

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

The screenshot shows a LeetCode submission page for a Java solution. The top navigation bar includes 'Array', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The submission status is 'Accepted' with 66/66 testcases passed, submitted by 'duefoxx' at Feb 11, 2026 23:40. The runtime chart shows 0 ms beats 100.00%, and the memory chart shows 45.11 MB beats 10.03%. The code editor contains the following Java code for a binary search solution:

```
1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int left = 0;
4         int right = nums.length - 1;
5
6         while (left <= right) {
7             int mid = left + (right - left) / 2;
8
9             if (nums[mid] == target)
10                 return mid;
11             else if (nums[mid] < target)
12                 left = mid + 1;
13             else
14                 right = mid - 1;
15         }
16
17         // left is the correct insertion position
18         return left;
19     }
20 }
```

The code editor also shows a 'Saved' message and a 'Ln 1, Col 1' indicator. Below the code editor is a 'Testcase' section with three checked cases: Case 1, Case 2, and Case 3. There is also an 'Input' field.

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times.

Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

Input: candidates = [2,3,6,7], target = 7

Output: [[2,2,3],[7]]

Explanation:

2 and 3 are candidates, and $2 + 2 + 3 = 7$. Note that 2 can be used multiple times.

7 is a candidate, and $7 = 7$.

These are the only two combinations.

Example 2:

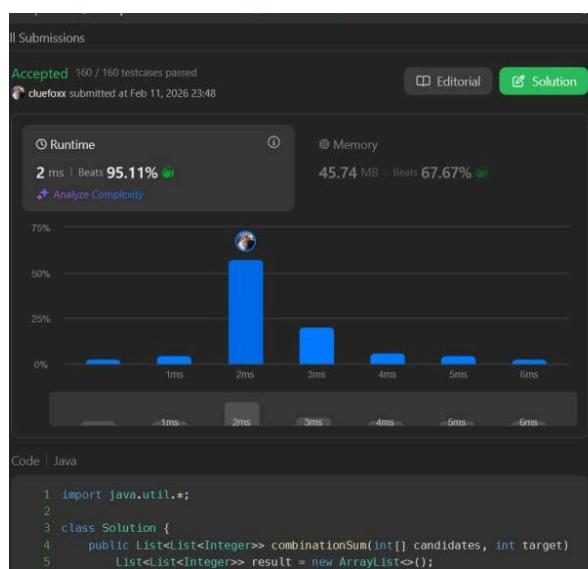
Input: candidates = [2,3,5], target = 8

Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:

Input: candidates = [2], target = 1

Output: []



The screenshot shows a LeetCode submission page for a Java solution. The top bar indicates "Accepted" with 160 / 160 testcases passed, submitted by duefoxx at Feb 11, 2026 23:48. Below this, there are tabs for "Editorial" and "Solution". The "Solution" tab is active, showing the following code:

```
1 import java.util.*;
2
3 class Solution {
4     public List<List<Integer>> combinationSum(int[] candidates, int target) {
5         List<List<Integer>> result = new ArrayList<>();
6         backtrack(candidates, target, 0, new ArrayList<>(), result);
7     }
8
9     private void backtrack(int[] candidates, int target, int start,
10                           List<Integer> current, List<List<Integer>> result) {
11
12         // Base case
13         if (target == 0) {
14             result.add(new ArrayList<>(current));
15             return;
16         }
17
18         if (target < 0) {
19             return;
20         }
21     }
22 }
```

The "Runtime" section shows a histogram with a single bar at 2ms, labeled "Beats 95.11%". The "Memory" section shows 45.74 MB, labeled "Beats 67.67%". Below the code editor, the "Testcase" and "Test Result" sections show "Accepted" status with "Runtime: 0 ms" and "Case 1", "Case 2", and "Case 3" checked.

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.

Each number in candidates may only be used once in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

Input: candidates = [10,1,2,7,6,1,5], target = 8

Output:

```
[  
[1,1,6],  
[1,2,5],  
[1,7],  
[2,6]  
]
```

Example 2:

Input: candidates = [2,5,2,1,2], target = 5

Output:

```
[  
[1,2,2],  
[5]  
]
```

40. Combination Sum II

Medium Topics Companies

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`. Each number in `candidates` may only be used once in the combination. Note: The solution set must not contain duplicate combinations.

