125. Valid Palindrome

Solved

Easy | Topics | Companies

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

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

Example 1:

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

Example 2:

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

Example 3:

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric
characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

```
bool isPalindrome(string s) {
    int left=0,right = s.size()-1;
    while(left<right){
        if(!isalnum(s[left])){
            left++;
        }
        else if(!isalnum(s[right])){
            right--;
        }
        else if(tolower(s[left])!=tolower(s[right])){
            return false;
        }else{
            left++;
            right--;
        }

    }
    return true;

}
```

Time Complexity : O(N)

Space Complexity : O(1)

Two Integer Sum II

Solved

Medium

Given an array of integers `numbers` that is sorted in **non-decreasing order**.

Return the indices (**1-indexed**) of two numbers, `[index1, index2]`, such that they add up to a given target number `target` and `index1 < index2`. Note that `index1` and `index2` cannot be equal, therefore you may not use the same element twice.

There will always be **exactly one valid solution**.

Your solution must use $O(1)$ additional space.

Example 1:

```
Input: numbers = [1,2,3,4], target = 3

Output: [1,2]
```

Explanation:
The sum of 1 and 2 is 3. Since we are assuming a 1-indexed array, `index1` = 1, `index2` = 2. We return `[1, 2]`.

Constraints:

- `2 <= numbers.length <= 1000`
- `-1000 <= numbers[i] <= 1000`
- `-1000 <= target <= 1000`

```
vector<int> twoSum(vector<int>& num, int target) {
    int n = num.size();
    int left = 0,right = n-1;
    while(left<=right){
        int sum = num[left]+num[right];

        if(sum == target){
            return {left+1,right+1};
        }
        else if(sum>target){
            right--;
        }
        else{
            left++;
        }
    }
    return {};
}
```

Time Complexity : O(N)

space Complexity : O(1)

Container With Most Water

Difficulty: **Medium** Accuracy: **53.84%** Submissions: **49K+** Points: **4**

Given non-negative integers $arr_1, arr_2, ...arr_n$ where each represents a point at coordinate (i, arr_i). For each i vertical lines are drawn such that the two endpoints of line **i** is at (i, arr_i) and (i,0). Find two lines, which together with x-axis form a container, such that it contains the most water.

Note: In the case of a single verticle line it will not be able to hold water.

Examples:

Input: arr[] = [1, 5, 4, 3]
Output: 6
Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 * 2 = 6.

Input: arr[] = [3, 1, 2, 4, 5]
Output: 12
Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12.

Input: arr[] = [2, 1, 8, 6, 4]
Output: 8
Explanation: The indices 2 (height 8) and 4 (height 4) are distance 2 apart. So the size of the base is 2. The height of the container is the minimum of 8 and 4, which is 4. Therefore, the total area is 4 * 2 = 8.

```
int maxArea(vector<int>& height) {
    int result =0;

    int left =0,right = height.size()-1;
    while(left<right){
        int area = (right-left)* min(height[left] , height[right]);
        result = max(area,result);


        if(height[left]>height[right]){
            right--;

        }else{
            left++;
        }
    }
    return result;
}
```

Time Complexity : O(n)

Space Complexity : O(1)

392. Is Subsequence

Solved

Easy | Topics | Companies

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

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

Example 1:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

Example 2:

```
Input: s = "axc", t = "ahbgdc"
Output: false
```

Constraints:

- `0 <= s.length <= 100`
- `0 <= t.length <= 10^4`
- `s` and `t` consist only of lowercase English letters.

```
bool isSubsequence(string s, string t) {
    int n = s.size();
    int m = t.size();
    if(s == "" ) return true;

    int i=0;

    for(int j=0;j<m;j++){
        if(t[j] == s[i]){
            i++;
        }
    }

    if(i == n){
        return true;
    }

    return false;
}
```

Time complexity : O(n)

Space Complexity : O(1)

Triplet Sum in Array

Difficulty: **Medium** Accuracy: **35.0%** Submissions: **306K+** Points: **4**

Given an array **arr[]** and an integer **target**, determine if there exists a triplet in the array whose sum equals the given target.

Return true if such a triplet exists, otherwise, return false.

Examples

Input: arr[] = [1, 4, 45, 6, 10, 8], target = 13
Output: true
Explanation: The triplet {1, 4, 8} sums up to 13

Input: arr[] = [1, 2, 4, 3, 6, 7], target = 10
Output: true
Explanation: The triplets {1, 3, 6} and {1, 2, 7} both sum to 10.

