**QUESTION # 1**

**7. Reverse Integer** ***Medium***

Given a signed 32-bit integer x, return x*with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.

**Assume the environment does not allow you to store 64-bit integers (signed or unsigned).**

**Example 1: Input:** x = 123 **Output:** 321

**Example 2: Input:** x = -123 **Output:** -321

**Example 3: Input:** x = 120 **Output:** 21

**CODE**

class Solution {

public:

    int reverse(int x) {

        long long  reversed=0;

        while(x!=0){

            int digit=x%10;

            reversed=reversed\*10 + digit;

            x=x/10;

        }

        if (reversed > INT\_MAX || reversed < INT\_MIN) {

                return 0;

            }

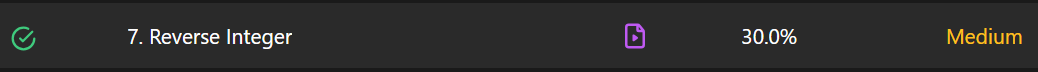
        return int(reversed);

    }

};

**EXPLANATION OF CODE**

1. Here I have reverse a integer by taking last digit of number by modulus operator.
2. Then add it in reversed number which was initially zero.
3. Then permanently take out last digit by divide operator.
4. Lastly I check a condition that a reversed number shouldn’t be greater or lesser than Integer range.
5. Now return the answer by typecasting it into int.



**QUESTION # 2**

**344. Reverse String** ***Easy***

Write a function that reverses a string. The input string is given as an array of characters s.

You must do this by modifying the input array [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Example 1: Input:** s = ["h","e","l","l","o"] **Output:** ["o","l","l","e","h"]

**Example 2: Input:** s = ["H","a","n","n","a","h"] **Output:** ["h","a","n","n","a","H"]

**CODE**

class Solution {

public:

    void reverseString(vector<char>& s) {

        int i=0;

        int j=s.size()-1;

        while(i<j && i!=j){

            swap(s[i],s[j]);

            i++;

            j--;

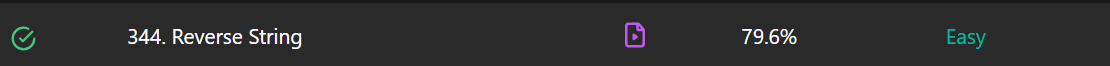
        }

    }

};

**EXPLANATION OF CODE**

1. Here I have reverse a string by pointing at zero index (i) and last index (j).
2. Then I swap those.
3. Then I increment (i) and decrement (j)
4. Loop continues till (i) is greater then (j)



**QUESTION # 3**

**509. Fibonacci Number** ***Easy***

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is, F(0) = 0, F(1) = 1 F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

**Example 1: Input:** n = 2 **Output:** 1 **Explanation:** F(2) = F(1) + F(0) = 1 + 0 = 1.

**Example 2: Input:** n = 3 **Output:** 2 **Explanation:** F(3) = F(2) + F(1) = 1 + 1 = 2.

**Example 3: Input:** n = 4 **Output:** 3 **Explanation:** F(4) = F(3) + F(2) = 2 + 1 = 3.

**CODE**

class Solution {

public:

    int fib(int n) {

        if (n==0){

            return 0;

        }

        if (n==1){

            return 1;

        }

        return fib(n-1) + fib(n-2);

    }

};

**EXPLANATION OF CODE**

1. Here I have used a recursion to get a fibonacci series.
2. The function is recursively called when a base case that is one is reached.
3. The base case zero only executes when the number was initially zero.



**QUESTION # 4**

**442. Find All Duplicates in an Array** ***Medium***

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **at most** **twice**, return *an array of all the integers that appears****twice***.

