    vector<vector<string>> groupAnagrams(vector<string>*&* strs) {

        vector<vector<string>> ans;

        unordered\_map<string,vector<string>>holder;

        for(int i=0;i<strs.size();i++)

        {

            string temp=strs[i];

            sort(temp.begin(),temp.end());

            holder[temp].push\_back(strs[i]);

*// cout<<holder[temp]<<endl;*

        }

        for(auto i:holder)

        {

            ans.push\_back(i.second);

        }

        return ans;

    }

};

1. **Notes:**

* So basically we just traverse the array once, on each iteration we store the value of the array in a temp variable and sort it, now that sorted array will be used as a key in a hashmap. And of every string that matches the key will be stored against it the hashmap (classic definition of anagram)
* ‘ant’ and ‘tan’ when sorted become ‘ant’.

Both of these strings will be stored against the keys in the hashmap like

ant: [‘ant’,’tan’]

* Now we traverse the hashmap once again and for every key we push the corresponding value pairs in a vector of vector of strings vector <vector<string>>
* In the end we return the ans which contains the vectors of strings as our solution to the problem.

1. **Question Number and Name:**

## [347. Top K Frequent Elements](https://leetcode.com/problems/top-k-frequent-elements/)

1. **Topics:**

Array, hashtable, divide and conquer, sorting, Heap (Priority Queue), Bucket Sort, Counting, Quickselect

1. **Problem Statement:**

Medium

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

**Example 1:**

**Input:** nums = [1,1,1,2,2,3], k = 2

**Output:** [1,2]

**Example 2:**

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

**Output:** [1]

**Constraints:**

* 1 <= nums.length <= 105
* -104 <= nums[i] <= 104
* k is in the range [1, the number of unique elements in the array].
* It is **guaranteed** that the answer is **unique**.

**Follow up:** Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

1. **Code:**

class Solution {

public:

    vector<int> topKFrequent(vector<int>*&* nums, int k) {

        vector<int> ans;

        vector<vector<int>> freq (nums.size()+1);

        unordered\_map<int,int> freq\_holder;

        int count=0;

*//O(N) time complexity*

        for(int i=0;i<nums.size();i++)

        {

            freq\_holder[nums[i]]++;

        }

*//O(N) time complexity*

        for(auto i:freq\_holder)

        {

            freq[i.second].push\_back(i.first);

        }

*//2D Array traversal O(nums.size()+1 x freq[i].size) = O (N)*

        for(int i=nums.size();i>=0;i--)

        {

               if(freq[i].size()!=0)

               {

                   for(int j=0;j<freq[i].size();j++)

                   {

                       ans.push\_back(freq[i][j]);

                       cout<<freq[i][j]<<endl;

                       count++;

                       if(count==k)

                       {

                           return ans;

                       }

                   }

               }

        }

*// overall time complexity O(N) + O(N) + O(N) = O (N)*

        return ans;

    }

};

1. **Notes:**

* We keep an int vector for answers
* We create a vector of vectors for keeping how many elements have which kind of frequency
* We keep an unordered map to find out the frequency of all elements.
* Firstly we put all values in the unordered hashmap to find out their frequencies in O(N) time

unordered\_map<int,int> freq\_holder;

*//O(N) time complexity*

        for(int i=0;i<nums.size();i++)

        {

            freq\_holder[nums[i]]++;

        }

* Now we put all the elements in the vector of vectors called “freq” corresponding to their frequency

vector<vector<int>> freq (nums.size()+1);

for(auto i:freq\_holder)

        {

            freq[i.second].push\_back(i.first);

        }

* Now we traverse through the vector of vectors “freq” from the end (since we have to find top-k elements) and keep pushing values in our “ans vector”, we also keep a count of the elements inserted in the answer vector. After every insertion we check if the count has become equal to K or not. If the condition of k==count is fulfilled we return our answer vector.

1. **Question Number and Name:**

## [238. Product of Array Except Self](https://leetcode.com/problems/product-of-array-except-self/)

1. **Topics:**

Array, Prefix Sum

1. **Problem Statement:**

Medium

Given an integer array nums, return *an array* answer *such that* answer[i] *is equal to the product of all the elements of* nums *except* nums[i].

