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.
```

Constraints:

- $1 <= s.length <= 2 * 10^5$
- `s` consists only of printable ASCII characters.

Code:

```
bool isPalindrome(string s) {
        string b="";
        for(auto a:s){
            if(isalpha(a)||isdigit(a)){
                b+=tolower(a);
            }
        }
        int left=0,right=b.size()-1;
        while(left<right){
            if(b[left]!=b[right]){
                return false;
            }
            left++;
            right--;
        }
        return true;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(n)

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.

Code:

```
bool isSubsequence(string s, string t) {
        int sPointer=0;
        for(int i=0;i<t.size();i++){
            if(sPointer<s.size() and s[sPointer]==t[i])sPointer++;
        }
        if(sPointer==s.size())return true;
        return false;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(1)

167. Two Sum II - Input Array Is Sorted

Solved

Medium | Topics | Companies

Given a **1-indexed** array of integers `numbers` that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific `target` number. Let these two numbers be `numbers[index₁]` and `numbers[index₂]` where `1 <= index₁ < index₂ <= numbers.length`.

Return *the indices of the two numbers,* `index₁` *and* `index₂`*, **added by one** as an integer array* `[index₁, index₂]` *of length 2.*

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

```
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].
```

Example 3:

```
Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].
```

Constraints:

- `2 <= numbers.length <= 3 * 10⁴`
- `-1000 <= numbers[i] <= 1000`

Code:

```
vector<int> twoSum(vector<int>& numbers, int target) {
        int left=0,right=numbers.size()-1;
        while(left<right){
            int sum=numbers[left]+numbers[right];
            if(sum==target){
                return {left+1,right+1};
            }
            else if(sum<target){
                left++;
            }
            else{
                right--;
            }
        }
        return {-1,-1};
}
```

TimeComplexity:O(n)
SpaceComplexity:O(1)

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

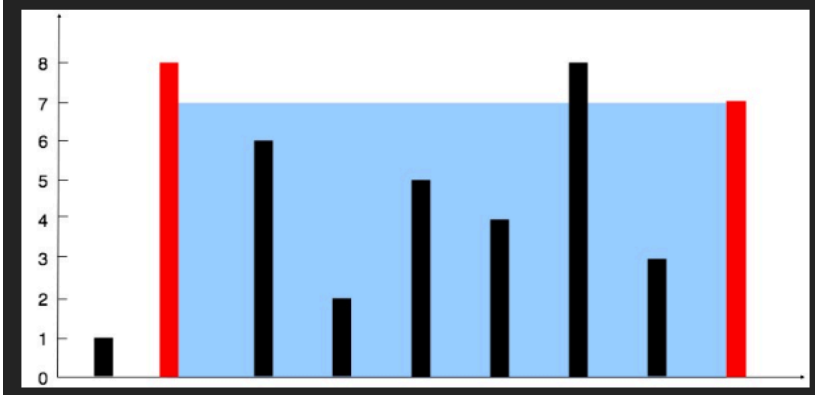
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:




```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
```

Code:

```
int maxArea(vector<int>& height) {
        int left=0,right=height.size()-1;
        int ans=0;
        while(left<right){
            ans=max(ans,(right-left)*min(height[right],height[left]));
            if(height[right]>=height[left]){
                left++;
            }
            else{
                right--;
            }
        }
        return ans;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(1)

15. 3Sum

Solve

Medium | Topics | Companies | Hint

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

Constraints:

- `3 <= nums.length <= 3000`
- `-10^5 <= nums[i] <= 10^5`

Code:

```
vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> v1;
        sort(nums.begin(),nums.end());
        for(int i=0;i<nums.size();i++){
            if(i>0 and nums[i]==nums[i-1])continue;
            int j=i+1;
            int k=nums.size()-1;
            while(j<k){
                int sum=nums[i]+nums[j]+nums[k];
                if(sum<0){
                    j++;
                }
                else if(sum>0){
                    k--;
                }
                else{
                    v1.push_back({nums[i],nums[j],nums[k]});
                    j++;
                    k--;
                    while(j<k and nums[j]==nums[j-1])j++;
                    while(j<k and nums[k]==nums[k+1])k--;
                }
            }
        }
        return v1;
}
```

