Placement Preparation :

Coding Questions

(Description + Answers)

- Nachiket Gavad

Table of Contents

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:

- $2 <= nums.length <= 10^4$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$
- **Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

Tip :

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

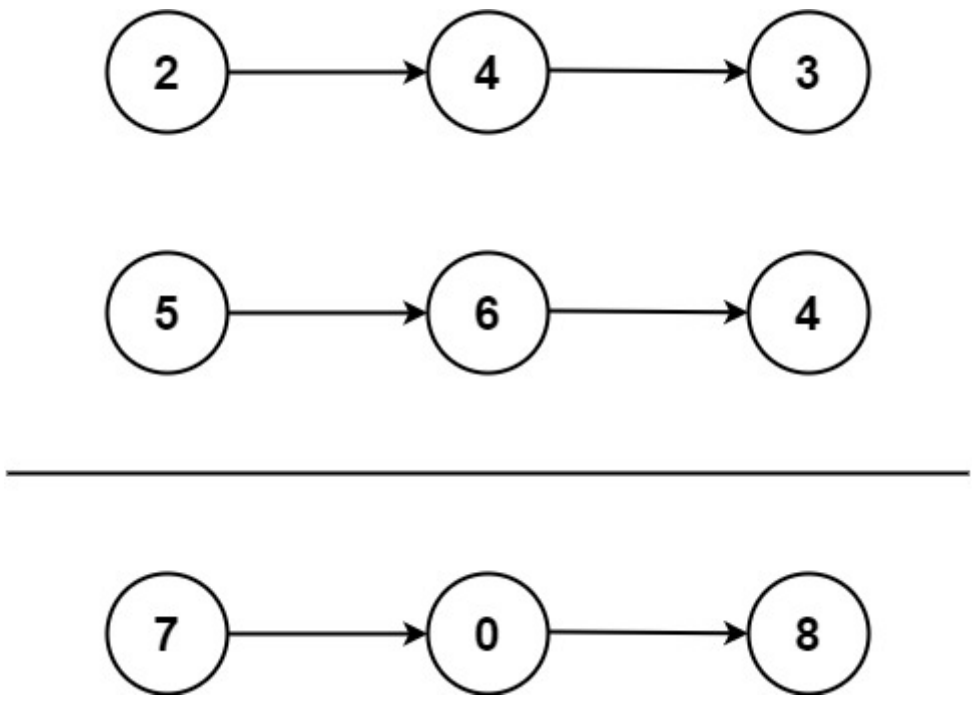
```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

        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;
    }
};
```

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        // by hashmap
        vector<int> ans;
        unordered_map<int,int> m;
        for(int i=0;i<nums.size();i++){
            m[nums[i]]=i;
        }

        int rem;
        for(int i=0;i<nums.size();i++){
            rem=target-nums[i];
            if(m[rem]!=0 && m[rem]!=i){
                ans.push_back(i);
                ans.push_back(m[rem]);
                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
- -10^6 <= nums1[i], nums2[i] <= 10^6

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.

```cpp
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 $[-2^{31}, 2^{31} - 1]$, 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:

- $-2^{31} <= x <= 2^{31} - 1$

```cpp
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:

- $-2^{31} <= x <= 2^{31} - 1$

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

```cpp
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].

