Placement Preparation : Coding Questions (Description + Answers)

- Nachiket Gavad

Table of Contents

1. Two Sum 4
2. Add Two Numbers 6
4. Median of Two Sorted Arrays 10
7. Reverse Integer 12
9. Palindrome Number 14
13. Roman to Integer 16
14. Longest Common Prefix 19
17. Letter Combinations of a Phone Number 21
19. Remove Nth Node From End of List 24
20. Valid Parentheses 27
21. Merge Two Sorted Lists 29
26. Remove Duplicates from Sorted Array 32
27. Remove Element 34
28. Find the Index of the First Occurrence in a String 36
Using stl 37
33. Search in Rotated Sorted Array 38
34. Find First and Last Position of Element in Sorted Array 41
42. Trapping Rain Water 44
46. Permutations 46
48. Rotate Image 48
53. Maximum Subarray 50
54. Spiral Matrix 52
58. Length of Last Word 54
66. Plus One 56
69. Sqrt(x) 58
70. Climbing Stairs 60
78. Subsets 62
100. Same Tree 65
104. Maximum Depth of Binary Tree 68
110. Balanced Binary Tree 70
121. Best Time to Buy and Sell Stock 73
191. Number of 1 Bits 75
198. House Robber 77

226. Invert Binary Tree 79
258. Add Digits 81
268. Missing Number 83
338. Counting Bits 85
374. Guess Number Higher or Lower 87
389. Find the Difference 89
342. Power of Four 91
412. Fizz Buzz 93
415. Add Strings 95
448. Find All Numbers Disappeared in an Array 97
455. Assign Cookies 99
485. Max Consecutive Ones 101
495. Teemo Attacking 103
500. Keyboard Row 105
509. Fibonacci Number 109
540. Single Element in a Sorted Array 111
543. Diameter of Binary Tree 113
605. Can Place Flowers 116
700. Search in a Binary Search Tree 119
709. To Lower Case 121
875. Koko Eating Bananas 122
912. Sort an Array 124
1011. Capacity To Ship Packages Within D Days 126
1137. N-th Tribonacci Number 129
2187. Minimum Time to Complete Trips 131
2348. Number of Zero-Filled Subarrays 134
2444. Count Subarrays With Fixed Bounds 136
Intuition 137
Explanation 137
Complexity 138

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

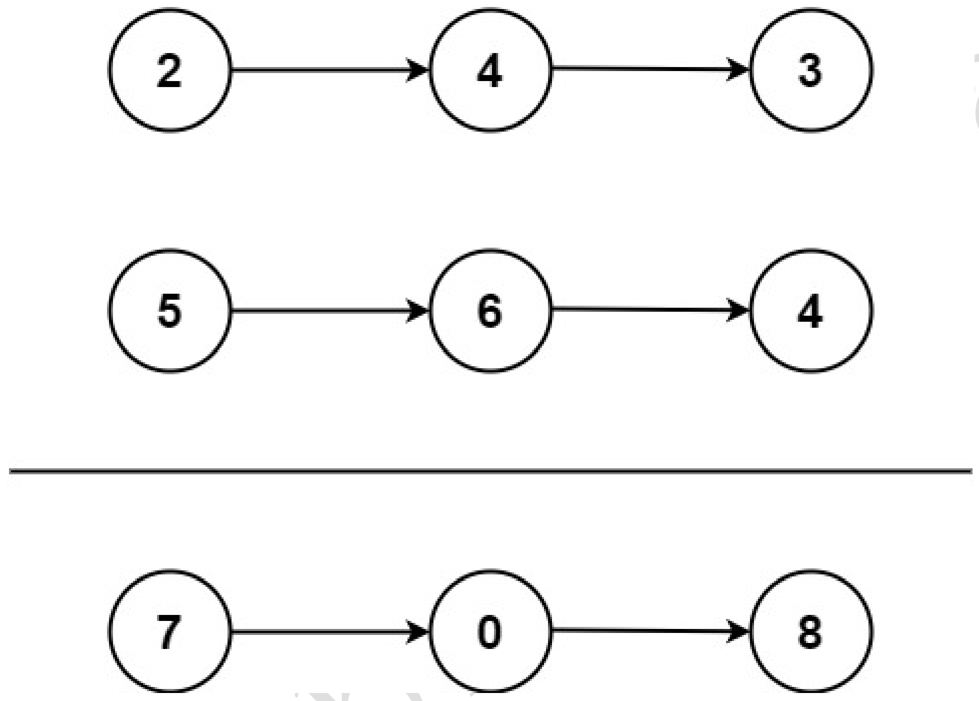
```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

        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;
    }
};
```

```
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);
```

```
/**
 * 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* addTwoNumbers(ListNode* l1, ListNode* l2) {

        // calculate sum & carry for first node
        int sum=l1->val + l2->val,carry=0;
        if(sum>9){
            carry=1;
            sum=sum%10;
        }
        else{
            carry=0;
        }

        // creation of first node
        // finalans for storing address of ans
        ListNode *ans=new ListNode(sum),*finalans=ans;
        l1=l1->next;
        l2=l2->next;

        // traverse till both lists have not null value
        while(l1 && l2){
            sum=l1->val + l2->val+carry;
            if(sum>9){
                carry=1;
                sum=sum%10;
            }
            else{
                carry=0;
            }
            ListNode *nextnode=new ListNode(sum);
            ans->next=nextnode;
            ans=ans->next;
            l1=l1->next;
            l2=l2->next;
        }
```

```
       // traverse remaining part of first
        while(l1){
            sum=l1->val+carry;
            if(sum>9){
                carry=1;
                sum=sum%10;
            }
            else{
                carry=0;
            }
            ListNode *nextnode=new ListNode(sum);
            ans->next=nextnode;
            ans=ans->next;
            l1=l1->next;
        }

        // traverse remaining part of second
        while(l2){
            sum= l2->val+carry;
            if(sum>9){
                carry=1;
                sum=sum%10;
            }
            else{
                carry=0;
            }
            ListNode *nextnode=new ListNode(sum);
            ans->next=nextnode;
            ans=ans->next;
            l2=l2->next;
        }

        // check for carry
        if(carry>0){
            ListNode *nextnode=new ListNode(carry);
            ans->next=nextnode;
        }

        // return final answer
        return finalans;
    }
};
```

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].

```
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:

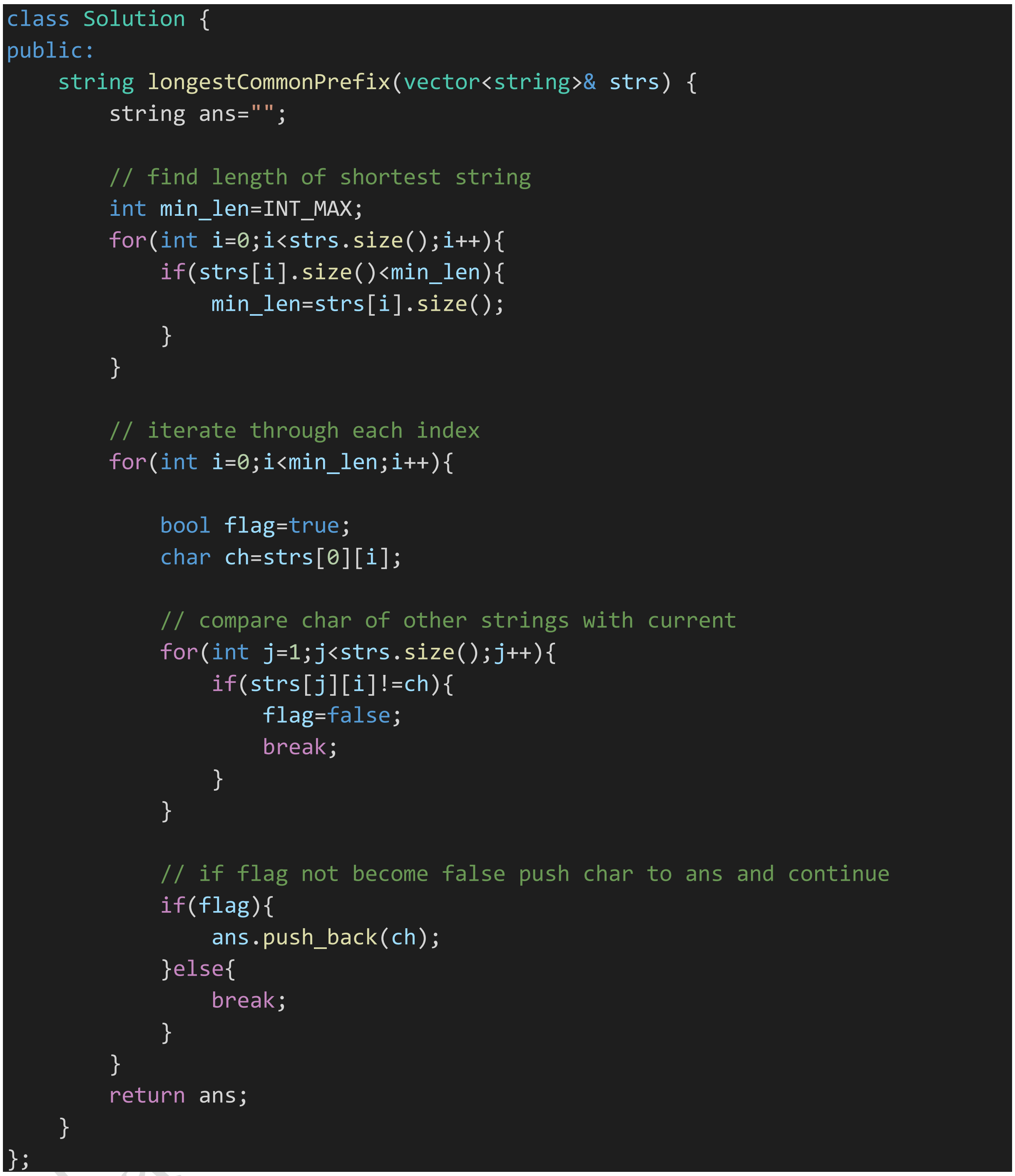
Input: strs = ["dog","racecar","car"]

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.

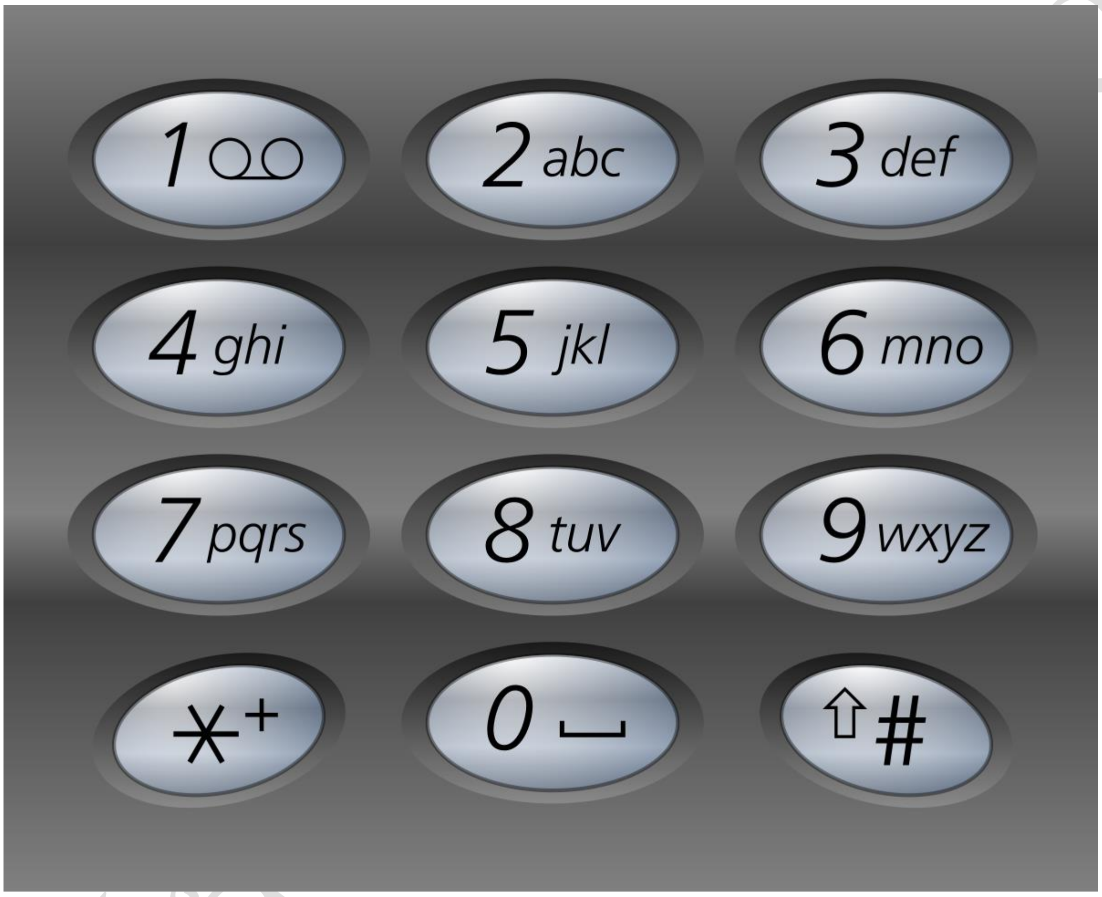