Example 1:
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]

Example 2:
Input: candidates = [2,5,2,1,2], target = 5
Output:
[
[1,2,2],
[5]
]

```
Java Auto  
1 import java.util.*;  
2  
3 class Solution {  
4     public List<List<Integer>> combinationSum2(int[] candidates, int target) {  
5         List<List<Integer>> result = new ArrayList<>();  
6  
7         Arrays.sort(candidates); // Step 1: Sort  
8  
9         backtrack(candidates, target, 0, new ArrayList<>(), result);  
10    }  
11  
12    private void backtrack(int[] candidates, int target, int start,  
13                           List<Integer> current, List<List<Integer>> result) {  
14        if (target == 0) {  
15            result.add(new ArrayList<>(current));  
16            return;  
17        }  
18        for (int i = start; i < candidates.length; i++) {  
19            if (target - candidates[i] <= 0) {  
20                break;  
21            }  
22            current.add(candidates[i]);  
23            backtrack(candidates, target - candidates[i], i + 1, current, result);  
24            current.remove(current.size() - 1);  
25        }  
26    }  
27}
```

Saved Ln 43; Col 1

Testcase Test Result
Accepted Runtime: 1 ms
Case 1 Case 2

12.1K 221 169 Online

You are given a 0-indexed array of integers `nums` of length `n`. You are initially positioned at index 0.

Each element `nums[i]` represents the maximum length of a forward jump from index `i`.

In other words, if you are at index `i`, you can jump to any index `(i + j)` where:

- $0 \leq j \leq \text{nums}[i]$ and
- $i + j < n$

Return the minimum number of jumps to reach index `n - 1`. The test cases are generated such that you can reach index `n - 1`.

Example 1:

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

Output: 2

Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

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

Output: 2

Link: <https://leetcode.com/problems/jump-game-ii/>?envType=problem-list-v2&envId=array

The screenshot shows the LeetCode problem page for "Jump Game II". The problem description and examples are visible on the left, and the Java code solution is on the right. The code is a one-pass greedy algorithm that iterates through the array, keeping track of the farthest index it can reach and the number of jumps taken. It increments the jump count whenever it reaches the current end of the jump range. The code is well-formatted with line numbers and comments. On the right, the code editor shows the accepted status with a runtime of 0 ms for two test cases.

```
Java ▾ Auto
1 class Solution {
2     public int jump(int[] nums) {
3         int jumps = 0;
4         int currentEnd = 0;
5         int farthest = 0;
6
7         for (int i = 0; i < nums.length - 1; i++) {
8
9             farthest = Math.max(farthest, i + nums[i]);
10
11            if (i == currentEnd) {
12                jumps++;
13                currentEnd = farthest;
14            }
15        }
16
17     }
18 }
19 }
```

Saved Ln 20, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Given an array of strings `strs`, group the anagrams together. You can return the answer in any order.

Example 1:

Input: `strs = ["eat", "tea", "tan", "ate", "nat", "bat"]`

Output: `[["bat"], ["nat", "tan"], ["ate", "eat", "tea"]]`

Explanation:

- There is no string in `strs` that can be rearranged to form "bat".
- The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
- The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

Example 2:

Input: `strs = [""]`

Output: `[[""]]`

Example 3:

Input: `strs = ["a"]`

Output: `[["a"]]`

The screenshot shows a Java code editor interface with the following details:

- Title:** 49. Group Anagrams
- Difficulty:** Medium
- Topics:** Companies
- Description:** Given an array of strings `strs`, group the anagrams together. You can return the answer in any order.
- Code:**

```
1 class Solution {
2     public List<List<String>> groupAnagrams(String[] strs) {
3         Map<String, List<String>> map = new HashMap<>();
4
5         // Step 2: Traverse each string
6         for (String s : strs) {
7
8             // Convert to char array and sort
9             char[] chars = s.toCharArray();
10            Arrays.sort(chars);
11
12            // Convert back to string (key)
13            String key = new String(chars);
14
15            // Step 3: Insert into map
16            if (!map.containsKey(key)) {
17                map.put(key, new ArrayList<>());
18            }
19
20            map.get(key).add(s);
21        }
22    }
23 }
```
- Test Result:** Accepted, Runtime: 1 ms
- Test Cases:** Case 1, Case 2, Case 3

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]`.

The screenshot shows a programming environment with the following details:

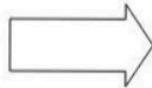
- Problem Title:** 66. Plus One
- Description:** You are given a large integer represented as an integer array `digits`, where each `digits[i]` is the i^{th} digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.
- 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]`.
- Code:** Java code for `plusOne` method.
- Test Result:** Accepted, Runtime: 0 ms, Cases 1, 2, and 3 passed.

Given an $m \times n$ integer matrix matrix , if an element is 0, set its entire row and column to 0's.

You must do it in place.