```cpp
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.

```cpp
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:

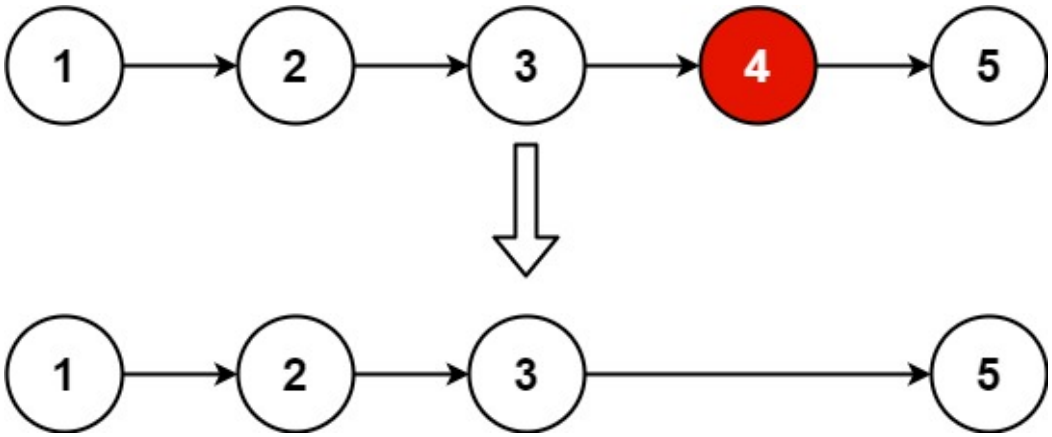
- 0 <= digits.length <= 4
- digits[i] is a digit in the range ['2', '9'].

```
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 n^{th} 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:

$1 <= s.length <= 10^4$

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

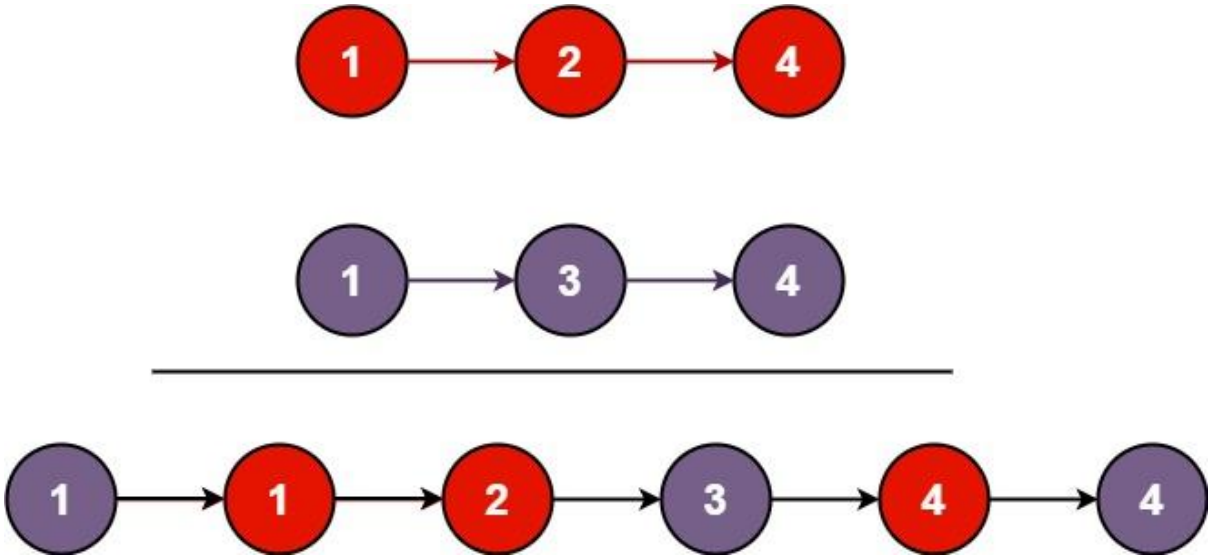
```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;

        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** 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** 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:

$1 <= nums.length <= 3 * 10^4$

$-100 <= nums[i] <= 100$

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;
    }
}
```

27. Remove Element (Easy)

Given an integer array nums and an integer val, remove all occurrences of val in nums **in-place**. The relative order of the elements may be changed.