```cpp
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string ans="";

        // 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:

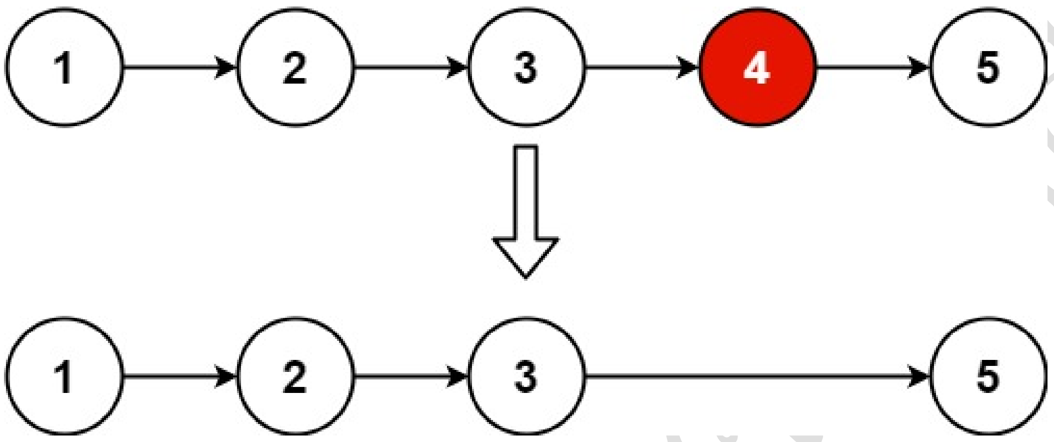
- 0 <= digits.length <= 4
- digits[i] is a digit in the range ['2', '9'].

```
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?

```cpp
/**
 * 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 '()[]{}'.

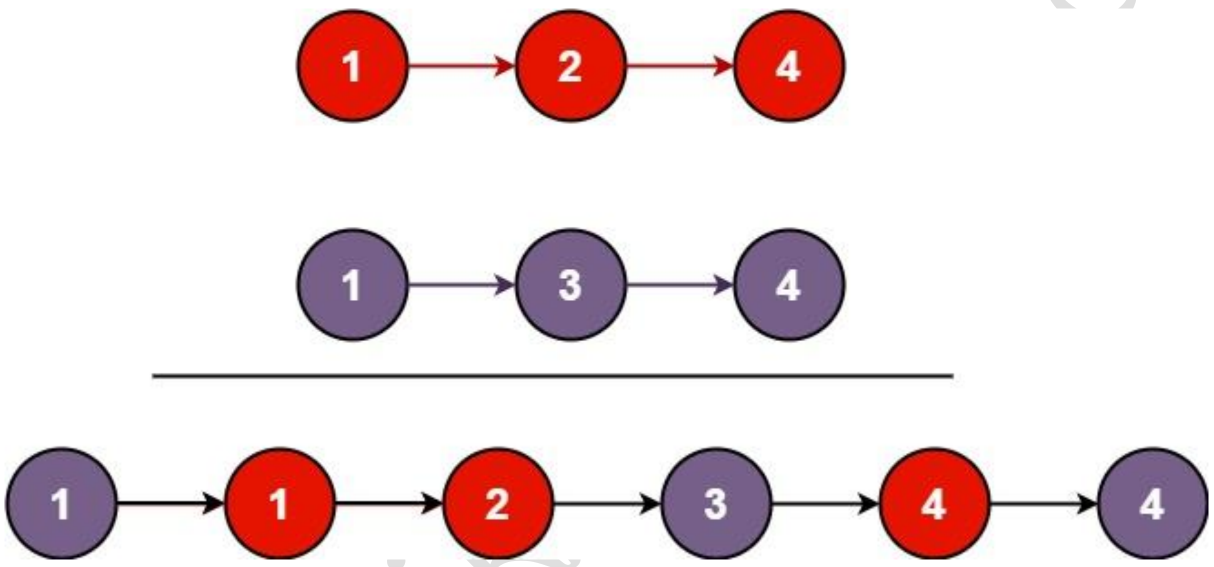
```cpp
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;

        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.

```cpp
/**
 * 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.

```cpp
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;
    }
};
```

28. Find the Index of the First Occurrence in a String Medium

Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "sadbutsad", needle = "sad"

Output: 0

Explanation: "sad" occurs at index 0 and 6.

The first occurrence is at index 0, so we return 0.

Example 2:

Input: haystack = "leetcode", needle = "leeto"

Output: -1

Explanation: "leeto" did not occur in "leetcode", so we return -1.

Constraints:

- 1 <= haystack.length, needle.length <= 10^4
- haystack and needle consist of only lowercase English characters.

```cpp
class Solution {
public:
    int strStr(string haystack, string needle) {
        int m=haystack.size(),n=needle.size();
        if(n>m){
            return -1;
        }
        for(int i=0;i<=m-n;i++){
            // cout<<i;
            int k=0,c=0,j=0;
            if(haystack[i]==needle[0]){
                c=1;
                j=i+1;
                int k=1;
                while((k<n) && (j<m) && (haystack[j++]==needle[k++])){
                    // cout<<haystack[j++]<<" "<<needle[k++];
                    c++;
                }
                // cout<<c;
                if(c==n){
                    return i;
                }
            }
        }
        return -1;
    }
};
```

Using stl

```cpp
class Solution {
public:
    int strStr(string haystack, string needle) {
        return haystack.find(needle);
    }
};
```

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

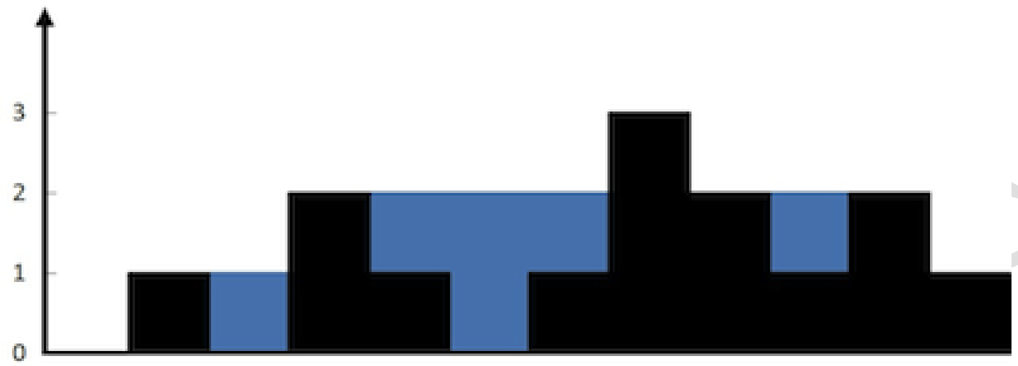
nums is a non-decreasing array.

-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]

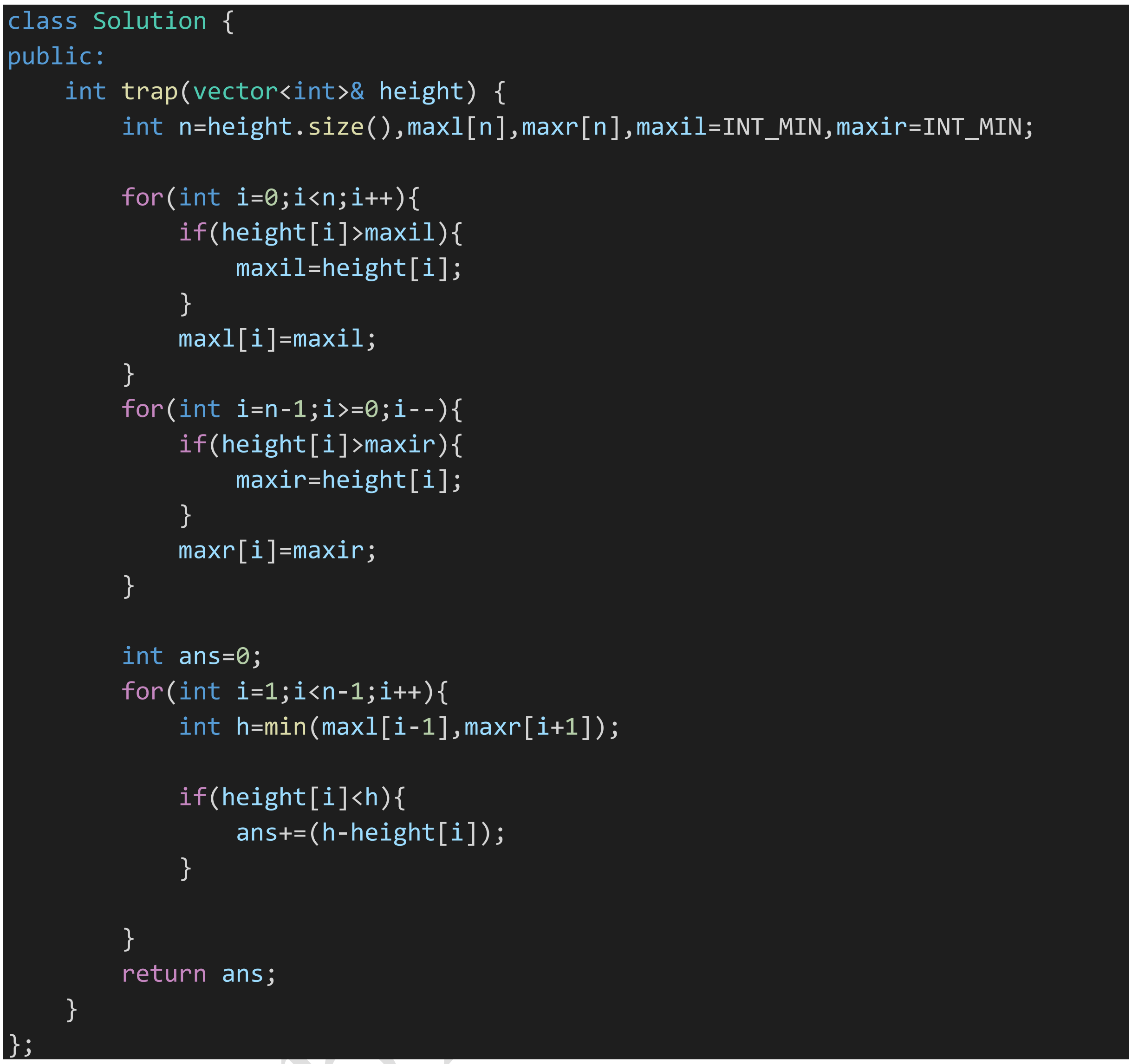
Output: 9

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]]