Input: arr[] = [40, 20, 10, 3, 6, 7], target = 24
Output: false
Explanation: No triplet in the array sums to 24

Constraints:
$3 \le arr.size() \le 10^3$
$1 \le arr[i] \le 10^5$

```
#include<bits/stdc++.h>
using namespace std;
bool hasTripletSum(vector<int> &arr, int target) {
    sort(arr.begin(),arr.end());
    int n = arr.size();
    for(int i=0;i<n;i++){
        int j = i+1;
        int k = n-1;
        while(j<k){
            int sum = arr[i]+arr[j]+arr[k];

            if(sum == target) return true;

            else if(sum > target){
                k--;
            }
            else{
                j++;
            }
        }
    }
    return false;
}
```

Time Complexity : O(n^2)

Space Complexity : O(1)

209. Minimum Size Subarray Sum

Solv

Medium | Topics | Companies

Given an array of positive integers `nums` and a positive integer `target`, return *the* ***minimal length*** *of a subarray whose sum is greater than or equal to* `target`. If there is no such subarray, return `0` instead.

Example 1:

```
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem
constraint.
```

Example 2:

```
Input: target = 4, nums = [1,4,4]
Output: 1
```

Example 3:

```
Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0
```

```
#include<bits/stdc++.h>
using namespace std;
int minSubArrayLen(int target, vector<int>& nums) {
    int  n =nums.size();
    unordered_map<int,int>mpp;
    int left =0;
    int sum =0;
    int minlen = INT_MAX;
    for(int right = 0;right<n;right++){
        sum+=nums[right];
        while(sum>=target){
            minlen = min (minlen,right-left+1);
            sum-=nums[left];
            left++;
        }
    }

    if(minlen!= INT_MAX) return minlen;
    return 0;
}
```

Time Complexity:o(n)

Space Complexity:O(1)

3. Longest Substring Without Repeating Characters

Solved

Medium | Topics | Companies | Hint

Given a string `s`, find the length of the **longest** substring without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a
substring.
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int lengthOfLongestSubstring(string s) {
    int n = s.size();
    int maxi=0;
    int cnt=0;
    unordered_map<char,int>mp;
    int left =0,right;
    for(right =0;right<n;right++){
        mp[s[right]]++;
        while(mp[s[right]]>1){
            mp[s[left]]-= 1;
            left+=1;
        }
        maxi = max(maxi,right-left+1);
    }

    return maxi;
}
```

Time Complexity:O(n)

Space Complexity: O(n)

20. Valid Parentheses

Solved

Easy | Topics | Companies | Hint

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

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. 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`

```cpp
#include<bits/stdc++.h>
using namespace std;

 bool isValid(string s) {
     stack<char> st;
     for(char c : s){
         if(c=='(' || c=='{' || c=='['){
             st.push(c);
         }
         else{
             if(st.empty()){
                 return false;
             }
             if(c==')' && st.top() == '('){
                 st.pop();
             }else if(c==']' && st.top()=='['){
                 st.pop();

             }
             else if(c=='}' && st.top()=='{'){
                 st.pop();

             }
             else{
                 return false;
             }
         }
       }
         return st.empty();
     }
```

Time Complexity : O(n)

Space Complexity : O(1)

150. Evaluate Reverse Polish Notation

Solved

Medium | Topics | Companies

You are given an array of strings `tokens` that represents an arithmetic expression in a Reverse Polish Notation.

Evaluate the expression. Return *an integer that represents the value of the expression*.

Note that:

- The valid operators are `'+'`, `'-'`, `'*'`, and `'/'`.
- Each operand may be an integer or another expression.
- The division between two integers always **truncates toward zero**.
- There will not be any division by zero.
- The input represents a valid arithmetic expression in a reverse polish notation.
- The answer and all the intermediate calculations can be represented in a **32-bit** integer.