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** with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

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

int val = ...; // Value to remove

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

                            // It is sorted with no values equaling val.


int k = removeElement(nums, val); // Calls your implementation


assert k == expectedNums.length;

sort(nums, 0, k); // Sort the first k elements of nums

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

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

}
```

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

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 doe `s not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

0 <= nums.length <= 100

0 <= nums[i] <= 50

0 <= val <= 100

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int n=nums.size(),r=n;

        for(int i=n-1;i>=0;i--){
            if(nums[i]==val){
                r--;
                swap(nums[i],nums[r]);
            }
        }
        return r;
    }
};
```

33. Search in Rotated Sorted Array (Medium)

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.

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

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0

Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Example 3:

Input: nums = [1], target = 0

Output: -1

Constraints:

1 <= nums.length <= 5000

-10^4 <= nums[i] <= 10^4

All values of nums are **unique**.

nums is an ascending array that is possibly rotated.

-10^4 <= target <= 10^4

```cpp
class Solution {
public:
    int findMin(vector<int>& nums) {
        int s=0,e=nums.size()-1,mid;

        // corner case - for array rotated in n times;
        if(nums[0]<nums[e]){
            return 0;
        }

        while(s<e){
            mid=(s+e)/2;
            if(nums[mid]>=nums[0]){
                s=mid+1;
            }
            else{
                e=mid;
            }
        }

        return s;
    }
    int binary_search(vector<int>& nums, int target,int s,int e){
        int mid;
        while(s<=e){
            mid=(s+e)/2;

            if(nums[mid]==target){
                return mid;
            }
            else if(nums[mid]<target){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        return -1;
    }


    int search(vector<int>& nums, int target) {
        int pivot=findMin(nums);

        if(target>=nums[pivot] && target<=nums[nums.size()-1]){
            return binary_search(nums,target,pivot,nums.size()-1);
        }
        else{
```

```
            return binary_search(nums,target,0,pivot-1);
        }

    }
};
```

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]

Constraints:

$0 <= nums.length <= 10^5$

$-10^9 <= nums[i] <= 10^9$

nums is a non-decreasing array.

$-10^9 <= target <= 10^9$

```cpp
class Solution {
public:
    int first(vector<int>& nums, int k){
        int s=0,e=nums.size()-1;
        int ans=-1;

        while(s<=e){
            int mid=s+(e-s)/2;
            if(nums[mid]==k){
                ans=mid;
                e=mid-1;
            }
            else if(nums[mid]<k){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }

        return ans;
    }

    int last(vector<int>& nums, int k){
        int s=0,e=nums.size()-1;
        int ans=-1;

        while(s<=e){
            int mid=s+(e-s)/2;
            if(nums[mid]==k){
                ans=mid;
                s=mid+1;
            }
            else if(nums[mid]<k){
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }

        return ans;
    }

    vector<int> searchRange(vector<int>& nums, int target) {
        vector<int> ans;
        ans.push_back(first(nums,target));
        ans.push_back(last(nums,target));
```

```
        return ans;
    }
};
```

42. Trapping Rain Water (Hard)

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]

Output: 9

Constraints:

n == height.length

$1 <= n <= 2 * 10^4$

$0 <= height[i] <= 10^5$

```cpp
class Solution {
public:
    int trap(vector<int>& height) {
        int n=height.size(),maxl[n],maxr[n],maxil=INT_MIN,maxir=INT_MIN;

        for(int i=0;i<n;i++){
            if(height[i]>maxil){
                maxil=height[i];
            }
            maxl[i]=maxil;
        }
        for(int i=n-1;i>=0;i--){
            if(height[i]>maxir){
                maxir=height[i];
            }
            maxr[i]=maxir;
        }

        int ans=0;
        for(int i=1;i<n-1;i++){
            int h=min(maxl[i-1],maxr[i+1]);

            if(height[i]<h){
                ans+=(h-height[i]);
            }

        }
        return ans;
    }
};
```

46. Permutations (Medium)

[Love Babbar – Lecture 39]

Given an array nums of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

Input: nums = [1,2,3]

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

Example 2:

Input: nums = [0,1]

Output: [[0,1],[1,0]]