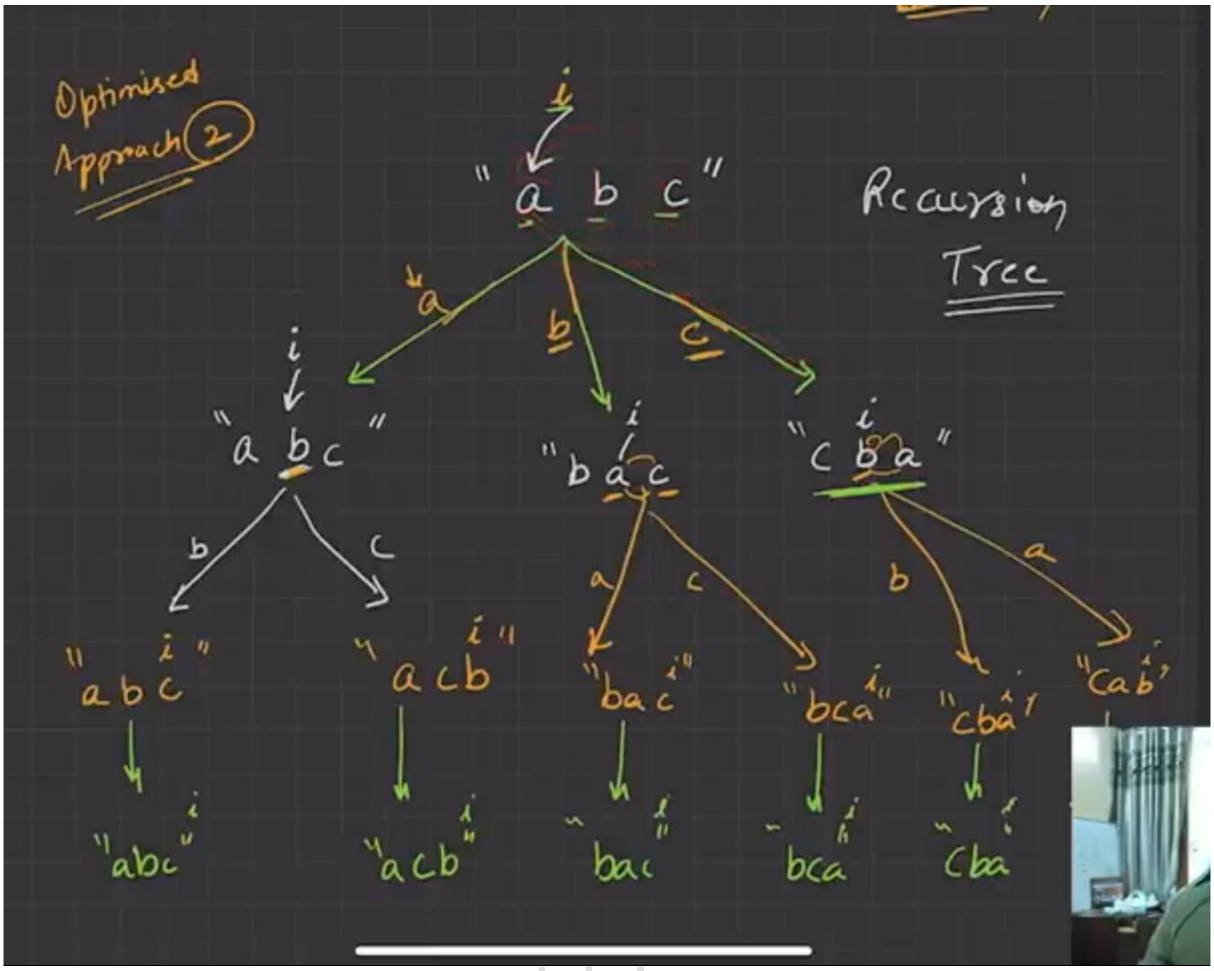
Constraints:

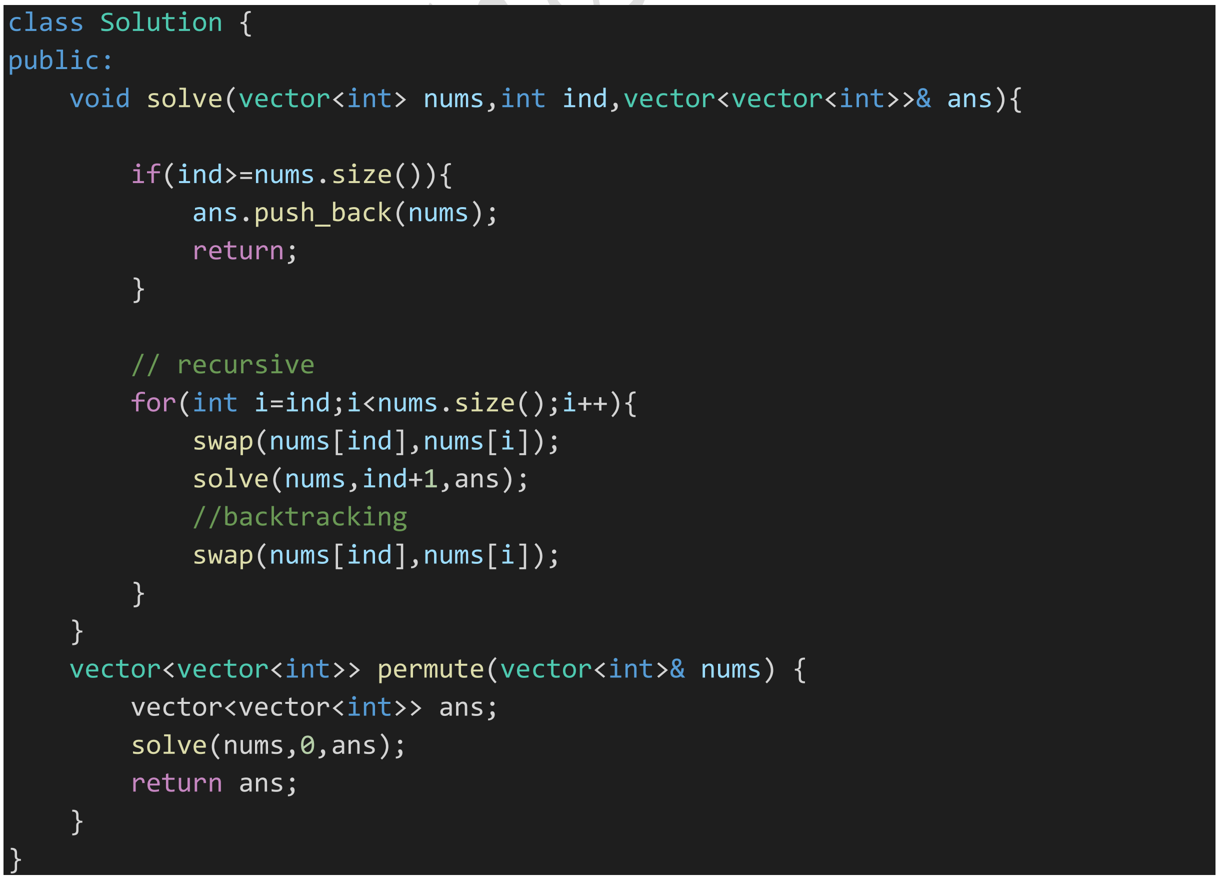
- 1 <= nums.length <= 6
- -10 <= nums[i] <= 10
- All the integers of nums are **unique**.

Idea :

1) Try putting every character at every position while simulation

2) Number of permutations will be n!



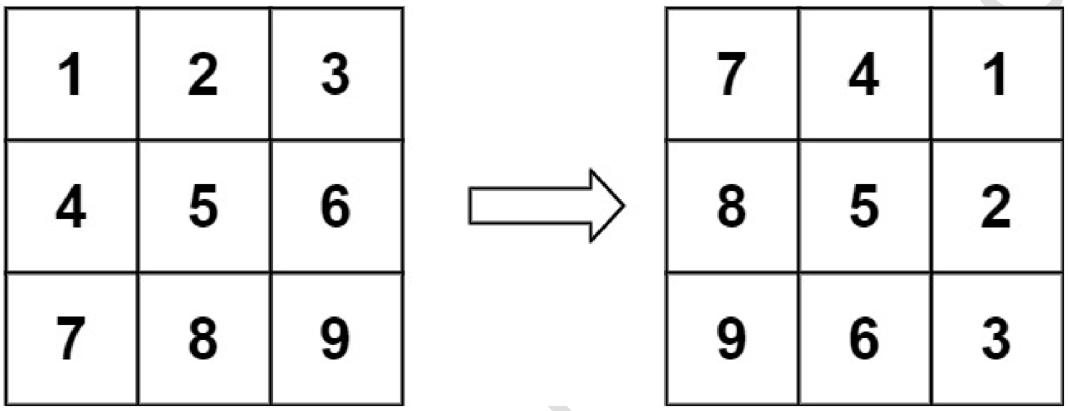

```
class Solution {
public:
    void solve(vector<int> nums,int ind,vector<vector<int>>& ans){

        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.

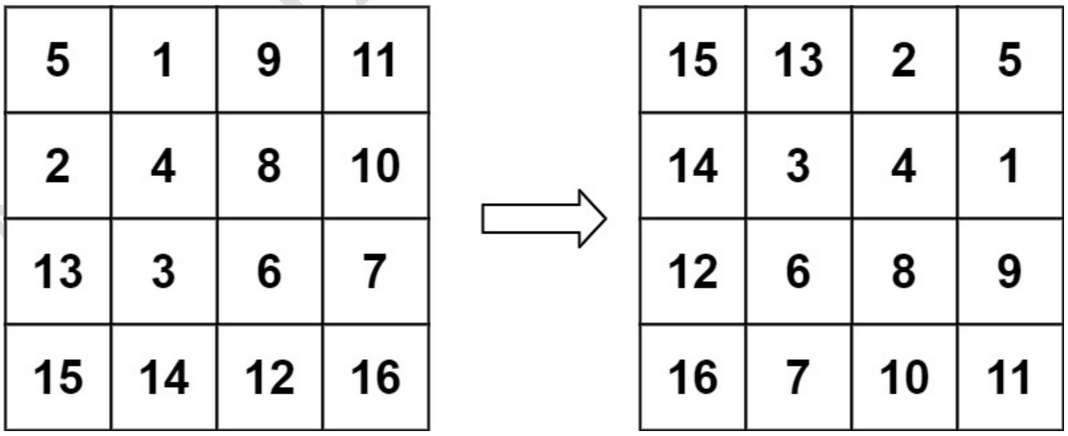
Example 1:



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

```
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.

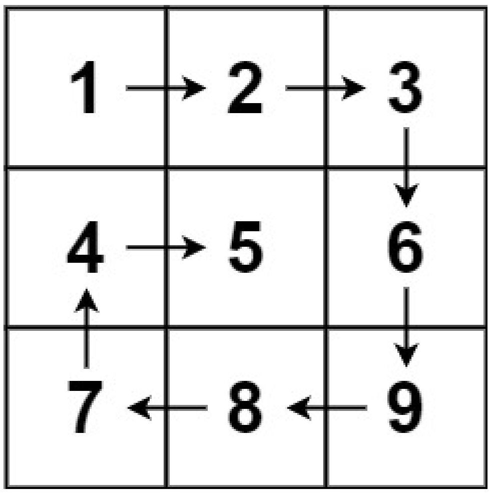
```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int max=INT_MIN,max_here=0;

        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 1:00:00]

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

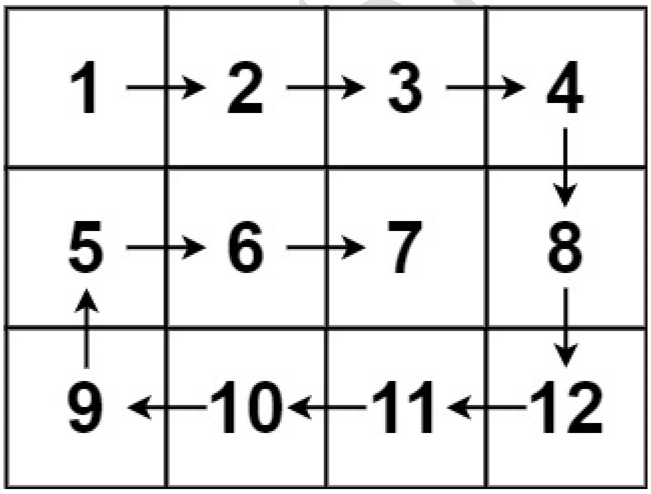
Example 1:



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.

```
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.

```cpp
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;
    }
};
```

78. Subsets Medium

Given an integer array nums of **unique** elements, return *all possible*

subsets

(the power set).

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

Input: nums = [1,2,3]

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

Example 2:

Input: nums = [0]

Output: [[],[0]]

Constraints:

1 <= nums.length <= 10

-10 <= nums[i] <= 10

All the numbers of nums are **unique**.

