

1. 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 coding environment with the following details:

- Description:** Accepted (66 / 66 testcases passed)
- Editor:** Anurag Betal submitted at Feb 16, 2026 20:47
- Runtime:** 0 ms | Beats 100.00%
- Memory:** 44.58 MB | Beats 92.63%
- Code:** Java (Auto)
- Code Content:**

```
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
18         return left;
19     }
20 }
```
- Test Result:** In 22, Col 2

2. 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 code editor interface with a Java file open. The code implements a backtracking algorithm to find all unique combinations of numbers in the candidates array that sum up to the target. The editor displays runtime and memory statistics, and a bar chart showing execution time distribution.

```
import java.util.*;
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<>();
        backtrack(candidates, target, 0, new ArrayList<>(), result);
        return result;
    }
    private void backtrack(int[] candidates, int target, int start,
                          List<Integer> current, List<List<Integer>> result) {
        if (target == 0) {
            result.add(new ArrayList<>(current));
            return;
        }
        for (int i = start; i < candidates.length; i++) {
            if (candidates[i] > target) {
                continue;
            }
            current.add(candidates[i]);
            backtrack(candidates, target - candidates[i], i, current, result);
            current.remove(current.size() - 1);
        }
    }
}
```

3. 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]

]

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

- Runtime:** 5 ms | Beats 99.22% (Memory: 45.30 MB | Beats 52.80%)
- Code:** Java
- Testcase:** Accepted | Runtime: 2 ms
- Case 1:** Passed
- Case 2:** Passed
- Input:** candidates = [10,1,2,7,6,1,5]
target = 8
- Output:** [1,1,6], [1,2,5], [1,7], [2,6]

```
import java.util.*;  
class Solution {  
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {  
        List<List<Integer>> result = new ArrayList<>();  
  
        Arrays.sort(candidates);  
  
        backtrack(candidates, target, 0, new ArrayList<>(), result);  
        return result;  
    }  
    private void backtrack(int[] candidates, int target, int start, List<Integer> current, List<List<Integer>> result) {  
        if (target == 0) {  
            result.add(new ArrayList<Integer>(current));  
            return;  
        }  
        for (int i = start; i < candidates.length; i++) {  
            if (candidates[i] > target) {  
                break;  
            }  
            current.add(candidates[i]);  
            backtrack(candidates, target - candidates[i], i + 1, current, result);  
            current.remove(current.size() - 1);  
        }  
    }  
}
```

4. You are given a 0-indexed array of integers nums of length n. You are initially

positioned at index 0.

Each element $\text{nums}[i]$ 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: $\text{nums} = [2,3,1,1,4]$

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: $\text{nums} = [2,3,0,1,4]$

Output: 2

The screenshot shows a Java code submission interface. At the top, it displays "Accepted" status with 110/110 testcases passed, submitted by Anurag Betal at Feb 16, 2026 21:02. Below this, there are sections for "Runtime" (1 ms, 99.56% beaten) and "Memory" (47.55 MB, 18.69% beaten). A complexity analysis chart shows performance across 100% of the input range. The code editor on the right contains the following Java code:

```
1 class Solution {
2     public int jump(int[] nums) {
3         int jumps = 0;
4         int end = 0;
5         int farthest = 0;
6         for (int i = 0; i < nums.length - 1; i++) {
7             farthest = Math.max(farthest, i + nums[i]);
8             if (i == end) {
9                 jumps++;
10                end = farthest;
11            }
12        }
13    }
}
```

The "Test Result" section shows "Accepted" status with runtime 0 ms. It includes two test cases: Case 1 and Case 2. The "Input" field contains the array [2,3,1,1,4], and the "Output" field shows the result 2. The "Expected" field is also 2.