Example 3:

Input: nums = [1]

Output: [[1]]

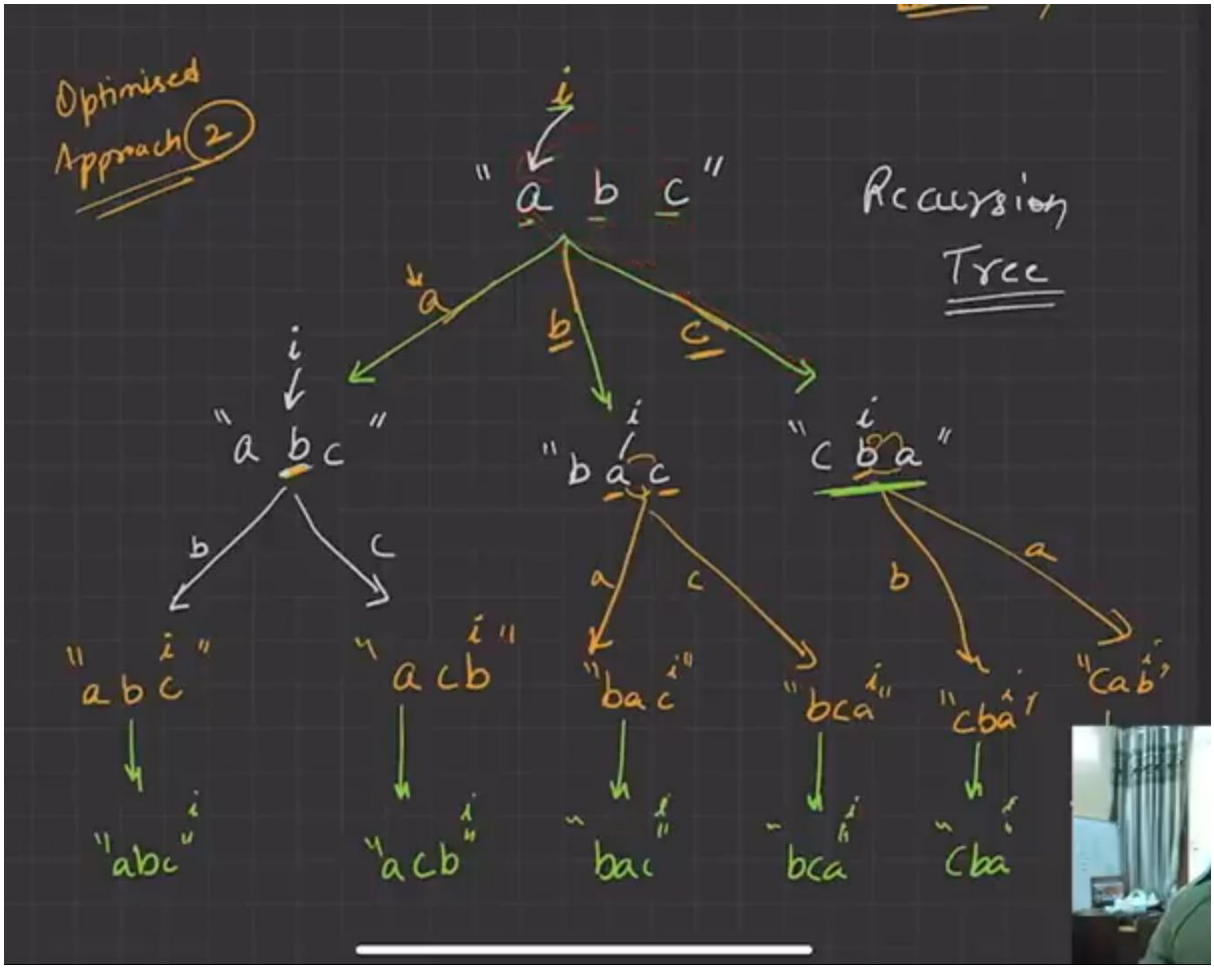
Constraints:

- 1 <= nums.length <= 6
- -10 <= nums[i] <= 10
- All the integers of nums are **unique**.

