Placement Preparation :

Coding Questions

(Description + Answers)

* Nachiket Gavad

Table of Contents

[1. Two Sum 3](#_Toc119999901)

[2. Add Two Numbers 5](#_Toc119999902)

[4. Median of Two Sorted Arrays 7](#_Toc119999903)

[7. Reverse Integer 9](#_Toc119999904)

[9. Palindrome Number 11](#_Toc119999905)

[13. Roman to Integer 13](#_Toc119999906)

[14. Longest Common Prefix 16](#_Toc119999907)

[17. Letter Combinations of a Phone Number 18](#_Toc119999908)

[19. Remove Nth Node From End of List 21](#_Toc119999909)

[20. Valid Parentheses 24](#_Toc119999910)

[21. Merge Two Sorted Lists 26](#_Toc119999911)

[26. Remove Duplicates from Sorted Array 29](#_Toc119999912)

[509. Fibonacci Number 33](#_Toc119999913)

[1137. N-th Tribonacci Number 35](#_Toc119999914)

## 1. Two Sum

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?

**Tip :**

1 ) Don’t see the solution of problem, before solving by yourself

class Solution {

public:

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

        vector<int> ans;

        //brute force

        for(int i=0;i<nums.size()-1;i++){

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

                if(nums[i]+nums[j]==target){

                    ans.push\_back(i);

                    ans.push\_back(j);

                    break;

                }

            }

        }

        return ans;

    }

};

## 2. Add Two Numbers

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.



**Example 1:**

**Input:** l1 = [2,4,3], l2 = [5,6,4]

**Output:** [7,0,8]

**Explanation:** 342 + 465 = 807.

**Example 2:**

**Input:** l1 = [0], l2 = [0]

**Output:** [0]

**Example 3:**

**Input:** l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

**Output:** [8,9,9,9,0,0,0,1]

**Constraints:**

* The number of nodes in each linked list is in the range [1, 100].
* 0 <= Node.val <= 9
* It is guaranteed that the list represents a number that does not have leading zeros.

ListNode\* dummy = new ListNode(0);

4. Median of Two Sorted Arrays **(**Hard)

Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be O(log (m+n)).

**Example 1:**

**Input:** nums1 = [1,3], nums2 = [2]

**Output:** 2.00000

**Explanation:** merged array = [1,2,3] and median is 2.

**Example 2:**

**Input:** nums1 = [1,2], nums2 = [3,4]

**Output:** 2.50000

**Explanation:** merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

**Constraints:**

* nums1.length == m
* nums2.length == n
* 0 <= m <= 1000
* 0 <= n <= 1000
* 1 <= m + n <= 2000
* -106 <= nums1[i], nums2[i] <= 106

The median would be the middle element in the case of an odd-length array or the mean of both middle elements in the case of even length array.

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

        double ans=0;

        vector<int> a;

        int i=0,j=0;

//The most basic approach is to merge both the sorted arrays using an array.

        while(i<nums1.size() && j<nums2.size()){

            if(nums1[i]<nums2[j]){

                a.push\_back(nums1[i]);

                i++;

            }

            else{

                a.push\_back(nums2[j]);

                j++;

            }

        }

        while(i<nums1.size()){

            a.push\_back(nums1[i]);

            i++;

        }

        while(j<nums2.size()){

            a.push\_back(nums2[j]);

            j++;

        }

        int c=nums1.size() + nums2.size();

        int mid=(0+c-1)/2;

        if(c%2==0){

            ans=(double)(a[mid]+a[mid+1])/2;

        }

        else{

            ans=a[mid];

        }

        return ans;

    }

};

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

**Constraints:**

* -231 <= x <= 231 - 1

class Solution {

public:

    int reverse(int x) {

        int ans=0;

        int temp=x;

        while(temp){

            int dig=temp%10;

            // outside range case

            if(ans>INT\_MAX/10 || ans < INT\_MIN/10){

                return 0;

            }

            ans=(ans\*10)+dig;

            temp/=10;

        }

        return ans;

    }

};

9. Palindrome Number **(**Easy)

Given an integer x, return true*if*x*is a****palindrome****, and*false*otherwise*.

**Example 1:**

**Input:** x = 121

**Output:** true

**Explanation:** 121 reads as 121 from left to right and from right to left.

**Example 2:**

**Input:** x = -121

**Output:** false

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3:**

**Input:** x = 10

**Output:** false

**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

**Constraints:**

* -231 <= x <= 231 - 1

**Follow up:** Could you solve it without converting the integer to a string?

class Solution {

public:

    bool isPalindrome(int x) {

        string s=to\_string(x);

        int st=0,e=s.size()-1;

        while(st<e){

            if(s[st]!=s[e]){

                return false;

            }

            st++;

            e--;

        }

        return true;

    }

};