5. 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 with a successful submission. The code implements a solution to group anagrams. It uses a HashMap where keys are sorted character arrays and values are lists of strings. The runtime is 7ms (76.53% beats) and memory usage is 49.92 MB (38.88% beats).

```
1 import java.util.*;
2 class Solution {
3     public List<List<String>> groupAnagrams(String[] strs) {
4         Map<String, List<String>> map = new HashMap<>();
5
6         for (String str : strs) {
7             char[] chars = str.toCharArray();
8             Arrays.sort(chars);
9             String key = new String(chars);
10
11            map.putIfAbsent(key, new ArrayList<>());
12            map.get(key).add(str);
13        }
14
15    }
16
17 }
```

6. 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:

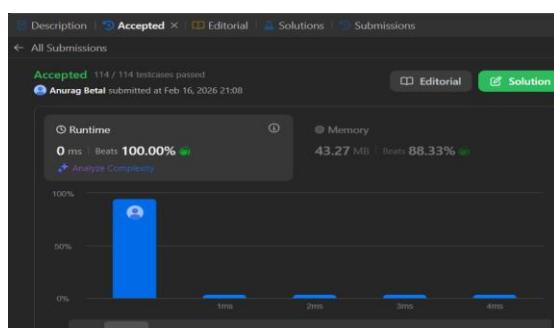
Input: digits = [9]

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 LeetCode submission page. At the top, it says "Accepted" with 114/114 testcases passed, submitted by Anurag Betal at Feb 16, 2026 21:08. Below this, there are two tabs: "Runtime" and "Memory". The "Runtime" tab shows 0 ms | Beats 100.00% with a bar chart. The "Memory" tab shows 43.27 MB | Beats 88.33% with a bar chart. Below these tabs is a "Analyze Complexity" button. On the right side of the page, there is a code editor window titled "Java" with the following code:

```
1 class Solution {
2     public int[] plusOne(int[] digits) {
3         for (int i = digits.length - 1; i >= 0; i--) {
4             if (digits[i] < 9) {
5                 digits[i]++;
6                 return digits;
7             }
8             digits[i] = 0;
9         }
10
11         int[] result = new int[digits.length + 1];
12         result[0] = 1;
13
14         return result;
15     }
16 }
```

At the bottom of the code editor, it says "Saved" and "Ln 15, Col 8". At the very bottom of the page, there are "Testcase" and "Test Result" buttons.

7. 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	1
0	0	0
1	0	1

Input: 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	0

Input: 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]]

Accepted 202 / 202 testcases passed

Anurag Betal submitted at Feb 16, 2026 21:11

Runtime: 1 ms | Beats 60.16% | Memory: 47.40 MB | Beats 70.68%

```

class Solution {
    public void setZeroes(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        boolean firstRowZero = false;
        boolean firstColZero = false;
        for (int j = 0; j < n; j++) {
            if (matrix[0][j] == 0) {
                firstRowZero = true;
                break;
            }
        }
        for (int i = 0; i < m; i++) {
            if (matrix[i][0] == 0) {
                firstColZero = true;
                break;
            }
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (matrix[i][j] == 0) {
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
            }
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (matrix[i][0] == 0 || matrix[0][j] == 0) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
}

```

```

1 class Solution {
2     public void setZeroes(int[][] matrix) {
3         int m = matrix.length;
4         int n = matrix[0].length;
5         boolean firstRowZero = false;
6         boolean firstColZero = false;
7         for (int j = 0; j < n; j++) {
8             if (matrix[0][j] == 0) {
9                 firstRowZero = true;
10            }
11        }
12        for (int i = 1; i < m; i++) {
13            if (matrix[i][0] == 0) {
14                firstColZero = true;
15            }
16        }
17        for (int i = 1; i < m; i++) {
18            for (int j = 1; j < n; j++) {
19                if (matrix[i][j] == 0) {
20                    matrix[i][0] = 0;
21                    matrix[0][j] = 0;
22                }
23            }
24        }
25        for (int i = 1; i < m; i++) {
26            for (int j = 1; j < n; j++) {
27                if (matrix[i][0] == 0 || matrix[0][j] == 0) {
28                    matrix[i][j] = 0;
29                }
30            }
31        }
32    }
33    if (firstRowZero) {
34        for (int j = 0; j < n; j++) {
35            matrix[0][j] = 0;
36        }
37    }
38    if (firstColZero) {
39        for (int i = 0; i < m; i++) {
40            matrix[i][0] = 0;
41        }
42    }
43 }
44 }
45 }

```

8. 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]