The product of any prefix or suffix of nums is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

**Example 1:**

**Input:** nums = [1,2,3,4]

**Output:** [24,12,8,6]

**Example 2:**

**Input:** nums = [-1,1,0,-3,3]

**Output:** [0,0,9,0,0]

**Constraints:**

* 2 <= nums.length <= 105
* -30 <= nums[i] <= 30
* The product of any prefix or suffix of nums is **guaranteed** to fit in a **32-bit** integer.

**Follow up:** Can you solve the problem in O(1) extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

1. **Code:**

class Solution {

public:

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

        vector<int> ans(nums.size(),1);

        for(int i=1;i<nums.size();i++)

        {

            ans[i]=ans[i-1]\*nums[i-1];

*// cout<<"Prefix is "<<ans[i]<<endl;*

        }

        for(int i=nums.size()-1, suffix=1;i>=0;i--)

        {

            ans[i]\*=suffix;

*// cout<<suffix<<endl;*

            suffix\*=nums[i];

        }

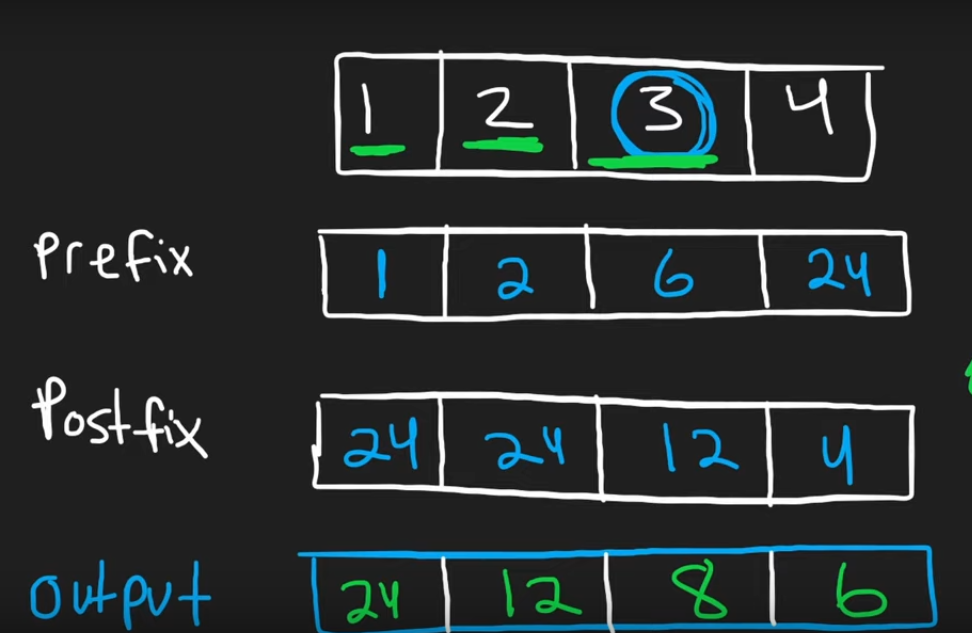
        return ans;

    }

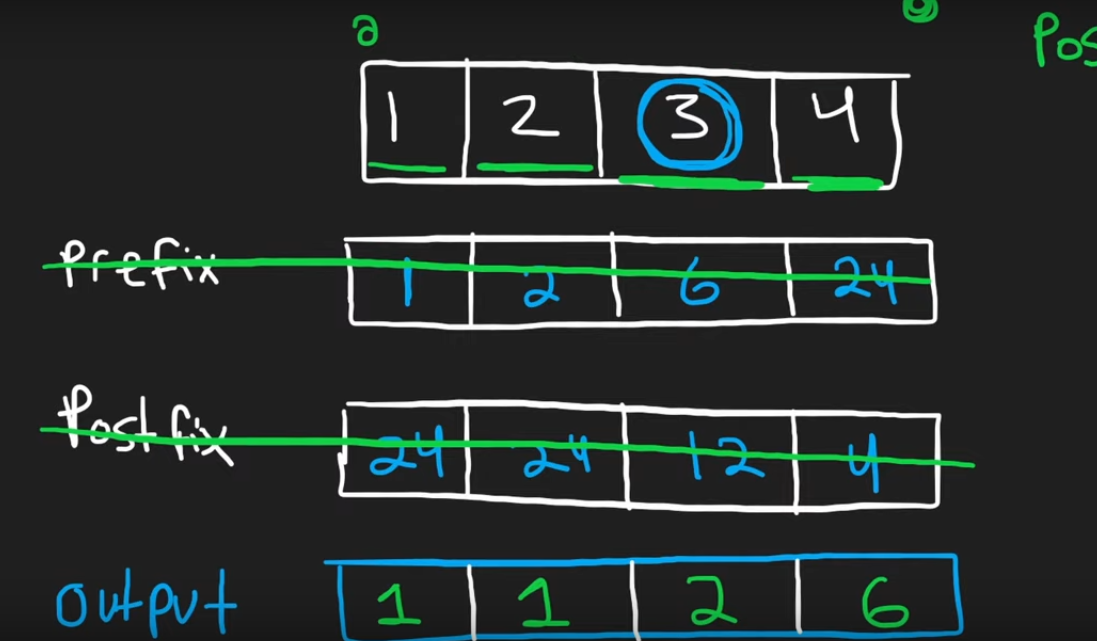
};