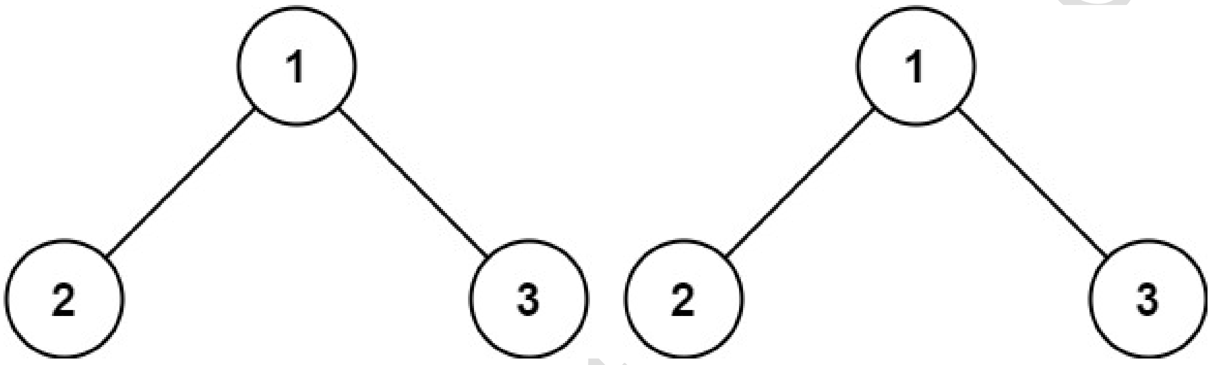
```
class Solution {
public:
    vector<vector<int>> solve(vector<int> &nums,int ind,vector<int> output,
vector<vector<int>> &ans){
        if(ind >=nums.size()){
            ans.push_back(output);
            return ans;
        }

        //exclude
        solve(nums,ind+1,output,ans);

        //include
        output.push_back(nums[ind]);
        solve(nums,ind+1,output,ans);

        return ans;
    }
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> output;

        return solve(nums,0,output,ans);
    }
};
```

100. Same Tree (Easy)

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

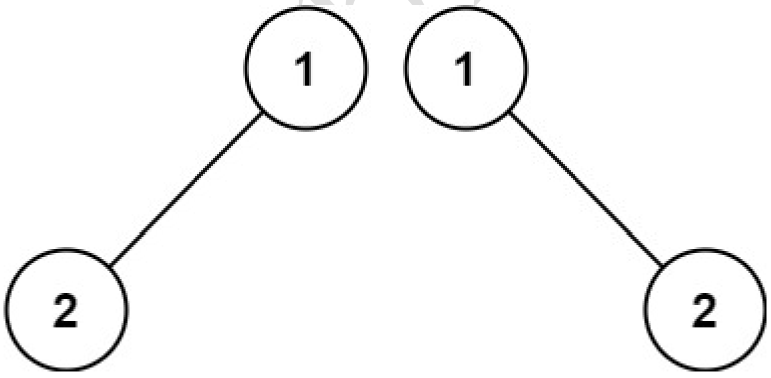
Example 1:



Input: p = [1,2,3], q = [1,2,3]

Output: true

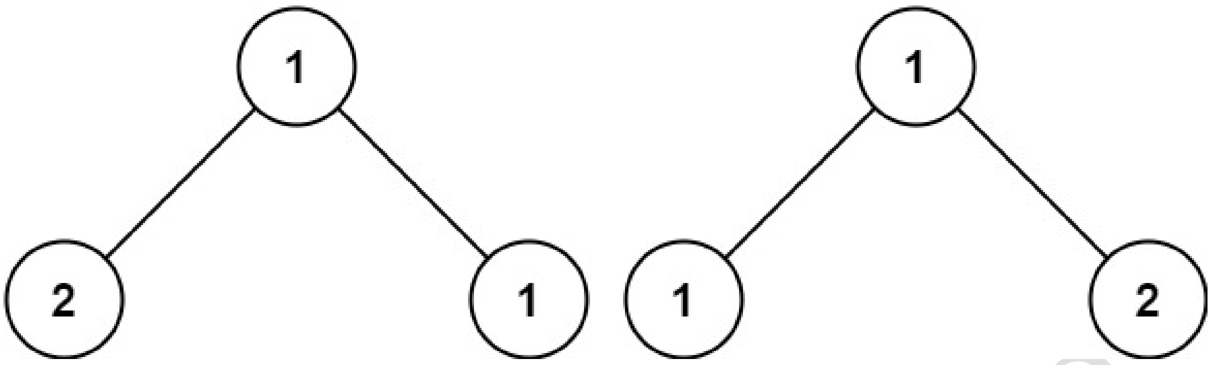
Example 2:



Input: p = [1,2], q = [1,null,2]

Output: false

Example 3:



Input: p = [1,2,1], q = [1,1,2]

Output: false

Constraints:

The number of nodes in both trees is in the range [0, 100].

$-10^4 <= Node.val <= 10^4$

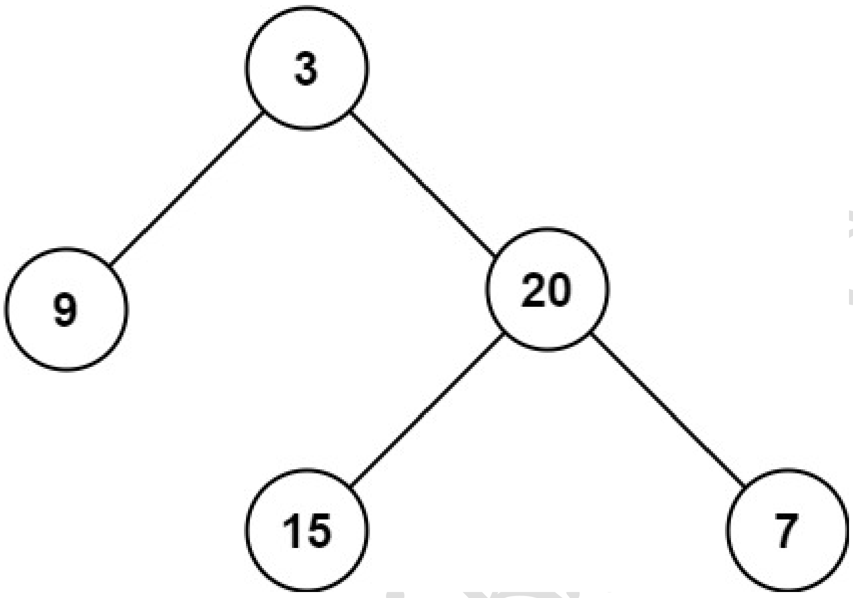
```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL && q==NULL){
            return true;
        }
        if(p==NULL && q!=NULL){
            return false;
        }
        if(p!=NULL && q==NULL){
            return false;
        }
        bool left=isSameTree(p->left,q->left);
        bool right=isSameTree(p->right,q->right);
        bool val=p->val==q->val;

        if(left && right && val){
            return true;
        }
        return false;
    }
};
```

104. Maximum Depth of Binary Tree (Easy)

Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: 3

Example 2:

Input: root = [1,null,2]

Output: 2

Constraints:

- The number of nodes in the tree is in the range [0, 10^4].
- -100 <= Node.val <= 100

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==NULL){
            return 0;
        }

        int left=maxDepth(root->left);
        int right=maxDepth(root->right);

        return max(left,right)+1;
    }
};
```

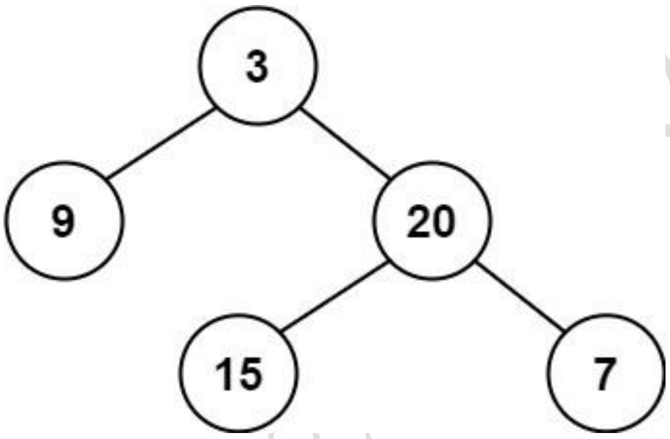
110. Balanced Binary Tree (Easy)

Given a binary tree, determine if it is

height-balanced

A height-balanced binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

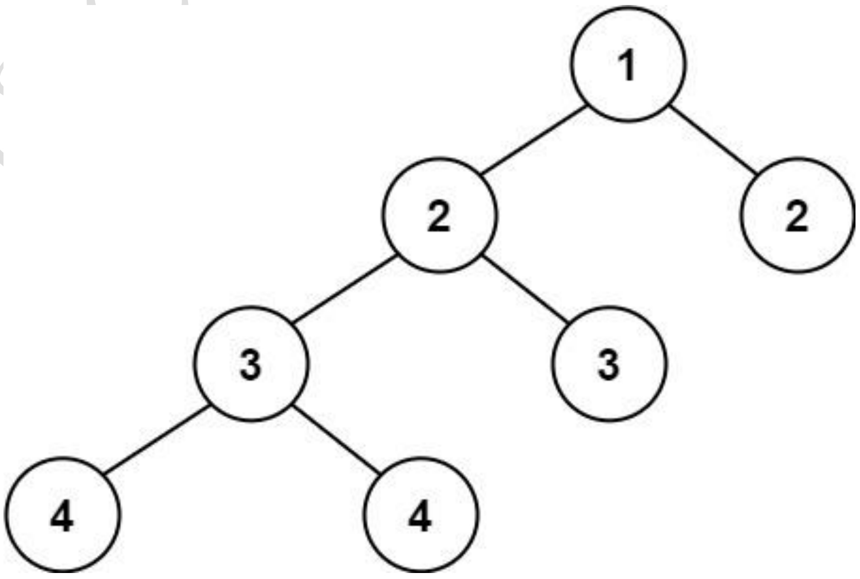
Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: true

Example 2:



Input: root = [1,2,2,3,3,null,null,4,4]

Output: false

Example 3:

Input: root = []

Output: true

Constraints:

- The number of nodes in the tree is in the range [0, 5000].
- -10^4 <= Node.val <= 10^4

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    int height(TreeNode* root) {
        if(root ==NULL){
            return 0;
        }

        int left=height(root->left);
        int right=height(root->right);

        return max(left,right)+1;
    }

    bool isBalanced(TreeNode* root) {
        if(root==NULL){
            return true;
        }
        bool left=isBalanced(root->left);
        bool right=isBalanced(root->right);
        bool val=(abs(height(root->left)-height(root->right)))<=1;

        if(left && right && val){
            return true;
        }
        return false;
    }
};
```

```
    pair<bool,int> isBalancedfast(TreeNode* root) {
        if(root==NULL){
            pair<bool,int> p=make_pair(true,0);
            return p;
        }

        pair<bool,int> left=isBalancedfast(root->left);
        pair<bool,int> right=isBalancedfast(root->right);
        bool valu=abs(left.second-right.second)<=1;

        pair<bool,int> ans;
        ans.second=max(left.second,right.second)+1;
        ans.first=left.first && right.first && valu;
        return ans;
    }

    bool isBalanced(TreeNode* root) {
        return isBalancedfast(root).first;
    }
```

121. Best Time to Buy and Sell Stock Easy

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

Constraints:

- 1 <= prices.length <= 10^5
- 0 <= prices[i] <= 10^4

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        long ansmaxi=INT_MIN,maxright=INT_MIN,cur=0;
        for(int i=prices.size()-1;i>=0;i--){
            if(prices[i]>maxright){
                maxright=prices[i];
            }
            cur=maxright-prices[i];
            if(cur>ansmaxi){
                ansmaxi=cur;
            }
        }
        return ansmaxi;
    }
};
```

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;
    }
};
```

198. House Robber (Medium)

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

Example 1:

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

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: nums = [2,7,9,3,1]

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

Constraints:

1 <= nums.length <= 100

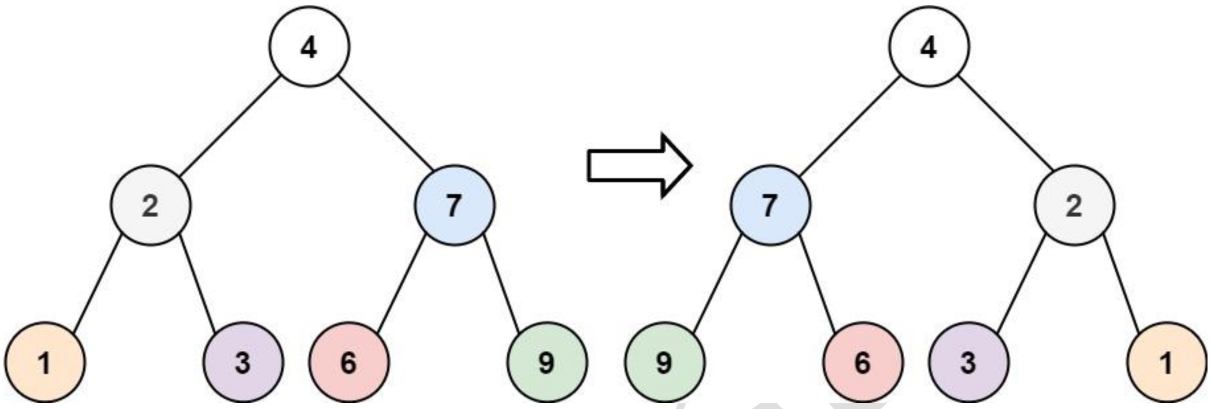
0 <= nums[i] <= 400

```
class Solution {
public:
    int robrec(vector<int>& nums, int n,int sum,vector<int> &temp){
        if(n<0){
            return 0;
        }
        if(temp[n]!=-1){
            return temp[n];
        }
        int taken=robrec(nums,n-2,sum,temp)+nums[n];
        int nott=robrec(nums,n-1,sum,temp);

        return temp[n]=sum+max(taken,nott);
    }

    int rob(vector<int>& nums) {
        vector<int> temp(nums.size(),-1);
        return robrec(nums,nums.size()-1,0,temp);
    }
};
```

226. Invert Binary Tree Easy

Given the root of a binary tree, invert the tree, and return *its root*.