Description | Accepted | Editorial | Solutions | Submissions

Submit Ctrl Enter ↵

← All Submissions

Accepted 89 / 89 testcases passed

Anurag Betal submitted at Feb 16, 2026 21:18

Runtime 0 ms | Beats 100.00% ⚡

Memory 43.43 MB | Beats 78.50% ⚡

Analyze Complexity

100%
75%
50%
25%
0%

1ms 2ms 3ms 4ms

Code | Java

```
1 class Solution {  
2     public void sortColors(int[] nums) {  
3         int low = 0;  
4         int mid = 0;  
5         int high = nums.length - 1;  
6  
7         while (mid <= high) {  
8             if (nums[mid] == 0) {  
9                 swap(nums, low, mid);  
10                low++;  
11                mid++;  
12            }  
13            else if (nums[mid] == 1) {  
14                mid++;  
15            }  
16        }  
17    }  
18}
```

Java ▾ Auto

Java ▾ Auto

1 class Solution {
2 public void sortColors(int[] nums) {
3 int low = 0;
4 int mid = 0;
5 int high = nums.length - 1;
6
7 while (mid <= high) {
8 if (nums[mid] == 0) {
9 swap(nums, low, mid);
10 low++;
11 mid++;
12 }
13 else if (nums[mid] == 1) {
14 mid++;
15 }
16 }
17 }
18}

Saved Ln 30, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums = [2,0,2,1,1,0]

Output

[0,0,1,1,2,2]

Expected

9. 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]]

Accepted | Accepted x | Editorial | Solutions | Submissions

Submit Ctrl Enter ↵

All Submissions

Accepted 10 / 10 testcases passed
Anurag Betal submitted at Feb 16, 2026 21:21

Runtime: 1 ms | Beats 87.84% | Memory: 44.27 MB | Beats 57.84%

Analyze Complexity

100%
50%
0%
1ms 2ms 3ms 4ms

Code | Java

```
import java.util.*;  
class Solution {  
    public List<List<Integer>> subsets(int[] nums) {  
        List<List<Integer>> result = new ArrayList<>();  
        backtrack(nums, 0, new ArrayList<>(), result);  
        return result;  
    }  
}
```

Java ▾ Auto

```
import java.util.*;  
class Solution {  
    public List<List<Integer>> subsets(int[] nums) {  
        List<List<Integer>> result = new ArrayList<>();  
        backtrack(nums, 0, new ArrayList<>(), result);  
        return result;  
    }  
}  
private void backtrack(int[] nums, int start,  
    List<Integer> current,  
    List<List<Integer>> result) {  
    result.add(new ArrayList<>(current));  
    for (int i = start; i < nums.length; i++) {  
        current.add(nums[i]);  
        backtrack(nums, i + 1, current, result);  
        current.remove(current.size() - 1);  
    }  
}
```

Ln 24, Col 1

Saved

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums =
[1,2,3]

Output

[[], [1], [1,2], [1,2,3], [1,3], [2], [2,3], [3]]

Expected

10. 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:

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 = "ABCCED"

Output: true

Example 2:

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 = "SEE"

Output: true

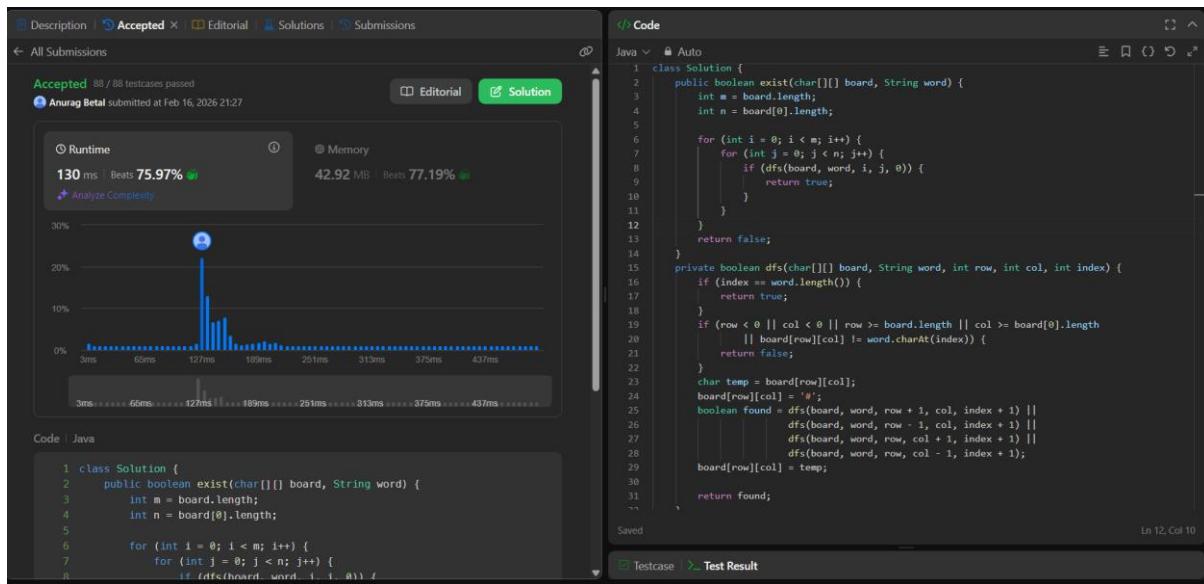
Example 3:

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 =

"ABCB"

Output: false



11. 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.
 - $\text{nums}[a] + \text{nums}[b] + \text{nums}[c] + \text{nums}[d] == \text{target}$

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 LeetCode submission page for a Java solution. The submission was accepted with 294 / 294 testcases passed. The runtime is 19 ms (Beats 80.52%) and the memory usage is 45.90 MB (Beats 62.91%). The code is a two-pointer approach to find all quadruplets that sum up to the target.

```
1 import java.util.*;
2 class Solution {
3     public List<List<Integer>> fourSum(int[] nums, int target) {
4         List<List<Integer>> result = new ArrayList<>();
5         int n = nums.length;
6
7         Arrays.sort(nums);
8
9         for (int i = 0; i < n - 3; i++) {
10             // Skip duplicate first elements
11             if (i > 0 && nums[i] == nums[i - 1]) continue;
12
13             for (int j = i + 1; j < n - 2; j++) {
14                 // Skip duplicate second elements
15                 if (j > i + 1 && nums[j] == nums[j - 1]) continue;
16
17                 int left = j + 1;
18                 int right = n - 1;
19
20                 while (left < right) {
21                     long sum = (long) nums[i] + nums[j] + nums[left] + nums[right];
22
23                     if (sum == target) {
24                         result.add(Arrays.asList(
25                             nums[i], nums[j], nums[left], nums[right]
26                         ));
27
28                         left++;
29                         right--;
30                     } else if (sum < target) {
31                         left++;
32                     } else {
33                         right--;
34                     }
35                 }
36             }
37         }
38
39         return result;
40     }
41 }
42
43
44
45 }
```

The screenshot shows a LeetCode submission page for a Java solution. The submission was accepted with 294 / 294 testcases passed. The runtime is 19 ms (Beats 80.52%) and the memory usage is 45.90 MB (Beats 62.91%). This solution is similar to the first one but uses a different approach to handle the two-pointer search.

```
1 import java.util.*;
2 class Solution {
3     public List<List<Integer>> fourSum(int[] nums, int target) {
4         List<List<Integer>> result = new ArrayList<>();
5         int n = nums.length;
6
7         Arrays.sort(nums);
8
9         int right = n - 1;
10
11         while (left < right) {
12             long sum = (long) nums[i] + nums[j] + nums[left] + nums[right];
13
14             if (sum == target) {
15                 result.add(Arrays.asList(
16                     nums[i], nums[j], nums[left], nums[right]
17                 ));
18
19                 left++;
20                 right--;
21             } else if (sum < target) {
22                 left++;
23             } else {
24                 right--;
25             }
26         }
27
28         return result;
29     }
30 }
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 }
```

12. 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 code submission interface. At the top, it says "Accepted" with 196/196 testcases passed, submitted by Anurag Betal at Feb 16, 2026 21:36. Below this are two tabs: "Editorial" and "Solution". The "Solution" tab is selected. On the left, there's a "Runtime" chart showing performance metrics: 0 ms (Beats 100.00%), 43.84 MB (Beats 65.43%). The chart has a blue bar reaching 100% and several smaller bars for memory usage. On the right, the Java code for the search algorithm is displayed:

```
1 class Solution {
2     public int search(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             }
12             if (nums[left] <= nums[mid]) {
13
14                 if (target >= nums[left] && target < nums[mid]) {
15                     right = mid - 1;
16                 } else {
17                     left = mid + 1;
18                 }
19             } else {
20                 if (target >= nums[mid] && target <= nums[right]) {
21                     left = mid + 1;
22                 } else {
23                     right = mid - 1;
24                 }
25             }
26         }
27         return -1;
28     }
29 }
```

13. 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]

Screenshot of a code submission interface showing runtime and memory statistics for a Java solution.