TimeComplexity:O(n2)**

SpaceComplexity:O(n)

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.
```

Constraints:

- $0 <= s.length <= 5 * 10^4$
- `s` consists of English letters, digits, symbols and spaces.

Code:

```
int lengthOfLongestSubstring(string s) {
        int maxi=0;
        unordered_map<char,int> hash;
        int left=0;
        for(int i=0;i<s.size();i++){
            hash[s[i]]++;
            while(hash[s[i]]>1){
                hash[s[left]]--;
                left++;
            }
            maxi=max(maxi,i-left+1);
        }
        return maxi;
}
```

TimeComplexity:O(n)
SpaceComplexity: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
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`
- `nums` contains **distinct** values sorted in **ascending** order.
- `-10^4 <= target <= 10^4`

Code:

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

TimeComplexity:O(logn)
SpaceComplexity:O(1)

74. Search a 2D Matrix

Solved

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
```

Code:

```
int BinSearch(vector<int> arr,int low,int high,int target){
        while(low<=high){
            int mid=(low+high)/2;
            if(target==arr[mid]){
                return mid;
            }
            else if(arr[mid]<target){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
        return -1;
}
bool searchMatrix(vector<vector<int>>& matrix, int target) {
        for(int i=0;i<matrix.size();i++){
            if(BinSearch(matrix[i],0,matrix[i].size()-1,target)!=-1)return true;
        }
        return false;
}
```

TimeComplexity:O(nlogm)
SpaceComplexity:O(1)

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 inde
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.
```

Constraints:

- `1 <= nums.length <= 1000`
- `$-2^{31}$ <= nums[i] <= $2^{31} - 1$`

Code:

```
int findPeakElement(vector<int>& nums) {
        int n=nums.size();
        if(nums.size()==1)return 0;
        if(nums[0]>nums[1])return 0;
        if(nums[n-1]>nums[n-2])return n-1;
        int low=1,high=n-2;
        while(low<=high){
            int mid=(low+high)/2;
            //cout<<low<<" "<<mid<<" "<<high<<endl;
            if(nums[mid-1]<nums[mid] and nums[mid]>nums[mid+1]){
                return mid;
            }
            else if(nums[mid-1]<nums[mid]){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
        return -1;
}
```

TimeComplexity:O(log n)
SpaceComplexity: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
```

Example 3:

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

Code:

```
int search(vector<int>& nums, int target) {
        int low=0,high=nums.size()-1;
        while(low<=high){
            int mid=(low+high)/2;
            //cout<<low<<" "<<nums[mid]<<" "<<high<<endl;
            if(nums[mid]==target)return mid;
            if(nums[mid]<nums[high]){
                if(nums[mid]<=target and target<=nums[high]){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
            else{
                if(nums[low]<=target and target<=nums[mid]){
                    high=mid-1;
                }
                else{
                    low=mid+1;
                }
            }
        }
        return -1;
}
```

TimeComplexity:O(log n)
SpaceComplexity: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]
```

Constraints:

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`
- `nums` is a non-decreasing array.
- `-10^9 <= target <= 10^9`

Code:

```
int upperBound(vector<int> arr,int target){
        int low=0,high=arr.size()-1,ans=arr.size();
        while(low<=high){
            int mid=(low+high)/2;
            if(arr[mid]<=target){
                low=mid+1;
                ans=mid;
            }
            else{
                high=mid-1;
            }
        }
        return ans;
    }
int lowerBound(vector<int> arr,int target){
        int low=0,high=arr.size()-1,ans=arr.size();
        while(low<=high){
            int mid=(low+high)/2;
            if(arr[mid]<target){
                low=mid+1;
            }
            else{
                high=mid-1;
                ans=mid;
            }
        }
        return ans;
}
vector<int> searchRange(vector<int>& nums, int target) {
        if(nums.size()==0)return {-1,-1};
        int lb=lowerBound(nums,target);
        if(lb==nums.size() or nums[lb]!=target)return {-1,-1};
        int ub=upperBound(nums,target);
        return {lb,ub};
}
```

TimeComplexity:O(log n)
SpaceComplexity: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.
```