You must write an algorithm that runs in O(n) time and uses only *constant* auxiliary space, excluding the space needed to store the output

**Example 1: Input:** nums = [4,3,2,7,8,2,3,1] **Output:** [2,3]

**Example 2: Input:** nums = [1,1,2] **Output:** [1]

**Example 3: Input:** nums = [1] **Output:** []

**CODE**

class Solution {

public:

    vector<int> findDuplicates(vector<int>& nums) {

        unordered\_set<int> seen;

        vector<int> result;

        for (int i:nums){

            if(seen.count(i)){

                result.push\_back(i);

            }

            else{

                seen.insert(i);

            }

        }

        return result;

    }

};

**EXPLANATION OF CODE**

1. Here I have created an unordered set that will store unique elements and a vector to store elements that are in unordered set.
2. Loop continues and when a element is found in set it is pushed to vector since it was a duplicate
3. If element not found in set then it is pushed into set
4. In the end I return the vetor result



**QUESTION # 5**

**349. Intersection of Two Arrays** ***Easy***

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

**Example 1: Input:** nums1 = [1,2,2,1], nums2 = [2,2] **Output:** [2]

**Example 2: Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4] **Output:** [9,4]

**Explanation:** [4,9] is also accepted.

**CODE**

class Solution {

public:

    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {

        unordered\_set<int> seen(nums1.begin(),nums1.end());

        vector<int> result;

        for (int i:nums2){

            if (seen.find(i) != seen.end()){

                result.push\_back(i);

                seen.erase(i);

            }

        }

        return result;

    }

};

**EXPLANATION OF CODE**

1. First I created unordered set and copy first vetor elements into it.
2. Then I created a new vector named result.
3. The loop runs for the second vector
   1. If element found in set then it is added to result and erase from the set
4. Lastly I return a result vector.

This way I get the intersection of two arrays.



**QUESTION # 6**

**88. Merge Sorted Array** ***Easy***

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.

**Example 2: Input:** nums1 = [1], m = 1, nums2 = [], n = 0 **Output:** [1] **Explanation:** The arrays we are merging are [1] and []. The result of the merge is [1].

**Example 3: Input:** nums1 = [0], m = 0, nums2 = [1], n = 1 **Output:** [1] **Explanation:** The arrays we are merging are [] and [1]. The result of the merge is [1]. Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

**CODE**

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        int p1 = m - 1;

        int p2 = n - 1;

        int p = m + n - 1;

        while(p1>=0 && p2>=0){

            if (nums1[p1]>nums2[p2]){

                nums1[p] = nums1[p1];

                p1--;

            }

            else{

                nums1[p] = nums2[p2];

                p2--;

            }

            p--;

        }

        while (p2 >= 0) {

        nums1[p] = nums2[p2];

        p2--;

        p--;

        }

    }

};

**EXPLANATION OF CODE**

1. I used three pointers: p1 for last element of nums1, p2 for last of nums2, and p for the last position in nums1.
2. Then I compared elements from the end and placed the bigger one at the end of nums1.
3. I decreased the pointers after placing values.
4. After that, if any elements were left in nums2, I copied them to nums1.
5. Now nums1 contains the fully merged sorted array.



**QUESTION # 7**

**189. Rotate Array** ***Medium***

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

**Example 1: Input:** nums = [1,2,3,4,5,6,7], k = 3 **Output:** [5,6,7,1,2,3,4]

**Explanation:** rotate 1 steps to the right: [7,1,2,3,4,5,6] rotate 2 steps to the right: [6,7,1,2,3,4,5] rotate 3 steps to the right: [5,6,7,1,2,3,4]

**Example 2: Input:** nums = [-1,-100,3,99], k = 2 **Output:** [3,99,-1,-100]

**Explanation:** rotate 1 steps to the right: [99,-1,-100,3]

rotate 2 steps to the right: [3,99,-1,-100]

**CODE**

class Solution {

public:

    void rotate(vector<int>& nums, int k) {

        int n=nums.size();

        vector<int> rotated(n);

        for (int i=0;i<n;i++){

            rotated[(i+k)%n]=nums[i];

        }

        nums=rotated;

    }

};

**EXPLANATION OF CODE**

1. I took the size of the original array nums and created a new array rotated of the same size.
2. Then I used a loop to shift each element to its new position by using the formula (i + k) % n.
3. This formula makes sure the index wraps around when it reaches the end.
4. After rotating all elements, I copied the rotated array back to nums.
5. This way, the array is rotated to the right by k steps.