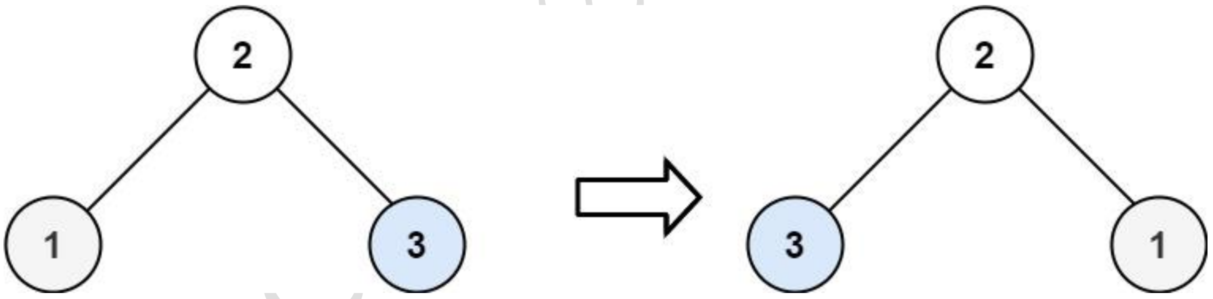
Example 1:



Input: root = [4,2,7,1,3,6,9]

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

Example 2:



Input: root = [2,1,3]

Output: [2,3,1]

Example 3:

Input: root = []

Output: []

Constraints:

The number of nodes in the tree is in the range [0, 100].

-100 <= Node.val <= 100

Swapping similar to array

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    void invert(TreeNode* root){
        if(root==NULL){
            return;
        }
        swap(root->left,root->right);
        invert(root->left);
        invert(root->right);
    }

    TreeNode* invertTree(TreeNode* root) {
        invert(root);
        return root;
    }
};
```

258. Add Digits (Easy)

Given an integer num, repeatedly add all its digits until the result has only one digit, and return it.

Example 1:

Input: num = 38

Output: 2

Explanation: The process is

38 --> 3 + 8 --> 11

11 --> 1 + 1 --> 2

Since 2 has only one digit, return it.

Example 2:

Input: num = 0

Output: 0

Constraints:

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

Follow up: Could you do it without any loop/recursion in O(1) runtime

```cpp
class Solution {
public:
    int addDigits(int num) {
        int t=num;

        while(t>9){
            int temp=t,d=0;
            while(temp>0){
                d=d+(temp%10);
                temp=temp/10;
            }
            t=d;
        }
        return t;
    }
};
```

268. Missing Number (Easy)

Given an array nums containing n distinct numbers in the range [0, n], return *the only number in the range that is missing from the array.*

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

Constraints:

- n == nums.length
- 1 <= n <= 10^4
- 0 <= nums[i] <= n
- All the numbers of nums are **unique**.

Follow up: Could you implement a solution using only O(1) extra space complexity and O(n) runtime complexity?

```cpp
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int n=nums.size();

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

        for(int i=0;i<n;i++){
            if(i!=nums[i]){
                return i;
            }
        }
        return n;
    }
};
```

338. Counting Bits (Easy)

Given an integer n, return *an array* ans *of length* n + 1 *such that for each* i (0 <= i <= n)*,* ans[i] *is the* ***number of*** 1***'s*** *in the binary representation of* i.

Example 1:

Input: n = 2

Output: [0,1,1]

Explanation:

0 --> 0

1 --> 1

2 --> 10

Example 2:

Input: n = 5

Output: [0,1,1,2,1,2]

Explanation:

0 --> 0

1 --> 1

2 --> 10

3 --> 11

4 --> 100

5 --> 101

Constraints:

- 0 <= n <= 10^5

Follow up:

- It is very easy to come up with a solution with a runtime of O(n log n). Can you do it in linear time O(n) and possibly in a single pass?
- Can you do it without using any built-in function (i.e., like __builtin_popcount in C++)?

```cpp
class Solution {
public:
    int bitcount(int n){
        int c=0;
        while(n>0){
            if(n&1){
                c++;
            }
            n=n>>1;
        }

        return c;
    }

    vector<int> countBits(int n) {
        vector<int> ans;

        for(int i=0;i<=n;i++){
            ans.push_back(bitcount(i));
        }
        return ans;
    }
};
```

374. Guess Number Higher or Lower (Easy)

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I will tell you whether the number I picked is higher or lower than your guess.

You call a pre-defined API int guess(int num), which returns three possible results:

-1: Your guess is higher than the number I picked (i.e. num > pick).

1: Your guess is lower than the number I picked (i.e. num < pick).

0: your guess is equal to the number I picked (i.e. num == pick).

Return *the number that I picked*.

Example 1:

Input: n = 10, pick = 6

Output: 6

Example 2:

Input: n = 1, pick = 1

Output: 1

Example 3:

Input: n = 2, pick = 1

Output: 1

Constraints:

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

1 <= pick <= n

```cpp
/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return       -1 if num is higher than the picked number
 *                1 if num is lower than the picked number
 *               otherwise return 0
 * int guess(int num);
 */

class Solution {
public:
    int guessNumber(int n) {
        long long s=1,e=n,mid;

        while(s<e){
            mid=s+e;
            mid=mid>>1;
            if(guess(mid)==0){
                return mid;
            }
            else if(guess(mid)<0){
                e=mid;
            }
            else{
                s=mid+1;
            }
        }
        return n;
    }
};
```

389. Find the Difference (Easy)

You are given two strings s and t.

String t is generated by random shuffling string s and then add one more letter at a random position.

Return the letter that was added to t.

Example 1:

Input: s = "abcd", t = "abcde"

Output: "e"

Explanation: 'e' is the letter that was added.

Example 2:

Input: s = "", t = "y"

Output: "y"

Constraints:

0 <= s.length <= 1000

t.length == s.length + 1

s and t consist of lowercase English letters.

```cpp
class Solution {
public:
    char findTheDifference(string s, string t) {
        int st=0,tt=0;

        for(auto i:s){
            st+=(int)i;
        }

        for(auto i:t){
            tt+=(int)i;
        }

        return tt-st;
    }
};
```

342. Power of Four (Easy)

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

An integer n is a power of four, if there exists an integer x such that n == 4^x.

Example 1:

Input: n = 16

Output: true

Example 2:

Input: n = 5

Output: false

Example 3:

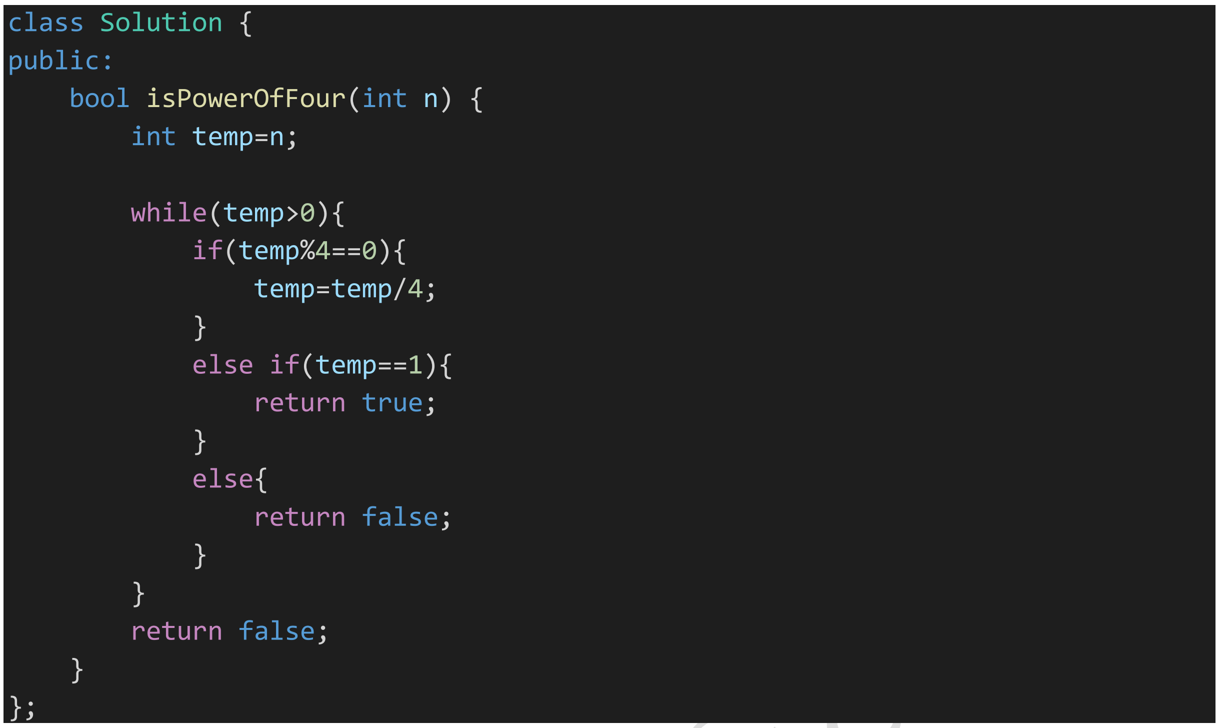
Input: n = 1

Output: true

Constraints:

- -2^{31} <= n <= 2^{31} - 1

Follow up: Could you solve it without loops/recursion?



```cpp
class Solution {
public:
    bool isPowerOfFour(int n) {
        int temp=n;

        while(temp>0){
            if(temp%4==0){
                temp=temp/4;
            }
            else if(temp==1){
                return true;
            }
            else{
                return false;
            }
        }
        return false;
    }
};
```

412. Fizz Buzz (Easy)

Given an integer n, return *a string array* answer ***(1-indexed)*** *where*:

- answer[i] == "FizzBuzz" if i is divisible by 3 and 5.
- answer[i] == "Fizz" if i is divisible by 3.
- answer[i] == "Buzz" if i is divisible by 5.
- answer[i] == i (as a string) if none of the above conditions are true.

Example 1:

Input: n = 3

Output: ["1","2","Fizz"]

Example 2:

Input: n = 5

Output: ["1","2","Fizz","4","Buzz"]

Example 3:

Input: n = 15

Output:
["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","FizzBuzz"]

Constraints:

- 1 <= n <= 10^4

```cpp
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        vector<string> ans(n);

        for(int i=0;i<n;i++){
            if((i+1)%3==0 && (i+1)%5==0){
                ans[i]="FizzBuzz";
            }
            else if((i+1)%3==0){
                ans[i]="Fizz";
            }
            else if((i+1)%5==0){
                ans[i]="Buzz";
            }
            else{
                ans[i]=to_string(i+1);
            }
        }

        return ans;
    }
};
```

415. Add Strings Easy

Given two non-negative integers, num1 and num2 represented as string, return *the sum of* num1 *and* num2 *as a string*.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

Example 1:

Input: num1 = "11", num2 = "123"

Output: "134"

Example 2:

Input: num1 = "456", num2 = "77"

Output: "533"

Example 3:

Input: num1 = "0", num2 = "0"

Output: "0"

Constraints:

- 1 <= num1.length, num2.length <= 10^4
- num1 and num2 consist of only digits.
- num1 and num2 don't have any leading zeros except for the zero itself.