Idea :

1) Try putting every character at every position while simulation

2) Number of permutations will be n!



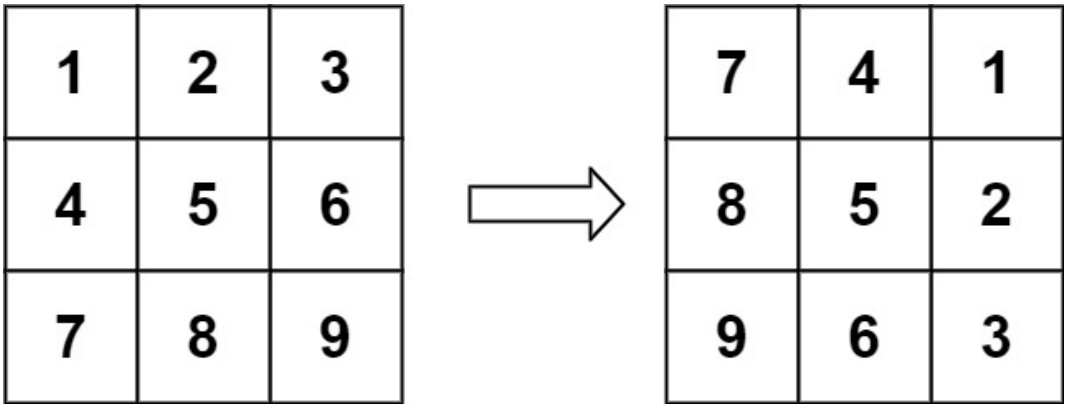

```cpp
class Solution {
public:
    void solve(vector<int> nums,int ind,vector<vector<int>>& ans){

        if(ind>=nums.size()){
            ans.push_back(nums);
            return;
        }

        // recursive
        for(int i=ind;i<nums.size();i++){
            swap(nums[ind],nums[i]);
            solve(nums,ind+1,ans);
            //backtracking
            swap(nums[ind],nums[i]);
        }
    }
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> ans;
        solve(nums,0,ans);
        return ans;
    }
}
```

48. Rotate Image (Medium)

You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

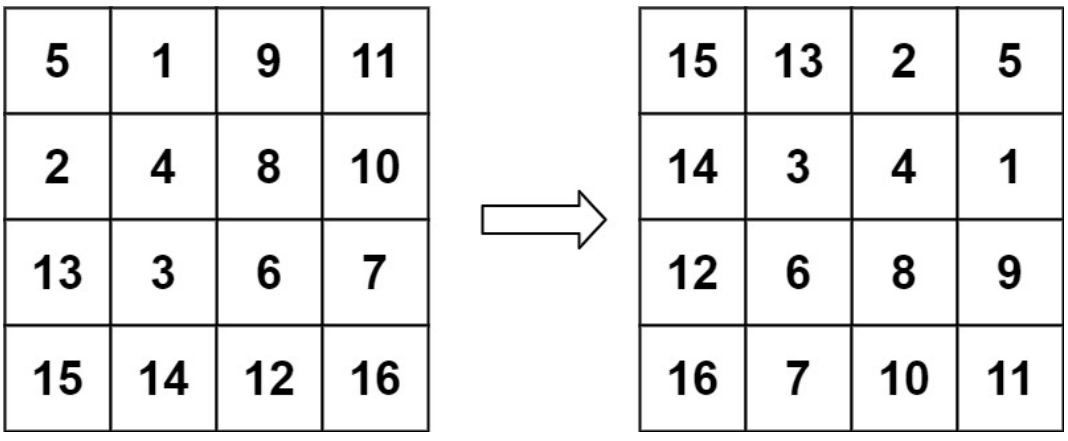
Example 1:



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:



Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Constraints:

n == matrix.length == matrix[i].length

1 <= n <= 20

-1000 <= matrix[i][j] <= 1000

```cpp
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n=matrix.size(),m=matrix[0].size();

        // transpose
        for(int i=0;i<n;i++){
            for(int j=i;j<m;j++){
                swap(matrix[i][j],matrix[j][i]);
            }
        }

        // reverse columns, using 2 ptr
        for(int i=0;i<n;i++){

            int l=0,r=m-1;

            while(l<r){
                swap(matrix[i][l],matrix[i][r]);
                l++;
                r--;
            }
        }
    }
};
```

53. Maximum Subarray (Medium)

[Kadane's Algorithm]

Given an integer array nums, find the subarray which has the largest sum and return *its sum*.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Example 2:

Input: nums = [1]

Output: 1

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

Constraints:

- $1 <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$

Follow up: If you have figured out the O(n) solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

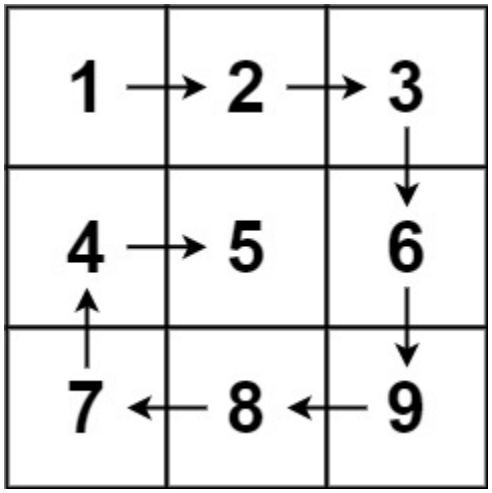
```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int max=INT_MIN,max_here=0;

        for(int i=0;i<nums.size();i++){
            max_here=max_here+nums[i];
            if(max<max_here){
                max=max_here;
            }
            if(max_here<0){
                max_here=0;
            }
        }
        return max;
    }
};
```

54. Spiral Matrix (Medium)

[Love Babbar – Lecture 23 - 46:00 to 1:00]

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

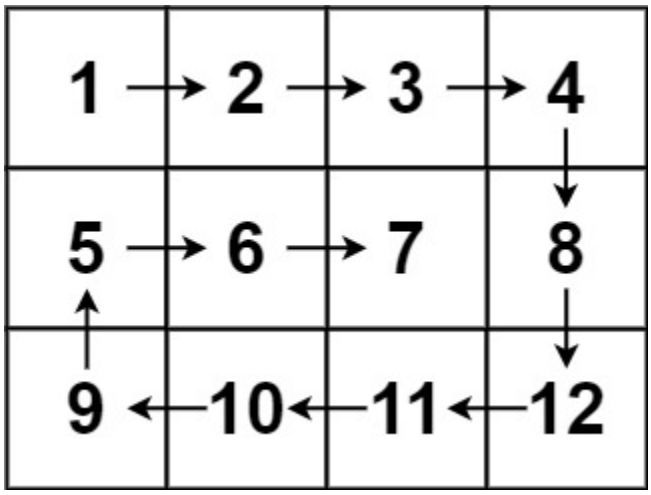
Example 1:



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]

Example 2:



Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Constraints:

- m == matrix.length
- n == matrix[i].length
- 1 <= m, n <= 10
- -100 <= matrix[i][j] <= 100

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> ans;
        int m=matrix.size(),n=matrix[0].size(),c=0,total=m*n;
        int startrow=0,endrow=m-1,startcol=0,endcol=n-1;

        while(c<total){
            //print start row
            for(int i=startcol;c<total && i<=endcol;i++){
                ans.push_back(matrix[startrow][i]);
                c++;
            }
            startrow++;

            //print end col
            for(int i=startrow;c<total && i<=endrow;i++){
                ans.push_back(matrix[i][endcol]);
                c++;
            }
            endcol--;

            //print end row
            for(int i=endcol;c<total && i>=startcol;i--){
                ans.push_back(matrix[endrow][i]);
                c++;
            }
            endrow--;

            //print start col
            for(int i=endrow;c<total && i>=startrow;i--){
                ans.push_back(matrix[i][startcol]);
                c++;
            }
            startcol++;
        }
        return ans;
    }
};
```

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.

Constraints:

- 1 <= s.length <= 10^4
- s consists of only English letters and spaces ' '.
- There will be at least one word in s.