**QUESTION # 8**

**1752. Check if Array Is Sorted and Rotated** ***Easy***

Given an array nums, return true*if the array was originally sorted in non-decreasing order, then rotated****some****number of positions (including zero)*. Otherwise, return false.

There may be **duplicates** in the original array.

**Note:** An array A rotated by x positions results in an array B of the same length such that B[i] == A[(i+x) % A.length] for every valid index i.

**Example 1: Input:** nums = [3,4,5,1,2] **Output:** true **Explanation:** [1,2,3,4,5] is the original sorted array. You can rotate the array by x = 3 positions to begin on the element of value 3: [3,4,5,1,2].

**Example 2: Input:** nums = [2,1,3,4] **Output:** false **Explanation:** There is no sorted array once rotated that can make nums.

**Example 3: Input:** nums = [1,2,3] **Output:** true **Explanation:** [1,2,3] is the original sorted array. You can rotate the array by x = 0 positions (i.e. no rotation) to make nums.

**CODE**

class Solution {

public:

    bool check(vector<int>& nums) {

        int count=0;

        int n=nums.size();

        for (int i=0;i<n-1;i++){

            if (nums[i]>nums[i+1]){

                count++;

            }

        }

        if (nums[0]<nums[n-1]){

            count++;

        }

        return count<=1;

    }

};

**EXPLANATION OF CODE**

1. I initialized a count variable to track how many times the order breaks in the array.
2. Then I looped through the array and checked where nums[i] > nums[i+1]. Each such case increases the count.
3. After the loop, I also checked the last and first elements to handle the circular rotation condition.
4. If nums[0] < nums[n-1], I increased the count again.
5. Finally, I returned true only if the count is less than or equal to 1, meaning the array can be sorted with one rotation.



**QUESTION # 9**

**125. Valid Palindrome** ***Easy***

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true*if it is a****palindrome****, or*false*otherwise*.

**Example 1: Input:** s = "A man, a plan, a canal: Panama" **Output:** true **Explanation:** "amanaplanacanalpanama" is a palindrome.

**Example 2: Input:** s = "race a car **Output:** false **Explanation:** "raceacar" is not a palindrome.

**Example 3: Input:** s = " " **Output:** true **Explanation:** s is an empty string "" after removing non-alphanumeric characters.

Since an empty string reads the same forward and backward, it is a palindrome.

**CODE**

class Solution {

private:

    char IsLower(char ch){

        if (ch>='A' && ch<='Z'){

            return ch+32;

        }

        return ch;

    }

public:

    bool isPalindrome(string s) {

        string newstr="";

        for (int i=0;i<s.length();i++){

            if ((s[i]>='A' && s[i]<='Z') || (s[i]>='a' && s[i]<='z') || s[i]>='0' && s[i]<='9'){

                newstr.push\_back(IsLower(s[i]));

            }

        }

        int start=0;

        int end=newstr.length()-1;

        while(start<end){

            if (newstr[start]==newstr[end]){

                start++;

                end--;

            }

            else{

                return false;

            }

        }

    return true;

    }

};

**EXPLANATION OF CODE**

1. I created a helper function IsLower to convert uppercase characters to lowercase.
2. In the main function, I made an empty string newstr to store only valid characters (letters and digits) in lowercase.
3. I looped through the original string and added characters to newstr only if they were alphanumeric, converting uppercase to lowercase.
4. Then I used two pointers: start from the beginning and end from the end of newstr.
5. I checked if characters at both pointers matched. If not, I returned false.
6. If the loop completed without mismatches, I returned true, meaning the string is a valid palindrome.