```cpp
class Solution {
public:
    string addStrings(string num1, string num2) {
        int n1=num1.length()-1,n2=num2.length()-1,carry=0;
        string ans;
        while(n1>=0 && n2>=0){
            int t1=int(num1[n1--])-int('0'),t2=int(num2[n2--])-int('0');
            int sum=t1+t2+carry;
            if(sum>9){
                carry=sum/10;
                ans.push_back(char(sum%10+48));
            }
            else{
                carry=0;
                ans.push_back(char(sum+48));
            }
        }
        while(n1>=0){
            int t1=int(num1[n1--])-int('0');
            int sum=t1+carry;
            if(sum>9){
                carry=sum/10;
                ans.push_back(char(sum%10+48));
            }
            else{
                carry=0;
                ans.push_back(char(sum+48));
            }
        }
        while(n2>=0){
            int t2=int(num2[n2--])-int('0');
            int sum=t2+carry;
            if(sum>9){
                carry=sum/10;
                ans.push_back(char(sum%10+48));
            }
            else{
                carry=0;
                ans.push_back(char(sum+48));
            }
        }
        if(carry>0){
            ans.push_back(char(carry+48));
        }
        reverse(ans.begin(),ans.end());
        return ans;
    }
};
```

448. Find All Numbers Disappeared in an Array (Easy)

Given an array nums of n integers where nums[i] is in the range [1, n], return *an array of all the integers in the range* [1, n] *that do not appear in* nums.

Example 1:

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

Output: [5,6]

Example 2:

Input: nums = [1,1]

Output: [2]

Constraints:

n == nums.length

1 <= n <= 10^5

1 <= nums[i] <= n

Follow up: Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

```
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        vector<int> ans;
        sort(nums.begin(),nums.end());
        int cur=1;

        for(int i=0;i<nums.size();i++){
            if(nums[i]==cur){
                cur++;
            }
            else if(nums[i]>cur){
                while(cur<nums[i]){
                    ans.push_back(cur++);
                }
                cur++;
            }
        }

        while(nums.size()>=cur && nums[nums.size()-1]<=cur){
            ans.push_back(cur++);
        }

        return ans;
    }
};
```

455. Assign Cookies (Easy)

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input: g = [1,2,3], s = [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Example 2:

Input: g = [1,2], s = [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.

You have 3 cookies and their sizes are big enough to gratify all of the children,

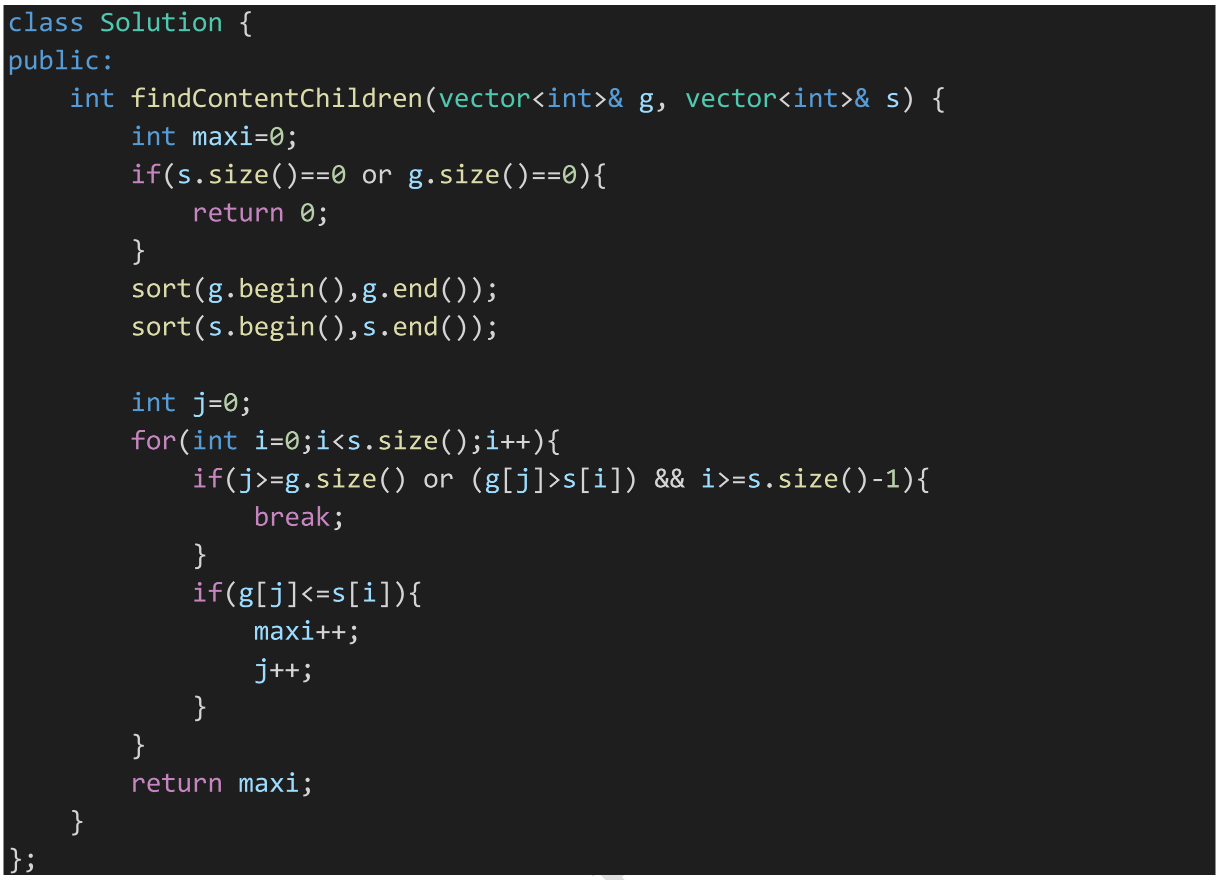
You need to output 2.

Constraints:

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

$0 <= s.length <= 3 * 10^4$

$1 <= g[i], s[j] <= 2^{31} - 1$



```cpp
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
        int maxi=0;
        if(s.size()==0 or g.size()==0){
            return 0;
        }
        sort(g.begin(),g.end());
        sort(s.begin(),s.end());

        int j=0;
        for(int i=0;i<s.size();i++){
            if(j>=g.size() or (g[j]>s[i]) && i>=s.size()-1){
                break;
            }
            if(g[j]<=s[i]){
                maxi++;
                j++;
            }
        }
        return maxi;
    }
};
```

485. Max Consecutive Ones (Easy)

Given a binary array nums, return *the maximum number of consecutive* 1's *in the array*.

Example 1:

Input: nums = [1,1,0,1,1,1]

Output: 3

Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3.

Example 2:

Input: nums = [1,0,1,1,0,1]

Output: 2

Constraints:

1 <= nums.length <= 10^5

nums[i] is either 0 or 1.

```cpp
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int maxi=INT_MIN,c=0;
        bool flag=false;

        for(int i=0;i<nums.size();i++){
            if(flag){
                if(nums[i]==0){
                    if(maxi<c){
                        maxi=c;
                    }
                    c=0;
                }
                else{
                    c++;
                }
            }
            else if(nums[i]==1){
                    c++;
                    flag=true;
            }
        }
        if(maxi<c){
            maxi=c;
        }
        return maxi;
    }
};
```

495. Teemo Attacking (Easy)

Our hero Teemo is attacking an enemy Ashe with poison attacks! When Teemo attacks Ashe, Ashe gets poisoned for a exactly duration seconds. More formally, an attack at second t will mean Ashe is poisoned during the **inclusive** time interval [t, t + duration - 1]. If Teemo attacks again **before** the poison effect ends, the timer for it is **reset**, and the poison effect will end duration seconds after the new attack.

You are given a **non-decreasing** integer array timeSeries, where timeSeries[i] denotes that Teemo attacks Ashe at second timeSeries[i], and an integer duration.

Return *the **total** number of seconds that Ashe is poisoned*.

Example 1:

Input: timeSeries = [1,4], duration = 2

Output: 4

Explanation: Teemo's attacks on Ashe go as follows:

- At second 1, Teemo attacks, and Ashe is poisoned for seconds 1 and 2.

- At second 4, Teemo attacks, and Ashe is poisoned for seconds 4 and 5.

Ashe is poisoned for seconds 1, 2, 4, and 5, which is 4 seconds in total.

Example 2:

Input: timeSeries = [1,2], duration = 2

Output: 3

Explanation: Teemo's attacks on Ashe go as follows:

- At second 1, Teemo attacks, and Ashe is poisoned for seconds 1 and 2.

- At second 2 however, Teemo attacks again and resets the poison timer. Ashe is poisoned for seconds 2 and 3.

Ashe is poisoned for seconds 1, 2, and 3, which is 3 seconds in total.

Constraints:

1 <= timeSeries.length <= 10^4

0 <= timeSeries[i], duration <= 10^7

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

```
class Solution {
public:
    int findPoisonedDuration(vector<int>& timeSeries, int duration) {
        int ans=0,n=timeSeries.size();

        for(int i=n-1;i>=0;i--){
            if(i!=n-1 && timeSeries[i]+duration>timeSeries[i+1]){
                ans+=timeSeries[i+1]-timeSeries[i];
            }
            else{
                ans+=duration;
            }
        }
        return ans;
    }
};
```

500. Keyboard Row (Easy)

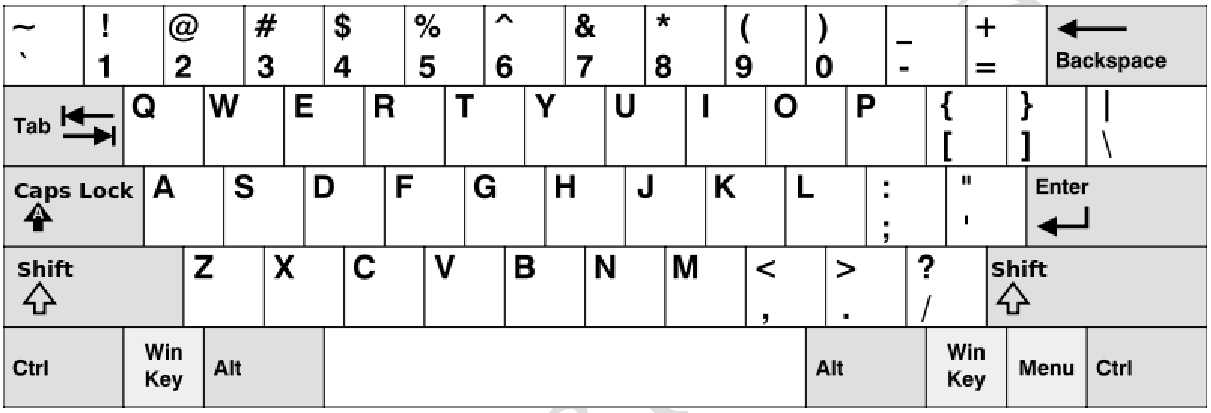
Given an array of strings words, return *the words that can be typed using letters of the alphabet on only one row of American keyboard like the image below*.

In the **American keyboard**:

the first row consists of the characters "qwertyuiop",

the second row consists of the characters "asdfghjkl", and

the third row consists of the characters "zxcvbnm".


~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Backspace
Tab Q W E R T Y U I O P { [}] | \
Caps Lock A S D F G H J K L : ; " ' Enter
Shift Z X C V B N M < , > . ? / Shift
Ctrl Win Key Alt Alt Win Key Menu Ctrl

Example 1:

Input: words = ["Hello","Alaska","Dad","Peace"]

Output: ["Alaska","Dad"]

Example 2:

Input: words = ["omk"]

Output: []

Example 3:

Input: words = ["adsdf","sfd"]

Output: ["adsdf","sfd"]

Constraints:

1 <= words.length <= 20

1 <= words[i].length <= 100

words[i] consists of English letters (both lowercase and uppercase).

```
class Solution {
public:
    char tolower(char ch){

        if(ch>='A' && ch<='Z'){
            ch=ch-'A'+'a';
        }

        return ch;
    }

    bool solve(string s1,string s2){
        bool flag=false;
        for(int i=0;i<s1.length();i++){
            flag=false;
            for(int j=0;j<s2.length();j++){
                if(tolower(s1[i])==s2[j]){
                    flag=true;
                    break;
                }
            }
            if(flag==false){
                return false;
            }
        }
        return true;
    }

    vector<string> findWords(vector<string>& words) {
        vector<string> ans;
        vector<string> mapi={"qwertyuiop","asdfghjkl","zxcvbnm"};

        // word in checking
        for(int i=0;i<words.size();i++){
            //check with map
            for(int j=0;j<3;j++){
                if(solve(words[i],mapi[j])){
                    ans.push_back(words[i]);
                    break;
                }
            }
        }
        return ans;
    }
};
```

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];
    }
};
```

540. Single Element in a Sorted Array

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once*.

Your solution must run in O(log n) time and O(1) space.