1. **Notes:**

* We use the prefix sum method in this question
* The core idea behind this solution is as follows, we create two additional vectors for holding the prefix and postfix sum of each element and then multiply the (n-1) prefix value and (n+1) postfix value to get the product of the array except itself.
* For edge cases i.e. the first element’s prefix sum is taken to be 1 and the last element’s postfix sum is taken to be 1.
* It can be demonstrated well via this picture:



* BUT we can do the same without creating those 2 additional arrays too!
* We are gonna create only 1 answer vector and first calculate the arrays prerfix sum into it like this



* Now we are gonna run another loop from the end of the given vector and keep its postfix multiple in a variable ‘postfix’=1, and keep multiplying it till i>=0.
* When both loops have ran, we will have the ‘product of the array except itself’ in the answer array.

1. **Question Number and Name:**

## [36. Valid Sudoku](https://leetcode.com/problems/valid-sudoku/)

1. **Topics:**

Array, Hash Table, Matrix

1. **Problem Statement:**

Medium

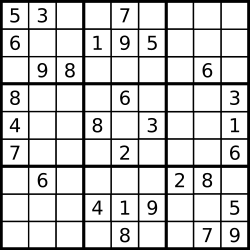
Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

**Note:**

* A Sudoku board (partially filled) could be valid but is not necessarily solvable.
* Only the filled cells need to be validated according to the mentioned rules.

**Example 1:**



**Input:** board =

[["5","3",".",".","7",".",".",".","."]

,["6",".",".","1","9","5",".",".","."]

,[".","9","8",".",".",".",".","6","."]

,["8",".",".",".","6",".",".",".","3"]

,["4",".",".","8",".","3",".",".","1"]

,["7",".",".",".","2",".",".",".","6"]

,[".","6",".",".",".",".","2","8","."]

,[".",".",".","4","1","9",".",".","5"]

,[".",".",".",".","8",".",".","7","9"]]

**Output:** true

**Example 2:**

**Input:** board =

[["8","3",".",".","7",".",".",".","."]

,["6",".",".","1","9","5",".",".","."]

,[".","9","8",".",".",".",".","6","."]

,["8",".",".",".","6",".",".",".","3"]

,["4",".",".","8",".","3",".",".","1"]

,["7",".",".",".","2",".",".",".","6"]

,[".","6",".",".",".",".","2","8","."]

,[".",".",".","4","1","9",".",".","5"]

,[".",".",".",".","8",".",".","7","9"]]

**Output:** false

**Explanation:** Same as Example 1, except with the **5** in the top left corner being modified to **8**. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

**Constraints:**

* board.length == 9
* board[i].length == 9
* board[i][j] is a digit 1-9 or '.'.

