





















DSA Practice

Date: Nov 21

Problems:

Two Pointers	Sliding Window
<input type="radio"/> Valid Palindrome  Easy	<input type="radio"/> Minimum Size Subarray S...  Medium
<input type="radio"/> Is Subsequence  Easy	<input type="radio"/> Longest Substring Witho...  Medium
<input type="radio"/> Two Sum II - Input Array I...  Medium	<input type="radio"/> Substring with Concatenati...  Hard
<input type="radio"/> Container With Most Water  Medium	<input type="radio"/> Minimum Window Substring  Hard
<input type="radio"/> 3Sum  Medium	

Stack	Binary Search
<input type="radio"/> Valid Parentheses  Easy	<input type="radio"/> Search Insert Position  Easy
<input type="radio"/> Simplify Path  Medium	<input type="radio"/> Search a 2D Matrix  Medium
<input type="radio"/> Min Stack  Medium	<input type="radio"/> Find Peak Element  Medium
<input type="radio"/> Evaluate Reverse Polish ...  Medium	<input type="radio"/> Search in Rotated Sorted ...  Medium
<input type="radio"/> Basic Calculator  Hard	<input type="radio"/> Find First and Last Positi...  Medium
	<input type="radio"/> Find Minimum in Rotated ...  Medium

Problem 1:

224. Basic Calculator

Hard Topics Companies

Given a string `s` representing a valid expression, implement a basic calculator to evaluate it, and return the result of the evaluation.

Note: You are **not** allowed to use any built-in function which evaluates strings as mathematical expressions, such as `eval()`.

Example 1:
Input: `s = "1 + 1"`
Output: `2`

Example 2:
Input: `s = "2-1 + 2 "`
Output: `3`

Example 3:
Input: `s = "(1+(4+5+2)-3)+(6+8)"`
Output: `23`

Constraints:

- 85 Online

```
4     index=0;
5     return cal(s);
6 }
7 public int cal(String s){
8     int num = 0;
9     int res=0;
10    int sign=1;
11    while(index<s.length()){
12        char ch = s.charAt(index++);
13        if(ch>='0' && ch<='9'){
14            num = num*10 + ch-'0';
15        }else if(ch=='('){
16            num = cal(s);
17        }else if(ch==')'){
18            return res+sign*num;
19        }else if(ch=='+' || ch=='-'){
20            res+=num*sign;
21            num=0;
22            sign = ch=='+'?1:-1;
23        }
24    }
25    return res+sign*num;
26 }
27 }
```

Problem 2:

76. Minimum Window Substring

Hard Topics Companies Hint

Solved

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.

Example 3:
Input: `s = "a"`, `t = "aa"`
Output: `""`
Explanation: Both 'a's from `t` must be included in the window.

Constraints:

- 18.3K Online

```
1 class Solution {
2     public String minWindow(String s, String t) {
3         int n = s.length();
4         int m = t.length();
5         int[] nums = new int[128];
6         int count=0;
7         for(char ch : t.toCharArray()){
8             if(nums[ch]==0){
9                 count++;
10            }
11            nums[ch]++;
12        }
13        int left=0;
14        int right=0;
15        int minlen = Integer.MAX_VALUE;
16        int startindex=-1, endindex=-1;
17        while(right<n){
18            char ch = s.charAt(right);
19            nums[ch]--;
20            if(nums[ch]==0){
21                count--;
22            }
23            while(count==0){
24                int len = right-left+1;
25                if(len<minlen){
26                    minlen=len;
27                    startindex=left;
28                }
29            }
30            right++;
31        }
32        return startindex==-1 ? "" : s.substring(startindex, endindex+1);
33    }
34 }
```

Problem 3:

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.

```
1 class Solution {
2     public int lengthOfLongestSubstring(String s) {
3         int n = s.length();
4         Map<Character, Integer> map = new HashMap<>();
5         int left = 0;
6         int max = 0;
7         for (int right = 0; right < n; right++) {
8             map.put(s.charAt(right), map.getOrDefault(s.charAt(right), 0) + 1);
9             if (map.size() >= (right - left + 1) + 1) {
10                 max = Math.max(max, (right - left) + 1);
11             }
12             while (map.size() > (right - left + 1)) {
13                 map.put(s.charAt(left), map.get(s.charAt(left)) - 1);
14                 if (map.get(s.charAt(left)) == 0) {
15                     map.remove(s.charAt(left));
16                 }
17                 left++;
18             }
19         }
20         return max;
21     }
22 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Problem 4:

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 class Solution {
2     public int minSubarrayLen(int target, int[] nums) {
3         int n = nums.length;
4         int left = 0;
5         int sum = 0;
6         int min = Integer.MAX_VALUE;
7         for (int right = 0; right < n; right++) {
8             sum += nums[right];
9             while (sum >= target) {
10                 min = Math.min(min, (right - left + 1));
11                 sum -= nums[left];
12                 left++;
13             }
14         }
15         return min == Integer.MAX_VALUE ? 0 : min;
16     }
17 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Problem 5:

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`

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0;
        int right = nums.length - 1;
        int ans = Integer.MAX_VALUE;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (nums[left] <= nums[mid]) {
                ans = Math.min(nums[left], ans);
                left = mid + 1;
            } else {
                ans = Math.min(nums[mid], ans);
                right = mid;
            }
        }
        return ans;
    }
}
```

Problem 6:

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]`

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        int n = nums.length;
        int low = 0;
        int high = n - 1;
        int index = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] == target) {
                index = mid;
                break;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        if (index == -1) {
            return new int[]{-1, -1};
        }
        int pos1 = index;
        while (pos1 > 0 && nums[pos1] == nums[pos1 - 1]) {
            pos1--;
        }
        int pos2 = index;
        while (pos2 < n - 1 && nums[pos2] == nums[pos2 + 1]) {
            pos2++;
        }
        return new int[]{pos1, pos2};
    }
}
```

Problem 7:

33. Search in Rotated Sorted Array

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 ($1 \leq k < \text{nums.length}$) 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 = [4,5,6,7,0,1,2]`, `target = 0`
Output: `4`

```
1 class Solution {
2     public int search(int[] nums, int target) {
3         int low = 0;
4         int high = nums.length-1;
5         while(low<high){
6             int mid = (low+high)/2;
7             if(nums[mid]==target){
8                 return mid;
9             }else if(nums[low]<nums[mid]){
10                if(nums[low]<target && target <= nums[mid]){
11                    high = mid - 1;
12                }else{
13                    low = mid + 1;
14                }
15            }else{
16                if(nums[mid]<target && target <= nums[high]){
17                    low = mid + 1;
18                }else{
19                    high = mid - 1;
20                }
21            }
22        }
23        return -1;
24    }
25 }
```

Problem 8:

162. Find Peak Element

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.

```
1 class Solution {
2     public int findPeakElement(int[] nums) {
3         int n = nums.length;
4         if(n==1){
5             return 0;
6         }
7         if(nums[0]>nums[1]){
8             return 0;
9         }
10        if(nums[n-1]>nums[n-2]){
11            return n-1;
12        }
13        int low=1;
14        int high=n-1;
15        while(low<high){
16            int mid = (low+high)/2;
17            if(nums[mid]>nums[mid-1] && nums[mid]>nums[mid+1]){
18                return mid;
19            }else if(nums[mid]>nums[mid-1]){
20                low=mid+1;
21            }else{
22                high=mid-1;
23            }
24        }
25        return 0;
26    }
27 }
```

Problem 9:

74. Search a 2D Matrix

Medium

You are given an $m \times n$ 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

```
1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         int n = matrix.length;
4         int m = matrix[0].length;
5         int low = 0;
6         int high = n-1;
7         int mid=-1;
8         while(low<high){
9             mid = (low+high)/2;
10            if(matrix[mid][0]==target){
11                return true;
12            }
13            else if(matrix[mid][0]<target){
14                low=mid+1;
15            }else{
16                high=mid-1;
17            }
18        }
19        int index=-1;
20        if(matrix[mid][0]<target){
21            index=mid;
22        } else{
23            index=mid==0?0:mid-1;
24        }
25        System.out.println(index);
26        int start=0;
27        int end=m-1;
28        while(start<end){
```

Problem 10:

35. Search Insert Position

Easy

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 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int n = nums.length;
4         int low = 0;
5         int high=n-1;
6         int mid = -1;
7         while(low<high){
8             mid = (low+high)/2;
9             if(nums[mid]==target){
10                return mid;
11            }else if(nums[mid]<target){
12                low=mid+1;
13            }else{
14                high=mid-1;
15            }
16        }
17        if(nums[mid]>target){
18            return mid==0?0:mid;
19        }
20        return mid+1;
21    }
22 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