Example 1:

1	1	1
1	0	1
1	1	1



1	0
0	0
1	0

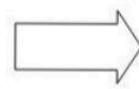
Take Home

1 **Input:** $\text{matrix} = [[1,1,1],[1,0,1],[1,1,1]]$

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

Example 2:

0	1	2	0
3	4	5	2
1	3	1	5



0	0
0	4
0	3

Input: $\text{matrix} = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]$

Output: $[[0,0,0,0],[0,4,5,0],[0,3,1,0]]$

Link: <https://leetcode.com/problems/set-matrix-zeroes/description/?envType=problem-list-v2&envId=array>

Array

Description Editorial Solutions Submissions

Medium Topics Companies Hint

Given an $m \times n$ integer matrix matrix , if an element is 0, set its entire row and column to 0's.

You must do it in place.

Example 1:

1	1	1
1	0	1
1	1	1

Input: $\text{matrix} = [[1,1,1],[1,0,1],[1,1,1]]$

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

Example 2:

0	1	2	0
2	4	5	3

Input: $\text{matrix} = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]$

Output: $[[0,0,0,0],[0,4,5,0],[0,3,1,0]]$

Code

```

Java Auto
39         matrix[i][j] = 0;
40     }
41 }
42 }
43
44 // Zero out first row if needed
45 if (firstRowZero) {
46     for (int j = 0; j < cols; j++) {
47         matrix[0][j] = 0;
48     }
49 }
50
51 // Zero out first column if needed
52 if (firstColZero) {
53     for (int i = 0; i < rows; i++) {
54         matrix[i][0] = 0;
55     }
56 }
57 }
58 }
```

Saved

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

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.

The screenshot shows a LeetCode problem page for "74. Search a 2D Matrix". The problem statement and example matrix are identical to the one above. The Java code in the editor is a binary search implementation for a 2D matrix:

```
11 while (left <= right) {  
12     int mid = left + (right - left) / 2;  
13  
14     int row = mid / n;  
15     int col = mid % n;  
16  
17     if (matrix[row][col] == target) {  
18         return true;  
19     }  
20     else if (matrix[row][col] < target) {  
21         left = mid + 1;  
22     }  
23     else {  
24         right = mid - 1;  
25     }  
26 }  
27  
return false;  
28}  
29}  
30}  
31}
```

The status bar indicates the code is saved and the current line is Ln 31, Col 1. Below the code editor is a test result panel showing "Accepted" status with a runtime of 0 ms, and two test cases labeled Case 1 and Case 2.

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

Example 2:

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 = 13

Output: false

Given an array `nums` with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

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

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

Example 2:

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

Output: `[0,1,2]`

The screenshot shows a programming challenge titled "75. Sort Colors". The problem statement asks to sort an array of integers (0, 1, 2) representing colors in-place. It provides two examples and constraints. The Java code implements the Dutch National Flag algorithm. The code editor shows the Java code, and the test results show it passed all cases with 0 ms runtime.

Description: Given an array `nums` with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

Medium, **Topics**, **Companies**, **Hint**

Example 1:
Input: `nums = [2,0,2,1,1,0]`
Output: `[0,0,1,1,2,2]`

Example 2:
Input: `nums = [2,0,1]`
Output: `[0,1,2]`

Constraints:

- `n == nums.length`
- `1 <= n <= 300`

Code

```
Java
6     while (mid <= high) {
7         if (nums[mid] == 0) {
8             int temp = nums[low];
9             nums[low] = nums[mid];
10            nums[mid] = temp;
11            low++;
12            mid++;
13        }
14        else if (nums[mid] == 1) {
15            mid++;
16        }
17        else { // nums[mid] == 2
18            int temp = nums[mid];
19            nums[mid] = nums[high];
20            nums[high] = temp;
21            high--;
22        }
23    }
24}
25
26
```

Saved | Ln 26, C

Testcase | **Test Result**

Accepted Runtime: 0 ms

Case 1 Case 2

Input

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]]`

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows "Array" and other navigation icons.
- Header:** "78. Subsets" and "Java".
- Description:** "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**.
- Code Area:** Contains the following Java code:

```
6     }
7
8     private void backtrack(int[] nums, int index, List<Integer> current,
9                           List<List<Integer>> result) {
10        // Add the current subset
11        result.add(new ArrayList<>(current));
12
13        // Generate all subsets starting from index
14        for (int i = index; i < nums.length; i++) {
15            // Include nums[i]
16            current.add(nums[i]);
17
18            // Recurse with next index
19            backtrack(nums, i + 1, current, result);
20
21            // Backtrack: remove last element
22            current.remove(current.size() - 1);
23        }
24    }
25 }
```
- Test Result:** Shows "Accepted" status with "Runtime: 0 ms" and two test cases: "Case 1" and "Case 2" both marked as passed.
- Bottom Status:** "Saved" and "Ln 26, C".
- Footer:** "19K" upvotes, "234" downvotes, a star icon, and "215 Online".

Given an $m \times n$ grid of characters board and a string word, return true if word exists in the grid.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word =

The screenshot shows a code editor interface with a Java file open. The code implements a depth-first search (DFS) algorithm to find a word in a grid. The grid is shown below the code editor.