13. Roman to Integer **(**Easy)

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

**Symbol** **Value**

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

**Example 1:**

**Input:** s = "III"

**Output:** 3

**Explanation:** III = 3.

**Example 2:**

**Input:** s = "LVIII"

**Output:** 58

**Explanation:** L = 50, V= 5, III = 3.

**Example 3:**

**Input:** s = "MCMXCIV"

**Output:** 1994

**Explanation:** M = 1000, CM = 900, XC = 90 and IV = 4.

**Constraints:**

* 1 <= s.length <= 15
* s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
* It is **guaranteed** that s is a valid roman numeral in the range [1, 3999].

class Solution {

public:

    int romanToInt(string s) {

        int ans=0;

        unordered\_map<char,int> m;

        m.insert({'I',1});

        m.insert({'V',5});

        m.insert({'X',10});

        m.insert({'L',50});

        m.insert({'C',100});

        m.insert({'D',500});

        m.insert({'M',1000});

        for(int i=0;i<s.size();i++){

            int s1=m[s[i]];

            if(i+1<s.size()){

                int s2=m[s[i+1]];

                if(s2>s1){

                    ans=ans-s1+s2;

                    i++;

                    continue;

                }

            }

            ans+=s1;

        }

        return ans;

    }

};

14. Longest Common Prefix **(**Easy)

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Example 1:**

**Input:** strs = ["flower","flow","flight"]

**Output:** "fl"

**Example 2:**

**Input:** strs = ["dog","racecar","car"]

**Output:** ""

**Explanation:** There is no common prefix among the input strings.

**Constraints:**

* 1 <= strs.length <= 200
* 0 <= strs[i].length <= 200
* strs[i] consists of only lowercase English letters.

class Solution {

public:

    string longestCommonPrefix(vector<string>& strs) {

        string ans="";

        // find length of shortest string

        int min\_len=INT\_MAX;

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

            if(strs[i].size()<min\_len){

                min\_len=strs[i].size();

            }

        }

        // iterate through each index

        for(int i=0;i<min\_len;i++){

            bool flag=true;

            char ch=strs[0][i];

            // compare char of other strings with current

            for(int j=1;j<strs.size();j++){

                if(strs[j][i]!=ch){

                    flag=false;

                    break;

                }

            }

            // if flag not become false push char to ans and continue

            if(flag){

                ans.push\_back(ch);

            }else{

                break;

            }

        }

        return ans;

    }

};

17. Letter Combinations of a Phone Number **(**Medium)

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



**Example 1:**

**Input:** digits = "23"

**Output:** ["ad","ae","af","bd","be","bf","cd","ce","cf"]

**Example 2:**

**Input:** digits = ""

**Output:** []

**Example 3:**

**Input:** digits = "2"

**Output:** ["a","b","c"]

**Constraints:**

* 0 <= digits.length <= 4
* digits[i] is a digit in the range ['2', '9'].

class Solution {

public:

    void solve(string digits,int ind,string output,vector<string>& ans,vector<string>& mapi){

        if(ind>=digits.size()){

            if(output.length()>0)

                ans.push\_back(output);

            return ;

        }

        // index in mapi

        int index= digits[ind]-'0';

        string str=mapi[index];

        // recursive solution

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

            // exclude

            // no exclude because only included we need in answer

            // solve(digits,ind+1,output,ans,mapi);

            //include

            output.push\_back(str[i]);

            solve(digits,ind+1,output,ans,mapi);

            output.pop\_back();

        }

    }

    vector<string> letterCombinations(string digits) {

        vector<string> ans;

        //edge case if given digits input is empty

        if(digits.length()==0){

            return ans;

        }

        vector<string> mapi={"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

        string output="";

        solve(digits,0,output,ans,mapi);

        return ans;

    }

};

## 19. Remove Nth Node From End of List

Medium

Given the head of a linked list, remove the nth node from the end of the list and return its head.



**Example 1:**

**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

**Example 2:**

**Input:** head = [1], n = 1

**Output:** []

**Example 3:**

**Input:** head = [1,2], n = 1

**Output:** [1]

**Constraints:**

The number of nodes in the list is sz.

1 <= sz <= 30

0 <= Node.val <= 100

1 <= n <= sz

**Follow up:** Could you do this in one pass?

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode() : val(0), next(nullptr) {}

 \*     ListNode(int x) : val(x), next(nullptr) {}

 \*     ListNode(int x, ListNode \*next) : val(x), next(next) {}

 \* };

 \*/

class Solution {

public:

    int count(ListNode\* head){

        int c=0;

        ListNode\* temp=head;

        while(temp){

            c++;

            temp=temp->next;

        }

        return c;

    }

    ListNode\* reverse(ListNode\* head){

        if(head==NULL){

            return head;

        }

        ListNode \*temp=head,\*prev=NULL,\*next=head;

        while(temp){

            next=temp->next;

            temp->next=prev;

            prev=temp;

            temp=next;

        }

        return prev;

    }

    ListNode\* removeNthFromEnd(ListNode\* head, int n) {

        if(head==NULL){

            return head;

        }

        int c=count(head);

        if(n>c){

            return head;

        }

        if(n==1 && c==1){

            return NULL;

        }

        head = reverse(head);

        ListNode \*temp=head,\*prev=head;

        c=1;

        if(n==1){

            head=head->next;