1. **Code:**

class *Solution* {

public:

    bool isValidSudoku(vector<vector<char>>*&* board) {

    vector<set<int>> horizontal(9),vertical(9), box(9);

     for(int i=0; i<board.size();i++)

     {

         for(int j=0;j<board[i].size();j++)

         {

            if (board[i][j]=='.')

                continue;

             int curr=board[i][j]-'0';

             if (horizontal[i].count(curr) || vertical[j].count(curr) || box[(i/3)\*3+j/3].count(curr))

             {

                 for(int i=0;i<horizontal.size();i++)

                    {

                        for(auto j:horizontal[i])

                            cout<<j<<" ";

                        cout<<endl;

                    }

                return false;

             }

             horizontal[i].insert(curr);

             vertical[j].insert(curr);

             box[(i/3)\*3+j/3].insert(curr);

         }

     }

    return true;

    }

};

1. **Notes:**

* Ngl this was a pretty tough question for me
* The conditions are pretty straight forward
  + Each row must contain the digits 1-9 without repetition.
  + Each column must contain the digits 1-9 without repetition.
  + Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.
* For each condition we will have a vectors of sets i.e. A vector of 9 sets for checking the horizontal condition, a vector of 9 sets for checking the vertical condition and a vector of 9 sets for checking the box condition.
* Now we will run two nested for-loops that will

int curr=board[i][j]-'0';

will store convert the value of the soduku board char values into ints

* Now we will check for each condition via this if statement and if a value is found before then we will return false

if (horizontal[i].count(curr) || vertical[j].count(curr) ||

box[(i/3)\*3+j/3].count(curr))

                return false;

Let’s disect this code one by one

horizontal[i].count(curr)

will check if the value previously exists in the horizontal row

vertical[j].count(curr)

will check if the value previously exists in the vertical column

box[(i/3)\*3+j/3].count(curr)

will check if the value previously exists in the box

IF any of the above given conditions are true we will return false since it will violate the conditions of the soduku.

* The condition to check the presence of the element in box is very interesting and it was explained in a very simple and easy manner in leetcode discussion board. You can further. I am copy pasting the whole article

The problem is to come up with an equation to map (n\*n) coordinates to (n) numbers so we can use it as an index into the blocks array.

Our board would look like this:

(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7) (0,8)  
(1,0) (1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8)  
(2,0) (2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) (2,8)  
(3,0) (3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (3,7) (3,8)  
(4,0) (4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7) (4,8)  
(5,0) (5,1) (5,2) (5,3) (5,4) (5,5) (5,6) (5,7) (5,8)  
(6,0) (6,1) (6,2) (6,3) (6,4) (6,5) (6,6) (6,7) (6,8)  
(7,0) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7) (7,8)  
(8,0) (8,1) (8,2) (8,3) (8,4) (8,5) (8,6) (8,7) (8,8)

Let's divide this into 3x3 boxes, and label the boxes like this:

0 1 2  
3 4 5  
6 7 8

Thus  
(0,0) (0,1) (0, 2)  
(1,0) (1,1) (1, 2)  
(2,0) (2,1) (2, 2)  
will map to box 0.

And  
(0,3) (0,4) (0,5)  
(1,3) (1,4) (1,5)  
(2,3) (2,4) (2,5)  
will map to box 1.

And so on.

We can see that for any coordinate (r, c), the column index (c), which ranges from 0-8 should contribute either (0, 1, or 2) to the boxIndex. So we can write this as c/3.  
This is because of integer division:  
{0, 1, 2} / 3 = 0  
{3, 4, 5} / 3 = 1  
{6, 7, 8} / 3 = 2

Also, for any coordinate (r, c), if the column already contributes (0, 1, or 2) to the boxIndex (which has a maximum of 8), then the row index (r) needs to contribute either (0, 3, or 6) to the boxIndex, since this gives all possible numbers from 0 - 8.  
e.g.  
0 + {0, 1, 2} = {0, 1, 2}  
3 + {0, 1, 2} = {3, 4, 5}  
6 + {0, 1, 2} = {6, 7, 8}

Well how do we map {0,1,2,3,4,5,6,7,8} to {0,3,6} ?  
We can use integer division again, but multiply the result by 3  
({0, 1, 2} / 3) \* 3 = (0) \* 3 = 0  
({3, 4, 5} / 3) \* 3 = (1) \* 3 = 3  
({6, 7, 8} / 3) \* 3 = (2) \* 3 = 6

So we have boxIndex = c/3 + r/3\*3

* Now if all of the given conditions are false we will insert the elements in the horizontal, vertical and box set to check in the future and loop will continue.

If the whole array is traversed without any return in false condition then we will return true in the end.

1. **Question Number and Name:**
2. **Problem Statement**
3. **Topics:**
4. **Code:**
5. **Notes:**