**QUESTION # 10**

**34. Find First and Last Position of Element in Sorted Array** ***Medium***

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

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

**Example 1: Input:** nums = [5,7,7,8,8,10], target = 8 **Output:** [3,4]

**Example 2: Input:** nums = [5,7,7,8,8,10], target = 6 **Output:** [-1,-1]

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

**CODE**

class Solution {

public:

    vector<int> searchRange(vector<int>& nums, int target) {

        vector<int> result(2);

        int start=0;

        int end=nums.size()-1;

        int ans=-1;

        while(start<=end){

            int mid = start + (end-start)/2;

            if (nums[mid]==target){

                ans=mid;

                end=mid-1;

            }

            else if (nums[mid]<target){

                start=mid+1;

            }

            else{

                end=mid-1;

            }

        }

        result[0]=ans;

        start=0;

        end=nums.size()-1;

        ans=-1;

        while(start<=end){

            int mid = start + (end-start)/2;

            if (nums[mid]==target){

                ans=mid;

                start=mid+1;

            }

            else if (nums[mid]<target){

                start=mid+1;

            }

            else{

                end=mid-1;

            }

        }

        result[1]=ans;

        return result;

    }

};

**EXPLANATION OF CODE**

1. I created a result vector of size 2 to store the first and last positions of the target.
2. First, I used binary search to find the **first occurrence** of the target by moving end = mid - 1 when I found the target.
3. If target matched, I stored its index in ans and updated result[0] after the loop.
4. Then I reset start, end, and ans to search again.
5. This time I looked for the **last occurrence** by moving start = mid + 1 when I found the target.
6. After the second binary search, I stored the last position in result[1].
7. Finally, I returned the result containing both starting and ending positions of the target.



**QUESTION # 11**

**58. Length of Last Word** ***Easy***

Given a string s consisting of words and spaces, return *the length of the****last****word in the string.*

A **word** is a maximal substring consisting of non-space characters only.

**Example 1: Input:** s = "Hello World" **Output:** 5 **Explanation:** The last word is "World" with length 5.

**Example 2: Input:** s = " fly me to the moon " **Output:** 4 **Explanation:** The last word is "moon" with length 4.

**Example 3: Input:** s = "luffy is still joyboy" **Output:** 6 **Explanation:** The last word is "joyboy" with length 6.

**CODE**

class Solution {

public:

    int lengthOfLastWord(string s) {

        int i=s.size() - 1;

        int length=0;

        while(i>=0 && s[i]==' '){

            i--;

        }

        while(i>=0 && s[i]!=' '){

            i--;

            length++;

        }

        return length;

    }

};

**EXPLANATION OF CODE**

1. I started by setting i to the last index of the string s.
2. I then used a loop to skip any trailing spaces at the end of the string.
3. Once I reached a non-space character, I started counting the characters of the last word.
4. I kept decrementing i and increasing the length until I encountered a space or reached the beginning of the string.
5. Finally, I returned the length, which is the length of the last word.



**QUESTION # 12**

**11. Reverse Words in a String** ***Medium***

Given an input string s, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in s will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

**Note** that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

**Example 1: Input:** s = "the sky is blue" **Output:** "blue is sky the"

**Example 2: Input:** s = " hello world " **Output:** "world hello" **Explanation:** Your reversed string should not contain leading or trailing spaces.

**Example 3: Input:** s = "a good example" **Output:** "example good a" **Explanation:** You need to reduce multiple spaces between two words to a single space in the reversed string.

**CODE**

class Solution {

public:

    string reverseWords(string s) {

        string newstr="";

        string ans="";

        int i=s.length()-1;

        while(i>=0){

        // removing leading spaces

        while(i>=0 && s[i]==' '){

            i--;

        }

        while(i>=0 && s[i]!=' '){

            newstr=s[i] + newstr;

            i--;

        }

        if (!newstr.empty()){

            if (!ans.empty()){

                ans=ans+" ";

            }

            ans=ans+newstr;

        }

        newstr="";

        }

    return ans;

    }

};

**EXPLANATION OF CODE**

1. I initialized two empty strings: newstr to hold individual words and ans to build the final result.
2. I started from the end of the string s and skipped any trailing spaces.
3. Then, I added characters to newstr until I reached a space or the start of the string.
4. After extracting a word, I added it to ans, ensuring there was a space between words.
5. I repeated this process for every word, resetting newstr each time.
6. Finally, I returned ans, which contains the reversed words.