Example 3:

```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

Code:

```
int findMin(vector<int>& nums) {
        int low=0,high=nums.size()-1,ans=INT_MAX;
        while(low<=high){
            int mid=(low+high)/2;
            //cout<<low<<" "<<high<<" "<<nums[mid]<<endl;
            if(nums[mid]<nums[high]){
                ans=min(ans,nums[mid]);
                high=mid-1;
            }
            else{
                low=mid+1;
                ans=min(ans,nums[mid]);
            }
        }
        return ans;
}
```

TimeComplexity:O(log n)
SpaceComplexity: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`

Code:

```
bool isValid(string s) {
        stack<char> st;
        for(int i=0;i<s.size();i++){
            if(s[i]=='('){
                st.push(')');
            }
            else if(s[i]=='['){
                st.push(']');
            }
            else if(s[i]=='{'){
                st.push('}');
            }
            else{
                if(st.empty())return false;
                if(st.top()!=s[i])return false;
                st.pop();
            }
        }
        if(st.empty())return true;
        return false;
}
```

TimeComplexity:

SpaceComplexity:

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"`

Code:

```
string simplifyPath(string path) {
        stack<string> st;
        for(int i=0;i<path.size();i++){
            if(path[i]=='/'){
                continue;
            }
            else if(path[i]=='.'){
                int count=1;
                string dots=".";
                while(i+1<path.size() and path[i+1]!='/'){
                    dots+=path[i+1];
                    count++;
                    i++;
                }
                if(dots==".."){
                    if(!st.empty()){st.pop();st.pop();}
                }
                else if(count>1){
                    st.push("/");
                    st.push(dots);
                }
            }
```

```
            else{
                string dir="";
                dir+=path[i];
                while(i+1<path.size() and path[i+1]!='/'){
                    dir+=path[i+1];
                    i++;
                }
                //cout<<dir<<endl;
                st.push("/");
                st.push(dir);
            }
        }
        string ans="";
        while(!st.empty()){
            ans=st.top()+ans;
            st.pop();
        }
        if(ans.size()==0)return "/";
        return ans;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(n)

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]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
```

Code:

```
class MinStack {
public:
    stack<int> original,monStack;
    MinStack() {}
    void push(int val) {
        original.push(val);
        if(monStack.empty() || monStack.top()>=val){
            monStack.push(val);
        }
    }
    void pop() {
        if(original.empty())return;
        if(!monStack.empty() and original.top()==monStack.top()){
            monStack.pop();
        }
        original.pop();
    }
    int top() {
        if(original.empty())return NULL;
        return original.top();
    }
    int getMin() {
        if(monStack.empty())return NULL;
        return monStack.top();
    }
};
```

TimeComplexity:O(1)
SpaceComplexity:O(n)

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.

Example 1:

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Code:

```
int evalRPN(vector<string>& tokens) {
        stack<int> st;
        for(auto aa:tokens){
            if(aa!="+" and aa!="*" and aa!="-" and aa!="/"){
                st.push(stoi(aa));
            }
            else{
                int a=st.top();
                st.pop();
                int b=st.top();
                st.pop();
                if(aa=="+"){
                    st.push(a+b);
                }
                else if(aa=="-"){
                    st.push(b-a);
                }
                else if(aa=="*"){
                    st.push(a*b);
                }
                else{
                    st.push(b/a);
                }
            }
        }
        return st.top();
}
```

TimeComplexity:O(n)
SpaceComplexity:O(n)

209. Minimum Size Subarray Sum

Solved

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
```

Constraints:

- $1 <= target <= 10^9$
- $1 <= nums.length <= 10^5$
- $1 <= nums[i] <= 10^4$

Code:

```
int minSubArrayLen(int target, vector<int>& nums) {
        int sum=0,left=0;
        int ans=10e5;
        for(int i=0;i<nums.size();i++){
            sum+=nums[i];
            while(sum>=target){
                ans=min(ans,i-left+1);
                sum-=nums[left];
                left++;
            }
        }
        if(ans==10e5)return 0;
        return ans;
}
```

TimeComplexity:O(n)
SpaceComplexity:O(1)