```cpp
#include<bits/stdc++.h>
using namespace std;
int evalRPN(vector<string>& tokens) {
    stack<int> st;
    for (auto& c : tokens) {
        if (c == "+") {
            int a = st.top(); st.pop();
            int b = st.top(); st.pop();
            st.push(a + b);
        }
        else if (c == "-") {
            int a = st.top(); st.pop();
            int b = st.top(); st.pop();
            st.push(b - a);
        }
        else if (c == "*") {
            int a = st.top(); st.pop();
            int b = st.top(); st.pop();
            st.push(a * b);
        }
        else if (c == "/") {
            int a = st.top(); st.pop();
            int b = st.top(); st.pop();
            st.push(b / a);
        }
        else {
            st.push(stoi(c));
        }
    }
    return st.top();
}
```

Time Complexity : O(n)
Space Complexity : O(n)

35. Search Insert Position

Solved

Easy | Topics | Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

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

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

```cpp
// using KeyWord
int searchInsert(vector<int>& nums, int target) {
    auto it =lower_bound(nums.begin(),nums.end(),target);
    int ind = it-nums.begin();
    return ind;

}
// By manual
int searchInsert(vector<int>& nums, int target) {
    int n=nums.size();
    int low=0,high=n-1,mid,ans=n;
    while(low<=high){
        mid=low+(high-low)/2;
        if(nums[mid]>=target){
            ans=mid;
            high=mid-1;
        }
        else{
            low=mid+1;
        }

    }
    return ans;
}
```

Time Complexity : O(log n)
Space Complexity : O(1)

155. Min Stack

Solved

Medium | Topics | Companies | Hint

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

> **Input**
> `["MinStack","push","push","push","getMin","pop","top","getMin"]`
> `[[],[-2],[0],[-3],[],[],[],[]]`
>
> **Output**
> `[null,null,null,null,-3,null,0,-2]`

```cpp
#include<bits/stdc++.h>
using namespace std;

class MinStack {
public:
    stack<int>numbers;
    stack<int>minstack;
    MinStack() {
    }
    void push(int val) {
        numbers.push(val);
        if(minstack.empty()){
            minstack.push(val);
        }
        else{
            minstack.push(val = min(val,minstack.top()));
        }
    }
    void pop() {
        numbers.pop();
        minstack.pop();
    }
    int top() {
        return numbers.top();
    }
    int getMin() {
        return minstack.top();
    }
};
```

Time Complexity : O(1) each function

Space Complexity : O(n) *uses stack to store elements

162. Find Peak Element

Solved

Medium | Topics | Companies

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.
```

```cpp
class Solution {
public:
    int findPeakElement(vector<int>& arr) {
        int n = arr.size();
        if(n <= 1 || arr[0]>arr[1]){
            return 0;
        };
        if(arr[n-1]>arr[n-2]) return n-1;
        int left =1;
        int right = n-2;
        int maxi = INT_MIN;
        while(left<=right){
            int mid = left+(right-left)/2;
            if(arr[mid]>arr[mid-1] && arr[mid]>arr[mid+1]){
               return mid;
            }
            else if(arr[mid]>arr[mid-1]){
                left = mid+1;
            }
            else{
                right = mid-1;
            }
        }
        return -1;
    }
};
```

Time Complexity : O(log n)

Space Complexity : O(1)

33. Search in Rotated Sorted Array

Solved

Medium | Topics | Companies

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
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int search(vector<int>& arr, int k) {
    int n=arr.size();
    int low=0,high=n-1;
    while(low<=high){
        int mid=(low+high)/2;
        if(arr[mid]==k) return mid;
        if(arr[low]<=arr[mid]){
            if(arr[low]<=k && k<=arr[mid]){
                high=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        else{
            if(arr[mid]<=k && k<=arr[high]){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
    }
    return -1;

}
```

Time Complexity: O(log N)
Space Complexity: O(1)

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

Solved

Medium | Topics | Companies

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]
```

```cpp
#include<bits/stdc++.h>
using namespace std;
vector<int> searchRange(vector<int>& nums, int target) {
    vector<int>res = {-1,-1};
    int low = 0 ;
    int n = nums.size();
    int high = nums.size()-1;

    while(low<=high){
        int mid = low+(high-low)/2;
        if(nums[mid] == target){
            res[0] = mid;
            high = mid-1;
        }else if(nums[mid]<target){
            low = mid+1;
        }else{
            high = mid-1;
        }
    }
    low = 0 ;
    high = n-1;
    while(low<=high){
        int mid = low+(high-low)/2;
        if(nums[mid] == target){
            res[1] = mid;
            low = mid+1;
        }else if(nums[mid]<target){
            low = mid+1;
        }else{
            high = mid-1;
        }
    }
    return res;
}
```

Time Complexity : O(log N)
Space Complexity : O(1)

153. Find Minimum in Rotated Sorted Array

Solved

Medium | Topics | Companies | Hint

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time`.