**QUESTION # 13**

**69. Sqrt(x)** ***Easy***

Given a non-negative integer x, return *the square root of*x*rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

For example, do not use pow(x, 0.5) in c++ or x \*\* 0.5 in python.

**Example 1: Input:** x = 4 **Output:** 2 **Explanation:** The square root of 4 is 2, so we return 2.

**Example 2: Input:** x = 8 **Output:** 2 **Explanation:** The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

**CODE**

class Solution {

public:

    int mySqrt(int x) {

        int start = 0;

        int end = x;

        long long int ans;

        // long long int square;

        while(start <= end){

            long long int mid = start + (end-start) / 2;

            long long int square = mid \* mid;

            if (square == x){

                return mid;

            }

            else if (square < x){

                ans = mid;

                start = mid+1;

            }

            else{

                end = mid -1;

            }

        }

        return ans;

    }

};

**EXPLANATION OF CODE**

1. I initialized start to 0 and end to x to search within the range.
2. I used binary search to find the square root by calculating mid and squaring it.
3. If mid \* mid equals x, I returned mid as the square root.
4. If mid \* mid is less than x, I updated ans to mid and moved start to mid + 1.
5. If mid \* mid is greater than x, I moved end to mid - 1.
6. Finally, I returned ans, which holds the integer square root of x.



**QUESTION # 14**

**27. Remove Element** ***Easy***

Given an integer array nums and an integer val, remove all occurrences of val in nums [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm). The order of the elements may be changed. Then return *the number of elements in*nums*which are not equal to*val.

**Example 1: 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).

**Example 2: Input:** nums = [0,1,2,2,3,0,4,2], val = 2 **Output:** 5, nums = [0,1,4,0,3,\_,\_,\_] **Explanation:** Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4. Note that the five elements can be returned in any order.

It does not matter what you leave beyond the returned k (hence they are underscores).

**CODE**

class Solution {

public:

    int removeElement(vector<int>& nums, int val) {

        int count=0;

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

            if (nums[i]!=val){

                nums[count]=nums[i];

                count++;

            }

        }

        return count;

    }

};

**EXPLANATION OF CODE**

1. I initialized a count variable to keep track of the new length of the array after removing the target value.
2. I looped through the nums array.
3. If the current element is not equal to val, I moved it to the position indicated by count and incremented count.
4. Finally, I returned count, which represents the new length of the array after removal.



**QUESTION # 15**

**392. Is Subsequence** ***Easy***

Given two strings s and t, return true*if*s*is a****subsequence****of*t*, or*false*otherwise*.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

**Example 1: Input:** s = "abc", t = "ahbgdc" **Output:** true

**Example 2: Input:** s = "axc", t = "ahbgdc" **Output:** false

**CODE**

class Solution {

    public boolean isSubsequence(String s, String t) {

        int i = 0;

        int j = 0;

        while (j < t.length()){

            if (i < s.length() && s.charAt(i) == t.charAt(j)){

                i++;

            }

            j++;

        }

        return i == s.length(); // If i reaches end, 's' is a subsequence

    }

}

**EXPLANATION OF CODE**

1. I initialized two pointers, i and j, both set to 0. i tracks the position in string s, and j tracks the position in string t.
2. I looped through t with j. For each character in t, I checked if it matches the current character in s (i.e., s.charAt(i)).
3. If the characters match, I moved i to the next character in s (i.e., i++).
4. After the loop ends, if i reaches the end of s, it means all characters of s have been found in t in order, so I returned true.
5. If i does not reach the end of s, I returned false, indicating s is not a subsequence of t.



**QUESTION # 16**

**443. String Compression** ***Medium***

Given an array of characters chars, compress it using the following algorithm:

Begin with an empty string s. For each group of **consecutive repeating characters** in chars:

If the group's length is 1, append the character to s. Otherwise, append the character followed by the group's length.