        }

        while(temp){

            if(c==n){

                prev->next=temp->next;

            }

            prev=temp;

            temp=temp->next;

            c++;

        }

        return reverse(head);

    }

};

20. Valid Parentheses(Easy)

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Every close bracket has a corresponding open bracket of the same type.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

**Constraints:**

1 <= s.length <= 104

s consists of parentheses only '()[]{}'.

class Solution {

public:

    bool isValid(string s) {

        stack<char> st;

        for(int i=0;i<s.size();i++){

            switch(s[i]){

                case '(':

                    st.push(s[i]);

                    break;

                case '[':

                    st.push(s[i]);

                    break;

                case '{':

                    st.push(s[i]);

                    break;

                case ')':

                    if(!st.empty() && st.top()=='('){

                        st.pop();

                    }

                    else{

                        return false;

                    }

                    break;

                case ']':

                    if(!st.empty() && st.top()=='['){

                        st.pop();

                    }

                    else{

                        return false;

                    }

                    break;

                case '}':

                    if(!st.empty() && st.top()=='{'){

                        st.pop();

                    }

                    else{

                        return false;

                    }

                    break;

            }

        }

        if(st.empty()){

            return true;

        }

        return false;

    }

};

21. Merge Two Sorted Lists **(**Easy)

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.



**Example 1:**

**Input:** list1 = [1,2,4], list2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Example 2:**

**Input:** list1 = [], list2 = []

**Output:** []

**Example 3:**

**Input:** list1 = [], list2 = [0]

**Output:** [0]

**Constraints:**

The number of nodes in both lists is in the range [0, 50].

-100 <= Node.val <= 100

Both list1 and list2 are sorted in **non-decreasing** order.

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode() : val(0), next(nullptr) {}

 \*     ListNode(int x) : val(x), next(nullptr) {}

 \*     ListNode(int x, ListNode \*next) : val(x), next(next) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

        if(list1==NULL){

            return list2;

        }

        if(list2==NULL){

            return list1;

        }

        ListNode \*i,\*j,\*head=NULL;

        i=list1;

        j=list2;

        if(i->val < j->val){

            head=i;

            i=i->next;

        }

        else{

            head=j;

            j=j->next;

        }

        ListNode \*temp=head;

        while(i!=NULL && j!=NULL){

            if((i->val) < (j->val)){

                temp->next=i;

                i=i->next;

            }

            else{

                temp->next=j;

                j=j->next;

            }

            temp=temp->next;

        }

        if(i!=NULL){

            temp->next=i;

        }

        else{

            temp->next=j;

        }

        return head;

    }

};

26. Remove Duplicates from Sorted Array(Easy)

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k*after placing the final result in the first*k*slots of*nums.

Do **not** allocate extra space for another array. You must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Custom Judge:**

The judge will test your solution with the following code:

int[] nums = [...]; // Input array

int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;

for (int i = 0; i < k; i++) {

assert nums[i] == expectedNums[i];

}

If all assertions pass, then your solution will be **accepted**.

**Example 1:**

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

**Output:** 2, nums = [1,2,\_]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

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

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

**Explanation:** Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

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

**Constraints:**

1 <= nums.length <= 3 \* 104

-100 <= nums[i] <= 100

nums is sorted in **non-decreasing** order.

class Solution {

public:

    int removeDuplicates(vector<int>& nums) {

        int r=nums.size()-1,num=nums[0],i=1,k=nums.size(),j=1;

        while(i<=r){

            if(nums[i]==num){

                k--;

            }

            else{

                num=nums[i];

                nums[j++]=num;

            }

            i++;

        }

        return k;

    }

}

## 509. Fibonacci Number

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.

**Constraints:**

* 0 <= n <= 30

class Solution {

public:

    int fib(int n) {

        //recursion

        if(n==0 or n==1){

            return n;

        }

        return fib(n-1)+fib(n-2);

        //memoization

        if(n==0 or n==1){

            return t[n]=n;

        }

        return t[n]=fib(n-1)+fib(n-2);

        //tabulation

        int t[n+1];

        for(int i=0;i<=n;i++){

            if(i==0 or i==1){

                t[i]=i;

            }

            else{

                t[i]=t[i-1]+t[i-2];

            }

        }

        return t[n];

    }

};

## 1137. N-th Tribonacci Number

The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.

**Example 1:**

**Input:** n = 4

**Output:** 4

**Explanation:**

T\_3 = 0 + 1 + 1 = 2

T\_4 = 1 + 1 + 2 = 4

**Example 2:**

**Input:** n = 25

**Output:** 1389537

**Constraints:**

* 0 <= n <= 37
* The answer is guaranteed to fit within a 32-bit integer, ie. answer <= 2^31 - 1.

class Solution {

public:

    int tribonacci(int n) {

        int t[40];

        t[0]=0;

        t[1]=1;

        t[2]=1;

        for(int i=3;i<=n;i++){

            t[i]=t[i-3]+t[i-2]+t[i-1];

        }

        return t[n];

    }

};