```cpp
class Solution {
public:
    int lengthOfLastWord(string s) {
        int e=s.size()-1,c=0;
        bool word=false;

        while(e>=0){
            if(word){
                if(s[e]==' '){
                    break;
                }
                c++;
            }
            else{
                if(s[e]!=' '){
                    word=true;
                    c++;
                }
            }
            e--;
        }
        return c;
    }
};
```

66. Plus One (Easy)

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].

Example 2:

Input: digits = [4,3,2,1]

Output: [4,3,2,2]

Explanation: The array represents the integer 4321.

Incrementing by one gives 4321 + 1 = 4322.

Thus, the result should be [4,3,2,2].

Example 3:

Input: digits = [9]

Output: [1,0]

Explanation: The array represents the integer 9.

Incrementing by one gives 9 + 1 = 10.

Thus, the result should be [1,0].

Constraints:

- 1 <= digits.length <= 100
- 0 <= digits[i] <= 9
- digits does not contain any leading 0's.

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int i=digits.size()-1,carry=1;

        while(carry && i>=0){
            if(digits[i]+1>9){
                carry=1;
                digits[i]=0;
            }
            else{
                digits[i]++;
                carry=0;
            }
            i--;
        }
        if(carry){
            digits.insert(digits.begin(),carry);
        }
        return digits;
    }
};
```

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.

Constraints:

- $0 <= x <= 2^{31} - 1$

```cpp
class Solution {
public:
    int mySqrt(int x) {
        long long s=0,e=x,mid;
        int ans=0;

        while(s<=e){
            mid=(s+e)/2;

            if(mid*mid<=x){
                ans=mid;
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        return ans;
    }
};
```

70. Climbing Stairs (Easy)

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

Constraints:

- 1 <= n <= 45

```cpp
class Solution {
public:
    int climbStairs(int n) {
        int f=0,s=1,ne;
        for(int i=0;i<=n;i++){
            ne=f+s;
            s=f;
            f=ne;
        }
        return ne;
    }
};
```

```cpp
class Solution {
public:
    vector<int> dp(46,-1);

    int climbStairs(int n) {
        if(dp[n]!=-1){
            return dp[n];
        }

        if(n==0 || n==1){
            return dp[n]=1;
        }

        int x,y;
        x=climbStairs(n-1);
        y=climbStairs(n-2);

        return dp[n]=x+y;
    }
};
```

191. Number of 1 Bits (Easy)

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 3**, the input represents the signed integer. -3.

Example 1:

Input: n = 00000000000000000000000000001011

Output: 3

Explanation: The input binary string **00000000000000000000000000001011** has a total of three '1' bits.

Example 2:

Input: n = 00000000000000000000000010000000

Output: 1

Explanation: The input binary string **00000000000000000000000010000000** has a total of one '1' bit.

Example 3:

Input: n = 11111111111111111111111111111101

Output: 31

Explanation: The input binary string **11111111111111111111111111111101** has a total of thirty one '1' bits.

Constraints:

- The input must be a **binary string** of length 32.

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int c=0;

        while(n>0){
            if(n&1){
                c=c+1;
            }
            n=n>>1;
        }
        return c;
    }
};
```

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

```cpp
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 T_n is defined as follows:

$T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for n >= 0.

Given n, return the value of T_n.

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.

```cpp
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];
    }
};
```