**Example 1: Input:** chars = ["a","a","b","b","c","c","c"] **Output:** Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"] **Explanation:** The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".

**Example 2: Input:** chars = ["a"] **Output:** Return 1, and the first character of the input array should be: ["a"] **Explanation:** The only group is "a", which remains uncompressed since it's a single character.

**Example 3: Input:** chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"] **Output:** Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"]. **Explanation:** The groups are "a" and "bbbbbbbbbbbb". This compresses to "ab12".

**CODE**

class Solution {

public:

    int compress(vector<char>& chars) {

        int i=0;

        int n=chars.size();

        while(i<n){

            int j=i+1;

            while(j<n && chars[i]==chars[j]){

                j++;

            }

        }

    }

};

**EXPLANATION OF CODE**

1. I initialized i to 0 to start iterating through the chars array and n to store the size of the array.
2. I used a while loop to iterate through the characters in the array.
3. Inside the loop, I set j to i + 1 and used another while loop to find consecutive repeated characters (i.e., chars[i] == chars[j]).
4. The second loop continues until the characters are no longer equal or until I reach the end of the array.



**QUESTION # 17**

**50. Pow(x, n)** ***Medium***

Implement [pow(x, n)](http://www.cplusplus.com/reference/valarray/pow/), which calculates x raised to the power n (i.e., xn).

**Example 1: Input:** x = 2.00000, n = 10 **Output:** 1024.00000

**Example 2: Input:** x = 2.10000, n = 3 **Output:** 9.26100

**Example 3: Input:** x = 2.00000, n = -2 **Output:** 0.25000

**Explanation:** 2-2 = 1/22 = 1/4 = 0.25

**CODE**

class Solution {

public:

    double myPow(double x, int n) {

        double long result=pow(x,n);

        return result;

    }

};

**EXPLANATION OF CODE**

1. I used the built-in pow(x, n) function to calculate the power of x raised to n.
2. I declared a variable result of type double long to store the result of pow(x, n).
3. Finally, I returned the result as the output of the function.



**QUESTION # 18**

**231. Power of Two** ***Easy***

Given an integer n, return *true if it is a power of two. Otherwise, return false*.

An integer n is a power of two, if there exists an integer x such that n == 2x.

**Example 1: Input:** n = 1 **Output:** true **Explanation:** 20 = 1 **Example 2: Input:** n = 16 **Output:** true **Explanation:** 24 = 16 **Example 3: Input:** n = 3 **Output:** false

**CODE**

class Solution {

public:

    bool isPowerOfTwo(int n) {

        int ans=1;

        if (n==1){

            return true;

        }

        for (int i=0;i<30;i++){

        ans=ans\*2;

        if (ans==n){

            return true;

            }

        }

        return false;

    }

};

**EXPLANATION OF CODE**

1. I initialized ans to 1, which will be used to check if n is a power of two.
2. If n is 1, I immediately return true, as 1 is a power of two (2^0).
3. I then loop from 0 to 29 (because 2^30 is the largest power of two within the integer limit) and repeatedly multiply ans by 2.
4. In each iteration, I check if ans equals n. If it does, it means n is a power of two, so I return true.
5. If the loop finishes without finding a match, I return false, indicating that n is not a power of two.



**QUESTION # 19**

**567. Permutation in String** ***Medium***

Given two strings s1 and s2, return true if s2 contains a permutation of s1, or false otherwise.

In other words, return true if one of s1's permutations is the substring of s2.

**Example 1: Input:** s1 = "ab", s2 = "eidbaooo" **Output:** true **Explanation:** s2 contains one permutation of s1 ("ba").