Example 1:

Input: nums = [1,1,2,3,3,4,4,8,8]

Output: 2

Example 2:

Input: nums = [3,3,7,7,10,11,11]

Output: 10

Constraints:

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

```
class Solution {
public:
    int singleNonDuplicate(vector<int>& nums) {
        int size=nums.size(),s=0,e=size-1,mid=0;
        if(size==1){
            return nums[0];
        }
        if(nums[0]!=nums[1]){
            return nums[0];
        }
        if(nums[size-1]!=nums[size-2]){
            return nums[size-1];
        }
        while(s<=e){
            mid=s+(e-s)/2;S
            if(mid>0 && mid<size-1 && nums[mid]!=nums[mid-1] &&
nums[mid]!=nums[mid+1]){
                return nums[mid];
            }
            else if(nums[mid]==nums[mid-1]){
                if((size-mid)&1){
                    e=mid-1;
                }
                else{
                    s=mid+1;
                }
            }
            else{
                if((size-mid)&1){
                    s=mid+1;
                }
                else{
                    e=mid-1;
                }
            }
        }

        return -1;
    }
};
```

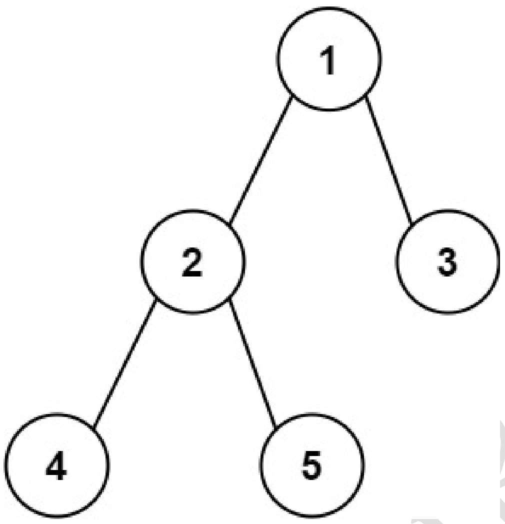
543. Diameter of Binary Tree (Easy)

Given the root of a binary tree, return *the length of the **diameter** of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:



Input: root = [1,2,3,4,5]

Output: 3

Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Example 2:

Input: root = [1,2]

Output: 1

Constraints:

The number of nodes in the tree is in the range [1, 10^4].

-100 <= Node.val <= 100

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    int height(TreeNode* root) {
        if(root ==NULL){
            return 0;
        }

        int left=height(root->left);
        int right=height(root->right);

        return max(left,right)+1;
    }

    int diameterOfBinaryTree(TreeNode* root) {
        if(root ==NULL){
            return 0;
        }

        int left=diameterOfBinaryTree(root->left);
        int right=diameterOfBinaryTree(root->right);
        int combi=height(root->left)+height(root->right);

        return max(left,max(combi,right));
    }
    pair<int,int> diafast(TreeNode* root) {
        if(root ==NULL){
            pair<int,int> p=make_pair(0,0);
            return p;
        }

        pair<int,int> left=diafast(root->left);
        pair<int,int> right=diafast(root->right);
        pair<int,int> ans=make_pair(0,0);
        int combi=left.second+right.second;

        ans.first= max(left.first,max(combi,right.first));
        ans.second=max(left.second,right.second)+1;
        return ans;
    }

    int diameterOfBinaryTree(TreeNode* root) {
        return diafast(root).first;
    }

};
```

605. Can Place Flowers Easy

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in **adjacent** plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return *if* n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule.

Example 1:

Input: flowerbed = [1,0,0,0,1], n = 1

Output: true

Example 2:

Input: flowerbed = [1,0,0,0,1], n = 2

Output: false

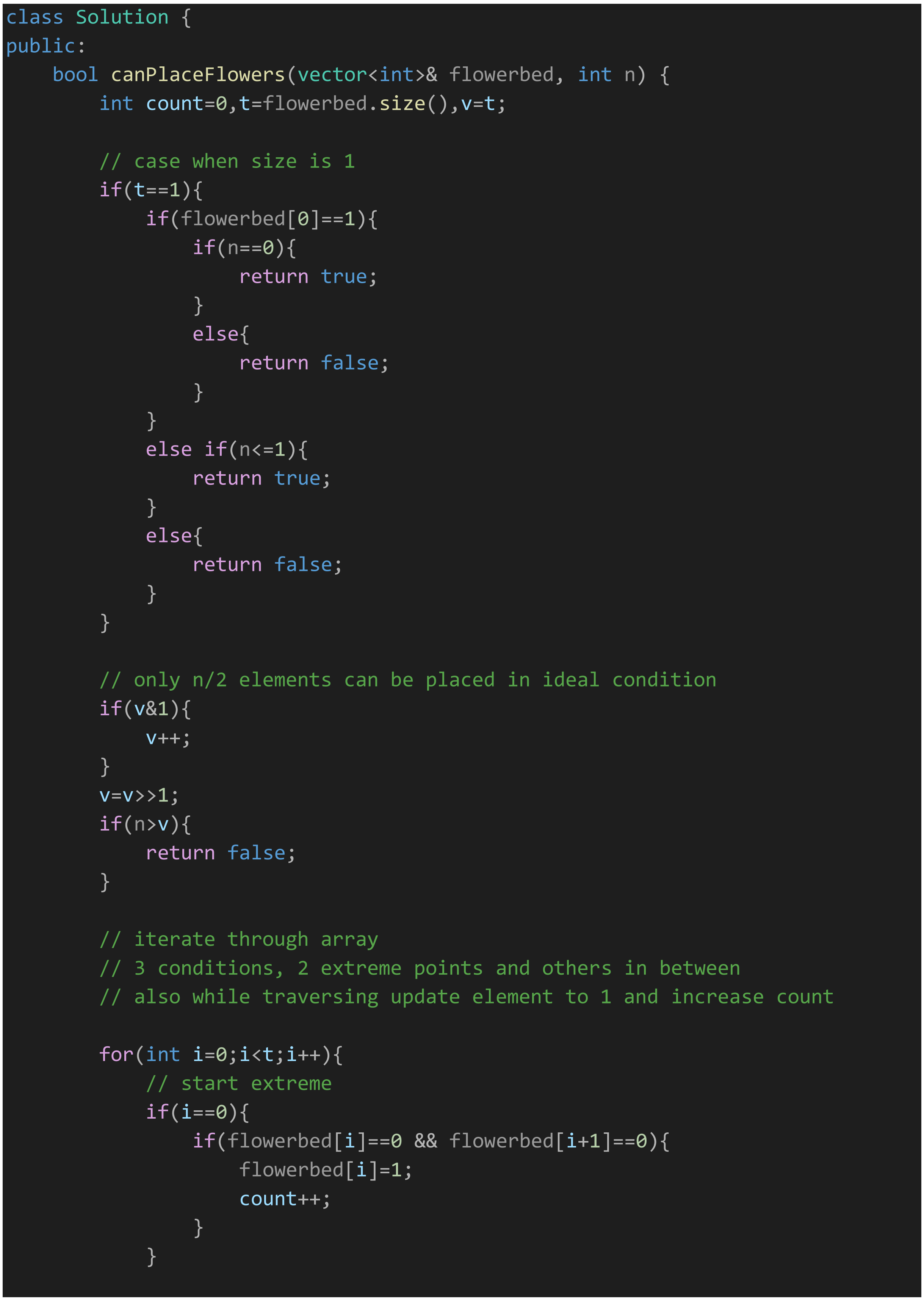
Constraints:

1 <= flowerbed.length <= 2 * 10^4

flowerbed[i] is 0 or 1.

There are no two adjacent flowers in flowerbed.

0 <= n <= flowerbed.length



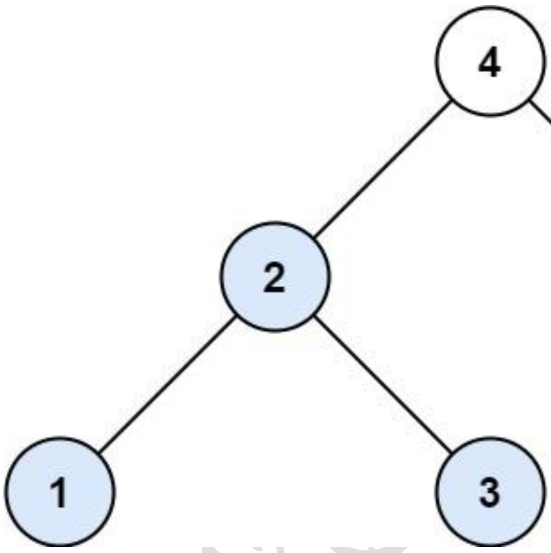
```cpp
class Solution {
public:
    bool canPlaceFlowers(vector<int>& flowerbed, int n) {
        int count=0,t=flowerbed.size(),v=t;

        // case when size is 1
        if(t==1){
            if(flowerbed[0]==1){
                if(n==0){
                    return true;
                }
                else{
                    return false;
                }
            }
            else if(n<=1){
                return true;
            }
            else{
                return false;
            }
        }

        // only n/2 elements can be placed in ideal condition
        if(v&1){
            v++;
        }
        v=v>>1;
        if(n>v){
            return false;
        }

        // iterate through array
        // 3 conditions, 2 extreme points and others in between
        // also while traversing update element to 1 and increase count

        for(int i=0;i<t;i++){
            // start extreme
            if(i==0){
                if(flowerbed[i]==0 && flowerbed[i+1]==0){
                    flowerbed[i]=1;
                    count++;
                }
            }
            // end extreme
            else if(i==t-1){
                if(flowerbed[i]==0 && flowerbed[i-1]==0){
                    flowerbed[i]=1;
                    count++;
                }
            }
            // middle elements
            else{
                if(flowerbed[i]==0 && flowerbed[i-1]==0 && flowerbed[i+1]==0){
                    flowerbed[i]=1;
                    count++;
                }
            }
        }

        // when count is greater or equal than required, return true
        if(count>=n){
            return true;
        }
        return false;
    }
};
```

700. Search in a Binary Search Tree Easy

You are given the root of a binary search tree (BST) and an integer val.

Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null.

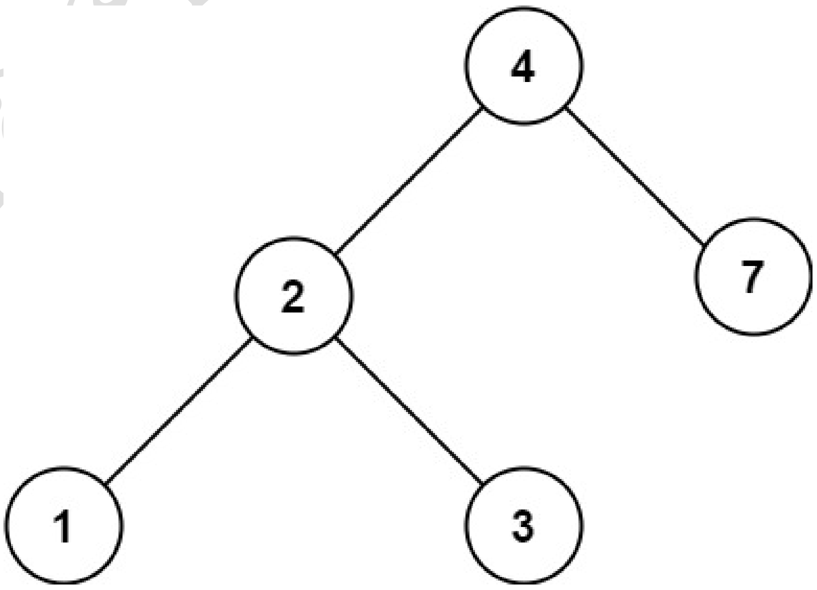
Example 1:



Input: root = [4,2,7,1,3], val = 2

Output: [2,1,3]

Example 2:



Input: root = [4,2,7,1,3], val = 5

Output: []

Constraints:

The number of nodes in the tree is in the range [1, 5000].

1 <= Node.val <= 10^7

root is a binary search tree.

1 <= val <= 10^7

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* searchBST(TreeNode* root, int val) {
        if(root==NULL){
            return NULL;
        }
        if(root->val==val){
            return root;
        }
        else if(root->val<val){
            return searchBST(root->right,val);
        }
        else{
            return searchBST(root->left,val);
        }
        return NULL;
    }
};
```

709. To Lower Case (Easy)

Given a string s, return *the string after replacing every uppercase letter with the same lowercase letter*.

Example 1:

Input: s = "Hello"

Output: "hello"

Example 2:

Input: s = "here"

Output: "here"

Example 3:

Input: s = "LOVELY"

Output: "lovely"

Constraints:

1 <= s.length <= 100

s consists of printable ASCII characters.

```
class Solution {
public:
    string toLowerCase(string s) {
        for(int i=0;i<s.size();i++){
            if(s[i]>='A' && s[i]<='Z'){
                s[i]=s[i]-'A'+'a';
            }
        }

        return s;
    }
};
```

875. Koko Eating Bananas Medium (8-3-23, 8pm, 40min)

Koko loves to eat bananas. There are n piles of bananas, the ith pile has piles[i] bananas. The guards have gone and will come back in h hours.

Koko can decide her bananas-per-hour eating speed of k. Each hour, she chooses some pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* k *such that she can eat all the bananas within* h *hours*.