Runtime: 0 ms | Beats 100.00% (Fastest)

Memory: 48.29 MB | Beats 49.95%

```
Java class Solution { public int[] searchRange(int[] nums, int target) { int[] result = new int[2]; result[0] = findFirst(nums, target); result[1] = findLast(nums, target); return result; } private int findFirst(int[] nums, int target) { int left = 0, right = nums.length - 1; int index = -1; while (left <= right) { int mid = left + (right - left) / 2; if (nums[mid] == target) { index = mid; right = mid - 1; } else if (nums[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return index; } private int findLast(int[] nums, int target) { int left = 0, right = nums.length - 1; int index = -1; while (left <= right) { int mid = left + (right - left) / 2; if (nums[mid] == target) { index = mid; left = mid + 1; } else if (nums[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return index; } }
```

Code: Java

```
1 class Solution {
2     public int[] searchRange(int[] nums, int target) {
3         int[] result = new int[2];
4         result[0] = findFirst(nums, target);
5         result[1] = findLast(nums, target);
6         return result;
7     }
8     private int findFirst(int[] nums, int target) {
9         int left = 0, right = nums.length - 1;
10        int index = -1;
11        while (left <= right) {
12            int mid = left + (right - left) / 2;
13            if (nums[mid] == target) {
14                index = mid;
15                right = mid - 1;
16            } else if (nums[mid] < target) {
17                left = mid + 1;
18            } else {
19                right = mid - 1;
20            }
21        }
22        return index;
23    }
24    private int findLast(int[] nums, int target) {
25        int left = 0, right = nums.length - 1;
26        int index = -1;
27        while (left <= right) {
28            int mid = left + (right - left) / 2;
29            if (nums[mid] == target) {
30                index = mid;
31                left = mid + 1;
32            } else if (nums[mid] < target) {
33                left = mid + 1;
34            } else {
35                right = mid - 1;
36            }
37        }
38        return index;
39    }
40 }
```

Screenshot of a code submission interface showing runtime and memory statistics for a Java solution.

Runtime: 0 ms | Beats 100.00% (Fastest)

Memory: 48.29 MB | Beats 49.95%

```
Java class Solution { public int[] searchRange(int[] nums, int target) { int[] result = new int[2]; result[0] = findFirst(nums, target); result[1] = findLast(nums, target); return result; } private int findFirst(int[] nums, int target) { int left = 0, right = nums.length - 1; int index = -1; while (left <= right) { int mid = left + (right - left) / 2; if (nums[mid] == target) { index = mid; right = mid - 1; } else if (nums[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return index; } private int findLast(int[] nums, int target) { int left = 0, right = nums.length - 1; int index = -1; while (left <= right) { int mid = left + (right - left) / 2; if (nums[mid] == target) { index = mid; left = mid + 1; } else if (nums[mid] < target) { left = mid + 1; } else { right = mid - 1; } } return index; } }
```

Code: Java

```
1 class Solution {
2     public int[] searchRange(int[] nums, int target) {
3         int[] result = new int[2];
4         result[0] = findFirst(nums, target);
5         result[1] = findLast(nums, target);
6         return result;
7     }
8     private int findFirst(int[] nums, int target) {
9         int left = 0, right = nums.length - 1;
10        int index = -1;
11        while (left <= right) {
12            int mid = left + (right - left) / 2;
13            if (nums[mid] == target) {
14                index = mid;
15                right = mid - 1;
16            } else if (nums[mid] < target) {
17                left = mid + 1;
18            } else {
19                right = mid - 1;
20            }
21        }
22        return index;
23    }
24    private int findLast(int[] nums, int target) {
25        int left = 0, right = nums.length - 1;
26        int index = -1;
27        while (left <= right) {
28            int mid = left + (right - left) / 2;
29            if (nums[mid] == target) {
30                index = mid;
31                left = mid + 1;
32            } else if (nums[mid] < target) {
33                left = mid + 1;
34            } else {
35                right = mid - 1;
36            }
37        }
38        return index;
39    }
40 }
```