**Example 2: Input:** s1 = "ab", s2 = "eidboaoo" **Output:** false

**CODE**

class Solution {

private:

    bool CheckEqual(int a[26], int b[26]){

        for (int i=0;i<26;i++){

            if (a[i] != b[i]){

                return 0;

            }

        }

        return 1;

    }

public:

    bool checkInclusion(string s1, string s2) {

        // Storing occurence of characters of s1

        int arr1[26]={0};

        for (int i=0;i<s1.length();i++){

            int index = s1[i]-'a';

            arr1[index]++;

        }

        int i=0;

        int window\_size=s1.length();

        int arr2[26]={0};

        while(i<window\_size && i<s2.length()){

            int index=s2[i]-'a';

            arr2[index]++;

            i++;

            if (CheckEqual(arr1, arr2)){

                return 1;

            }

        }

        // transversing through s2

        while(i<s2.length()){

            char new\_char = s2[i];

            int index = new\_char - 'a';

            arr2[index]++;

            char old\_char = s2[i-window\_size];

            index = old\_char - 'a';

            arr2[index]--;

            if (CheckEqual(arr1, arr2)){

                return 1;

            }

            i++;

        }

        return 0;

    }

};

**EXPLANATION OF CODE**

1. **CheckEqual Function**:  
   I created a function to compare the character frequency arrays (arr1 and arr2). If they are equal, it returns true; otherwise, false.
2. **Main Function** (checkInclusion):
   1. I first count the frequencies of characters in s1 and store them in arr1.
   2. Then, I slide a window over s2 with size equal to s1. I update the frequency of characters in arr2 and compare it with arr1 using CheckEqual.
   3. If at any point the frequencies match, I return true. If no match is found, I return false.



**QUESTION # 20**

**75. Sort Colors** ***Medium***

Given an array nums with n objects colored red, white, or blue, sort them [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

**Example 1: Input:** nums = [2,0,2,1,1,0] **Output:** [0,0,1,1,2,2]

**Example 2: Input:** nums = [2,0,1] **Output:** [0,1,2]

**CODE**

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int size=nums.size();

        for (int i=1;i<size;i++){

            for (int j=0;j<size-i;j++){

                if (nums[j]>nums[j+1]){

                    swap(nums[j],nums[j+1]);

                }

            }

        }

    }

};

**EXPLANATION OF CODE**

1. **Outer Loop**:  
   I loop through the array starting from index 1 to the last element. This controls the number of passes over the array to ensure that all elements are sorted.
2. **Inner Loop**:  
   The inner loop compares adjacent elements and swaps them if the current element is greater than the next one. This ensures the largest element "bubbles" to the end of the array after each pass.
3. **Swap**:  
   I use swap(nums[j], nums[j+1]) to exchange the positions of two elements if they're in the wrong order.



**QUESTION # 21**

**1910. Remove All Occurrences of a Substring** ***Medium***

Given two strings s and part, perform the following operation on s until **all** occurrences of the substring part are removed: Find the **leftmost** occurrence of the substring part and **remove** it from s.

Return s*after removing all occurrences of*part.

A **substring** is a contiguous sequence of characters in a string.

**Example 1: Input:** s = "daabcbaabcbc", part = "abc" **Output:** "dab"

**Example 2: Input:** s = "axxxxyyyyb", part = "xy" **Output:** "ab"

**CODE**

class Solution {

public:

    string removeOccurrences(string s, string part) {

        while(s.length()!=0 && s.find(part)<s.length()){

            s.erase(s.find(part),part.length());

        }

        return s;

    }

};

**EXPLANATION OF CODE**

1. I used a while loop to check if part exists in s using s.find(part).
2. Inside the loop, if part is found, I used s.erase() to remove it.
3. The loop continues until there are no more occurrences of part in s.
4. Finally, I returned the modified string s.



**QUESTION # 22**

**1929. Concatenation of Array** ***Easy***

Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (**0-indexed**).

Specifically, ans is the **concatenation** of two nums arrays.

Return *the array*ans.