Example 1:

Input: piles = [3,6,7,11], h = 8

Output: 4

Example 2:

Input: piles = [30,11,23,4,20], h = 5

Output: 30

Example 3:

Input: piles = [30,11,23,4,20], h = 6

Output: 23

Constraints:

1 <= piles.length <= 10^4

piles.length <= h <= 10^9

1 <= piles[i] <= 10^9

Applying binary search

```
class Solution {
public:
    bool ispossible(vector<int>& piles, int h,int k){
        long temp=0;
        for(auto i:piles){
            temp=temp+(i/k)+1;
            if(i%k==0){
                temp--;
            }
        }
        // cout<<k<<" "<<temp<<" "<<h<<endl;
        if(temp<=h){
            return true;
        }
        return false;
    }

    int minEatingSpeed(vector<int>& piles, int h) {
        // if(piles.size()==1 && piles[0]%h==0){
        //     return piles[0]/h;
        // }
        long long s=1,e=*max_element(piles.begin(),piles.end()),ans=e,mid=0;
        while(s<=e){
            mid=(s+e)/2;
            bool p=ispossible(piles,h,mid);
            // cout<<s<<" "<<e<<" "<<mid<<" "<<p<<endl;
            if(p){
                e=mid-1;
                ans=mid;
            }
            else{
                s=mid+1;
            }
        }
        return ans;
    }
};
```

912. Sort an Array Medium

Given an array of integers nums, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in O(nlog(n)) time complexity and with the smallest space complexity possible.

Example 1:

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

Output: [1,2,3,5]

Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:

Input: nums = [5,1,1,2,0,0]

Output: [0,0,1,1,2,5]

Explanation: Note that the values of nums are not necessairly unique.

Constraints:

1 <= nums.length <= 5 * 10^4

-5 * 10^4 <= nums[i] <= 5 * 10^4

```
class Solution {
public:
    void merge(vector<int>& nums,int s,int mid,int e){
        int l1=mid-s+1,l2=e-mid;

        // remember to write arrays this way for deletion
        int *a = new int[l1];
        int *b = new int[l2];
        int l=s;
        // take care of length variable
        for(int i=0;i<l1;i++){
            a[i]=nums[l++];
        }
        l=mid+1;
        for(int i=0;i<l2;i++){
            b[i]=nums[l++];
        }
        int i=0,j=0,k=s;
        while(i<l1 && j<l2){
            if(a[i]<b[j]){
                nums[k]=a[i];
                k++;i++;
            }
            // remember about else otherwise TLE
            else{
                nums[k]=b[j];
                k++;j++;
            }
        }
        while(i<l1){
            nums[k]=a[i];
            k++;i++;
        }
        while(j<l2){
            nums[k]=b[j];
            k++;j++;
        }

        // delete temp
        delete []a;
        delete []b;
    }
    void mergesort(vector<int>& nums,int s,int e){
        if(s>=e){
            return;
        }
        int mid=(s+e)/2;
        mergesort(nums,s,mid);
        mergesort(nums,mid+1,e);
        merge(nums,s,mid,e);
    }

    vector<int> sortArray(vector<int>& nums) {
        mergesort(nums,0,nums.size()-1);
        return nums;
    }
};
```

1011. Capacity To Ship Packages Within D Days (Medium)

A conveyor belt has packages that must be shipped from one port to another within days days.

The i^{th} package on the conveyor belt has a weight of weights[i]. Each day, we load the ship with packages on the conveyor belt (in the order given by weights). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within days days.

Example 1:

Input: weights = [1,2,3,4,5,6,7,8,9,10], days = 5

Output: 15

Explanation: A ship capacity of 15 is the minimum to ship all the packages in 5 days like this:

1st day: 1, 2, 3, 4, 5

2nd day: 6, 7

3rd day: 8

4th day: 9

5th day: 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 14 and splitting the packages into parts like (2, 3, 4, 5), (1, 6, 7), (8), (9), (10) is not allowed.

Example 2:

Input: weights = [3,2,2,4,1,4], days = 3

Output: 6

Explanation: A ship capacity of 6 is the minimum to ship all the packages in 3 days like this:

1st day: 3, 2

2nd day: 2, 4

3rd day: 1, 4

Example 3:

Input: weights = [1,2,3,1,1], days = 4

Output: 3

Explanation:

1st day: 1

2nd day: 2

3rd day: 3

4th day: 1, 1

Constraints:

1 <= days <= weights.length <= 5 * 10^4

1 <= weights[i] <= 500

```cpp
class Solution {
public:
    bool possible(vector<int>& weights, int days,int mid){
        long long sum=0,count=1;

        for(int i=0;i<weights.size();i++){
            if(sum+weights[i]<=mid){
                sum+=weights[i];
            }
            else{
                count++;
                if(count>days or weights[i]>mid){
                    return false;
                }
                sum=weights[i];
            }
        }
        return true;
    }

    int shipWithinDays(vector<int>& weights, int days) {
        int s=0,e=0,mid=0,ans=0;
        if(days>weights.size()){
            return -1;
        }
        for(auto i:weights){
            e+=i;
        }
        while(s<=e){
            mid=(e+s)/2;
            if(possible(weights,days,mid)){
                ans=mid;
                e=mid-1;
            }
            else{
                s=mid+1;
            }
        }
        return ans;
    }
};
```

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];
    }
};
```

2187. Minimum Time to Complete Trips

Medium (7-3-23, 9pm)

You are given an array time where time[i] denotes the time taken by the ith bus to complete **one trip**.

Each bus can make multiple trips **successively**; that is, the next trip can start **immediately after** completing the current trip. Also, each bus operates **independently**; that is, the trips of one bus do not influence the trips of any other bus.

You are also given an integer totalTrips, which denotes the number of trips all buses should make **in total**. Return *the **minimum time** required for all buses to complete **at least*** totalTrips *trips*.

Example 1:

Input: time = [1,2,3], totalTrips = 5

Output: 3

Explanation:

- At time t = 1, the number of trips completed by each bus are [1,0,0].

  The total number of trips completed is 1 + 0 + 0 = 1.

- At time t = 2, the number of trips completed by each bus are [2,1,0].

  The total number of trips completed is 2 + 1 + 0 = 3.

- At time t = 3, the number of trips completed by each bus are [3,1,1].

  The total number of trips completed is 3 + 1 + 1 = 5.

So the minimum time needed for all buses to complete at least 5 trips is 3.

Example 2:

Input: time = [2], totalTrips = 1

Output: 2

Explanation:

There is only one bus, and it will complete its first trip at t = 2.

So the minimum time needed to complete 1 trip is 2.

Constraints:

1 <= time.length <= 10^5

1 <= time[i], totalTrips <= 10^7

My normal solution, brute force

```
class Solution {
public:
    long long minimumTime(vector<int>& time, int totalTrips) {
        long long ans=0;
        sort(time.begin(),time.end());
        vector<int> complete(time.size());

        while(totalTrips>0){
            for(int i=0;i<complete.size();i++){
                complete[i]=complete[i]+1;
                if(complete[i]>=time[i]){
                    totalTrips--;
                    complete[i]=0;
                }
            }
            ans++;
        }
        return ans;
    }
};
```

Using binary search

Also runtime error on data type multiplication, so needed mul function with loop and addition of number

```
class Solution {
public:
    long long mul(int a,int b){
        long long ans=0;
        if(a<b){
            swap(a,b);
        }
        while(b){
            ans+=a;
            b--;
        }
        return ans;
    }
    bool ispossible(vector<int>& time,long long n, long long totalTrips){
        long long int temp=0;
        for(auto i:time){
            temp+=(n/i);
        }
        if(temp>=totalTrips){
            return true;
        }
        return false;
    }

    long long minimumTime(vector<int>& time, int totalTrips) {
        if(time.size()==1){
            return mul(time[0],totalTrips);
        }
        long long 
ans=0,s=0,e=mul(*(min_element(time.begin(),time.end())),totalTrips),mid=0;
        while(s<=e){
            mid=(e+s)/2;
            if(ispossible(time,mid,totalTrips)){
                e=mid-1;
                ans=mid;
            }
            else{
                s=mid+1;
            }
        }
        return ans;
    }
};
```

Learned new function in c++

```
(*(min_element(time.begin(),time.end())),totalTrips)
```

2348. Number of Zero-Filled Subarrays Medium

Given an integer array nums, return *the number of* ***subarrays*** *filled with* 0.

A **subarray** is a contiguous non-empty sequence of elements within an array.

Example 1:

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

Output: 6

Explanation:

There are 4 occurrences of [0] as a subarray.

There are 2 occurrences of [0,0] as a subarray.

There is no occurrence of a subarray with a size more than 2 filled with 0. Therefore, we return 6.

Example 2:

Input: nums = [0,0,0,2,0,0]

Output: 9

Explanation:

There are 5 occurrences of [0] as a subarray.

There are 3 occurrences of [0,0] as a subarray.

There is 1 occurrence of [0,0,0] as a subarray.

There is no occurrence of a subarray with a size more than 3 filled with 0. Therefore, we return 9.

Example 3:

Input: nums = [2,10,2019]

Output: 0

Explanation: There is no subarray filled with 0. Therefore, we return 0.

Constraints:

1 <= nums.length <= 10^5

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

```cpp
class Solution {
public:
    long long zeroFilledSubarray(vector<int>& nums) {
        long long ans=0,i=0,j=0,cur=0;

        while(i<nums.size()){
            // cur is size of subpart
            cur=0;

            // find the parts of array where all elements are 0
            while(i<nums.size() && nums[i]==0){
                cur++;
                i++;
            }
            // in this the total occurence will be n*n+1/2
            ans+=((cur*(cur+1))>>1);
            // increment to the next index
            // ans will be found in o(n) traversal
            i++;
        }
        return ans;
    }
};
```

2444. Count Subarrays With Fixed Bounds Hard

You are given an integer array nums and two integers minK and maxK.

A **fixed-bound subarray** of nums is a subarray that satisfies the following conditions:

- The **minimum** value in the subarray is equal to minK.
- The **maximum** value in the subarray is equal to maxK.

Return *the **number** of fixed-bound subarrays*.

A **subarray** is a **contiguous** part of an array.

Example 1:

Input: nums = [1,3,5,2,7,5], minK = 1, maxK = 5

Output: 2

Explanation: The fixed-bound subarrays are [1,3,5] and [1,3,5,2].

Example 2:

Input: nums = [1,1,1,1], minK = 1, maxK = 1

Output: 10

Explanation: Every subarray of nums is a fixed-bound subarray. There are 10 possible subarrays.

Constraints:

- 2 <= nums.length <= 10^5
- 1 <= nums[i], minK, maxK <= 10^6

My solution (TLE)

```cpp
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        long long count=0;
        for(int i=0;i<nums.size();i++){
            bool minp=false,maxp=false,less=false,great=false;
            for(int j=i;j<nums.size();j++){
                if(nums[j]==minK){
                    minp=true;
                }
                if(nums[j]==maxK){
                    maxp=true;
                }
                if(nums[j]<minK){
                    less=true;
                }
                if(nums[j]>maxK){
                    great=true;
                }
                // count
                if((minp && maxp) && !(less) && !(great)){
                    // cout<<i<<" "<<j<<" "<<minp<<maxp<<less<<great<<endl;
                    count++;
                }
            }
        }
        return count;
    }
};
```

Other solution with sliding window

Intuition

Count the number of subarrays,
using sliding window(three pointers).

Explanation

We maintain a maximum sliding window
with alll elements in range `[minK, maxK]`,
For all `A[i]` as rightmost element of the subarray,
we find the three indices `j`, where:

jbad is index of last seen A[jbad] < minK || A[jbad] > maxK
jmin is index of last seen A[jmin] = mink
jmax is index of last seen A[jmax] = maxk

Itearte the A[i],
if A[i] < minK || A[i] > maxK, update jbad = i.
if A[i] == minK, update jmin = i.
if A[i] == maxK, update jmax = i.

In the end of each iteration,
the subarray ends at A[i]
the starting element of the subarray,
can be choosen in interval [jbad + 1, min(jmin, jmax)]
There are min(jmin, jmax) - jbad choices,
so we update res += max(0, min(jmin, jmax) - jbad).

Complexity

Time O(n)
Space O(1)

```
class Solution {
public:
    long long countSubarrays(vector<int>& nums, int minK, int maxK) {
        long long result = 0;
        int subArrayStartIndex = 0;
        int latestMinIndex = -1;
        int latestMaxIndex = -1;

        for (int i = 0; i < nums.size(); i++) {
            // invalid sub array, start over
            if  (nums[i] < minK || nums[i] > maxK) {
                latestMinIndex = -1;
                latestMaxIndex = -1;
                subArrayStartIndex = i + 1;
            }
            if (nums[i] == minK) latestMinIndex = i;
            if (nums[i] == maxK) latestMaxIndex = i;
            result += max(0, min(latestMinIndex, latestMaxIndex) -
subArrayStartIndex + 1);
        }
        return result;
    }
};
```