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
```

```cpp
#include<bits/stdc++.h>
using namespace std;
int findMin(vector<int>& arr) {
    int n=arr.size();
    int low=0,high=n-1;
    int ans=INT_MAX;
    while(low<=high){
        int mid=low+(high-low)/2;
        if(arr[low]<=arr[mid]){
            ans=min(ans, arr[low]);
            low=mid+1;
        }
        else{
            ans=min(ans,arr[mid]);
            high=mid-1;
        }
    }
    return ans;
}
```

Time Complexity : O(log N)

Space Complexity : O(1)

74. Search a 2D Matrix

Medium | Topics | Companies

You are given an `m x n` integer matrix `matrix` with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer `target`, return `true` *if* `target` *is in* `matrix` *or* `false` *otherwise*.

You must write a solution in `O(log(m * n))` time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

```
Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true
```

```cpp
#include<bits/stdc++.h>
using namespace std;

bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int n = matrix.size();
    int m = matrix[0].size();

    int low =0,high = n*m-1;
    while(low<=high){
        int mid = (low+high)/2;
        int row = mid/m;
        int col = mid%m;

        if(matrix[row][col] == target){
            return true;
        }
        else if(matrix[row][col]<target)
        {
            low = mid+1;
        }
        else {
            high = mid-1;
        }
    }
    return false;
}
```

Time Complexity : O(log n)

Space Complexity l=: O(1)

71. Simplify Path

Solved

Medium | Topics | Companies

You are given an *absolute* path for a Unix-style file system, which always begins with a slash `'/'`. Your task is to transform this absolute path into its **simplified canonical path**.

The *rules* of a Unix-style file system are as follows:

- A single period `'.'` represents the current directory.
- A double period `'..'` represents the previous/parent directory.
- Multiple consecutive slashes such as `'//'` and `'///'` are treated as a single slash `'/'`.
- Any sequence of periods that does **not match** the rules above should be treated as a **valid directory or file name**. For example, `'...'` and `'....'` are valid directory or file names.

The simplified canonical path should follow these *rules*:

- The path must start with a single slash `'/'`.
- Directories within the path must be separated by exactly one slash `'/'`.
- The path must not end with a slash `'/'`, unless it is the root directory.
- The path must not have any single or double periods (`'.'` and `'..'`) used to denote current or parent directories.

Return the **simplified canonical path**.

Example 1:

> **Input:** `path = "/home/"`
>
> **Output:** `"/home"`
>
> **Explanation:**
>
> The trailing slash should be removed.

```cpp
#include<bits/stdc++.h>
using namespace std;

string simplifyPath(string path) {
    vector<string> st;
    string curr = "";

    for (char c : path + "/") {
        if (c == '/') {
            if (curr == "..") {
                if (!st.empty()) st.pop_back();
            }
            else if (!curr.empty() && curr != ".") {
                st.push_back(curr);
            }
            curr = "";
        }
        else {
            curr += c;
        }
    }

    string result = "/";
    for (int i = 0; i < st.size(); ++i) {
        result += st[i];
        if (i != st.size() - 1) result += "/";
    }

    return result;
}
```

Time Complexity : O(n) ; Space Complexity: O(n)

Description | Editorial | Solutions | Submissions

76. Minimum Window Substring

Solved

Hard | Topics | Companies | Hint

Given two strings `s` and `t` of lengths `m` and `n` respectively, return *the* ***minimum window*** ***substring*** *of* `s` *such that every character in* `t` ***(including duplicates)*** *is included in the window*. If there is no such substring, return *the empty string* `""`.

The testcases will be generated such that the answer is **unique**.

Example 1:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and
'C' from string t.
```

Example 2:

```
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
```

```cpp
#include<bits/stdc++.h>
using namespace std;
string minwindow(string s, string t){
    if (t.empty())
        return "";
    unordered_map<char, int> cntT, window;
    for (char c : t){
        cntT[c]++;
    }
    int have = 0, need = cntT.size();
    int resLen = INT_MAX;
    int l = 0, start = 0;
    for (int r = 0; r < s.size(); ++r){
        char c = s[r];
        window[c]++;
        if (cntT.count(c) && window[c] == cntT[c]){
            have++;
        }
        while (have == need){
            if ((r - l + 1) < resLen){
                resLen = r - l + 1;
                start = l;
            }
            window[s[l]]--;
            if (cntT.count(s[l]) && window[s[l]] < cntT[s[l]]){
                have--;
            }
            l++;
        }
    }
    return resLen == INT_MAX ? "" : s.substr(start, resLen);
}
```

Time Complexity: O(n+m), where n is the length of `s` and m is the length of `t`
Space Complexity: O(k), where k is the number of unique characters in s and t.