Java Code:

```
Java Auto
22     if (i < 0 || j < 0 || i >= board.length || j >= board[0].length
23         || board[i][j] != word.charAt(index)) {
24             return false;
25         }
26
27         char temp = board[i][j];
28         board[i][j] = '#'; // mark as visited
29
30         // Search all 4 directions
31         boolean found =
32             dfs(board, i + 1, j, word, index + 1) ||
33             dfs(board, i - 1, j, word, index + 1) ||
34             dfs(board, i, j + 1, word, index + 1) ||
35             dfs(board, i, j - 1, word, index + 1);
36
37         board[i][j] = temp; // backtrack
38
39         return found;
40     }
41 }
42 }
```

Test Result:

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Grid:

A	B	C	E
S	F	C	S
A	D	E	E

Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCED"

Output: true

Given an array `nums` of n integers, return an array of all the unique quadruplets $[nums[a], nums[b], nums[c], nums[d]]$ such that:

- $0 \leq a, b, c, d < n$
- $a, b, c,$ and d are distinct.
- $nums[a] + nums[b] + nums[c] + nums[d] == target$

You may return the answer in any order.

Example 1:

Input: `nums = [1,0,-1,0,-2,2]`, `target = 0`

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

Example 2:

Input: `nums = [2,2,2,2,2]`, `target = 8`

Output: `[[2,2,2,2]]`

The screenshot shows a programming challenge interface. On the left, the problem details for "18. 4Sum" are displayed, including the problem statement, constraints, and two examples with their respective inputs and outputs. On the right, a code editor window is open, showing a Java implementation of the solution. The code uses a two-pointer approach to find quadruplets. It includes comments explaining the logic: moving pointers based on the sum of the current quadruplet relative to the target. The code editor also shows test results at the bottom, indicating the solution was accepted with a runtime of 1 ms for two test cases.

```
Java
22
23     left++;
24     right--;
25
26     while (left < right && nums[left] == nums[left - 1]) left++;
27     while (left < right && nums[right] == nums[right + 1]) right--;
28
29     else if (sum < target) {
30         left++;
31     } else {
32         right--;
33     }
34
35 }
36
37 }
38
39     return result;
40 }
41 }
```

Testcase | Test Result
Accepted Runtime: 1 ms
Case 1 Case 2

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

Prior to being passed to your function, `nums` is possibly left rotated at an unknown index k ($1 \leq k < \text{nums.length}$) such that the resulting array is $[\text{nums}[k], \text{nums}[k+1], \dots, \text{nums}[\text{n}-1], \text{nums}[0], \text{nums}[1], \dots, \text{nums}[\text{k}-1]]$ (0-indexed). For example, $[0,1,2,4,5,6,7]$ might be left rotated by 3 indices 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

The screenshot shows a LeetCode problem page for "33. Search in Rotated Sorted Array". The problem statement and examples are identical to the ones above. The code editor contains a Java implementation of a binary search algorithm for a rotated sorted array. The code is as follows:

```
Java
18     if (target >= nums[left] && target < nums[mid]) {
19         right = mid - 1;
20     } else {
21         left = mid + 1;
22     }
23 }
24 // Else right part is sorted
25 else {
26     // Check whether target is in the right sorted half
27     if (target > nums[mid] && target <= nums[right]) {
28         left = mid + 1;
29     } else {
30         right = mid - 1;
31     }
32 }
33
34 return -1; // not found
35
36 }
37 }
```

The code editor shows the code saved with 38 columns. Below the editor, the "Test Result" section shows "Accepted" status with a runtime of 0 ms. It includes three test cases: Case 1, Case 2, and Case 3, all of which are checked. At the bottom, the statistics show 29.7K likes, 549 dislikes, and 429 online users.

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]

The screenshot shows a programming challenge interface. The title is "34. Find First and Last Position of Element in Sorted Array". The problem description states: "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 shows input `nums = [5,7,7,8,8,10]` and target 8, with output [3,4]. Example 2 shows input `nums = [5,7,7,8,8,10]` and target 6, with output [-1,-1]. Example 3 shows input `nums = []` and target 0, with output [-1,-1]. The code editor contains a Java solution:

```
Java Auto
18     return result;
19 }
20
21     result[0] = left;
22
23     // Find last position
24     right = nums.length - 1;
25     while (left <= right) {
26         int mid = left + (right - left) / 2;
27         if (nums[mid] <= target) {
28             left = mid + 1;
29         } else {
30             right = mid - 1;
31         }
32     }
33
34     result[1] = right;
35
36 }
37
38 }
```

The code is saved, and the test results show it is accepted with a runtime of 0 ms across three cases.