**Example 1: Input:** nums = [1,2,1] **Output:** [1,2,1,1,2,1] **Explanation:** The array ans is formed as follows: ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]] , ans = [1,2,1,1,2,1]

**Example 2: Input:** nums = [1,3,2,1] **Output:** [1,3,2,1,1,3,2,1] **Explanation:** The array ans is formed as follows: ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]] , ans = [1,3,2,1,1,3,2,1]

**CODE**

class Solution {

public:

    vector<int> getConcatenation(vector<int>& nums) {

        int n=nums.size();

        vector<int> result(2\*n);

        // for first

        for (int i=0;i<n;i++){

            result[i]=nums[i];

        }

        // for second

        for (int i=0;i<n;i++){

            result[n+i]=nums[i];

        }

        return result;

    }

};

**EXPLANATION OF CODE**

1. I started by finding the size of the input array nums and storing it in n.
2. I created a result vector of size 2 \* n to accommodate the concatenated array.
3. I used a for loop to copy elements from nums into the first half of result.
4. I used another for loop to copy the same elements into the second half of result, starting from index n.
5. Finally, I returned the concatenated result vector.



**QUESTION # 23**

**217. Contains Duplicate** ***Easy***

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 **Explanation:** The element 1 occurs at the indices 0 and 3.

**Example 2: Input:** nums = [1,2,3,4] **Output:** false **Explanation:** All elements are distinct.

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

**CODE**

class Solution {

public:

    bool containsDuplicate(vector<int>& nums) {

        unordered\_set<int> myset;

        for (int i:nums){

            if (myset.find(i)!=myset.end()){

                return true;

            }

            myset.insert(i);

        }

        return false;

    }

};

**EXPLANATION OF CODE**

1. I initialized an unordered set myset to store unique elements from the input array nums.
2. I used a for loop to iterate through each element i in nums.
3. Inside the loop, I checked if i is already present in myset using find().
4. If i is found in the set, I returned true immediately, indicating a duplicate.
5. If no duplicate was found, I inserted i into the set.
6. After the loop, if no duplicates were found, I returned false.



**QUESTION # 24**

**242. Valid Anagram** ***Easy***

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

**Example 1: Input:** s = "anagram", t = "nagaram" **Output:** true

**Example 2: Input:** s = "rat", t = "car" **Output:** false

**CODE**

class Solution {

public:

    bool isAnagram(string s, string t) {

        if (s.size()!=t.size()){

            return false;

        }

        sort(s.begin(),s.end());

        sort(t.begin(),t.end());

        if (s==t){

            return true;

        }

        return false;

    }

};

**EXPLANATION OF CODE**

1. I checked if the sizes of the two strings s and t are not equal. If they are not equal, I returned false immediately since they can't be anagrams.
2. I sorted both strings s and t using sort() to arrange the characters in alphabetical order.
3. I compared the sorted versions of s and t. If they are equal, I returned true, indicating that the strings are anagrams.
4. If the strings are not equal after sorting, I returned false, indicating that they are not anagrams.



**QUESTION # 25**

**387. First Unique Character in a String** ***Easy***

Given a string s, find the **first** non-repeating character in it and return its index. If it **does not** exist, return -1.

**Example 1: Input:** s = "leetcode" **Output:** 0 **Explanation:** The character 'l' at index 0 is the first character that does not occur at any other index.

**Example 2: Input:** s = "loveleetcode" **Output:** 2

**Example 3: Input:** s = "aabb" **Output:** -1

**CODE**

class Solution {

public:

    int firstUniqChar(string s) {

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

        for (char c:s){

            freq\_map[c]++;

        }

        for (int i=0;i<s.size();i++){

            if (freq\_map[s[i]]==1){

                return  i;

            }

        }

        return -1;

    }

};

**EXPLANATION OF CODE**

1. I initialized an unordered map freq\_map to store the frequency of each character in the string s.
2. I iterated over each character in s and updated the frequency count in freq\_map.
3. After building the frequency map, I iterated through the string again and checked if the frequency of the current character is 1 in the map.
4. If I found a character with a frequency of 1, I returned its index as the result, since it is the first unique character.
5. If no unique character is found, I returned -1, indicating that there are no unique characters in the string.

