

# JAVA PABL PRACTICE SHEET – 3

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

1. Output: 4

---

## SOLUTION-

The screenshot shows a LeetCode submission page for a Java solution. The code implements a binary search algorithm to find the index of a target value in a sorted array. The runtime is 0 ms (100.00% beats) and memory usage is 44.66 MB (83.11% beats). The code editor displays the Java code, and the test results show it passed all three test cases (Case 1, Case 2, Case 3) with an accepted status and 0 ms runtime.

```
1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int low = 0, high = nums.length - 1;
4
5         while (low <= high) {
6             int mid = low + (high - low) / 2;
7
8             if (nums[mid] == target) {
```

Accepted 66 / 66 testcases passed  
harshittiwari07 submitted at Feb 15, 2026 21:05

Runtime: 0 ms Beats 100.00%  
Memory: 44.66 MB Beats 83.11%

Testcase | Test Result  
Accepted Runtime: 0 ms  
Case 1 Case 2 Case 3  
Input  
nums = [1,3,5,6]  
target = 5  
Output  
2  
Expected

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:** []

2.

## Solution -

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

- Description:** Accepted 160 / 160 testcases passed
- Editorial:** harshittiwari07 submitted at Feb 15, 2026 21:15
- Runtime:** 3 ms | Beats 36.96%
- Memory:** 45.84 MB | Beats 51.48%
- Code:**

```
1 class Solution {
2     public List<List<Integer>> combinationSum(int[] candidates, int target) {
3         List<List<Integer>> ans = new ArrayList<>();
4         solve(0, candidates, target, new ArrayList<>(), ans);
5         return ans;
6     }
7
8     void solve(int i, int[] candidates, int target, List<Integer> temp, List<List<Integer>>
```
- Test Result:** Accepted Runtime: 0 ms
- Input:** candidates = [2,3,6,7]
- target:** 7
- Output:** [[2,2,3],[7]]
- Expected:** [[2,2,3],[7]]

### 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]  
]
```

## Solution –

The screenshot shows a LeetCode submission page for a Java solution. The top navigation bar includes 'Array', 'Editorial', 'Solutions', and 'Submissions'. The submission details show it was accepted with 176/176 testcases passed by user 'harshitiwari07' at Feb 15, 2026 21:17. The runtime is 5 ms (Beats 99.21%) and memory usage is 44.98 MB (Beats 93.58%). Below this are charts for runtime and memory distribution. The code editor contains the following Java code:

```
1 class Solution {  
2     public List<List<Integer>> combinationSum2(int[] candidates, int target) {  
3         Arrays.sort(candidates);  
4         List<List<Integer>> ans = new ArrayList<>();  
5         solve(0, candidates, target, new ArrayList<>(), ans);  
6         return ans;  
7     }  
8 }
```

The test result section shows the input [10,1,2,7,6,1,5], target = 8, output [[1,1,6],[1,2,5],[1,7],[2,6]], and expected output [[1,1,6],[1,2,5],[1,7],[2,6]].

You are given a **0-indexed** array of integers  $\text{nums}$  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

4.

## Solution-

The screenshot shows a LeetCode submission page for problem 4. The top navigation bar includes 'Array', 'Description', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The submission was made by user 'harshittiwari07' on Feb 15, 2026, at 21:22. It has been accepted with 110/110 testcases passed. The runtime is 1 ms (99.54% beaten) and memory usage is 47.39 MB (55.60% beaten). The code editor shows a Java solution with a time complexity of O(n). The test result section shows 'Accepted' with runtime 0 ms. The input is '[2,3,1,1,4]' and the output is '2'. The expected output is also '2'. A note at the bottom right says 'Contribute a testcase'.

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

---

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:** `[[""]]`

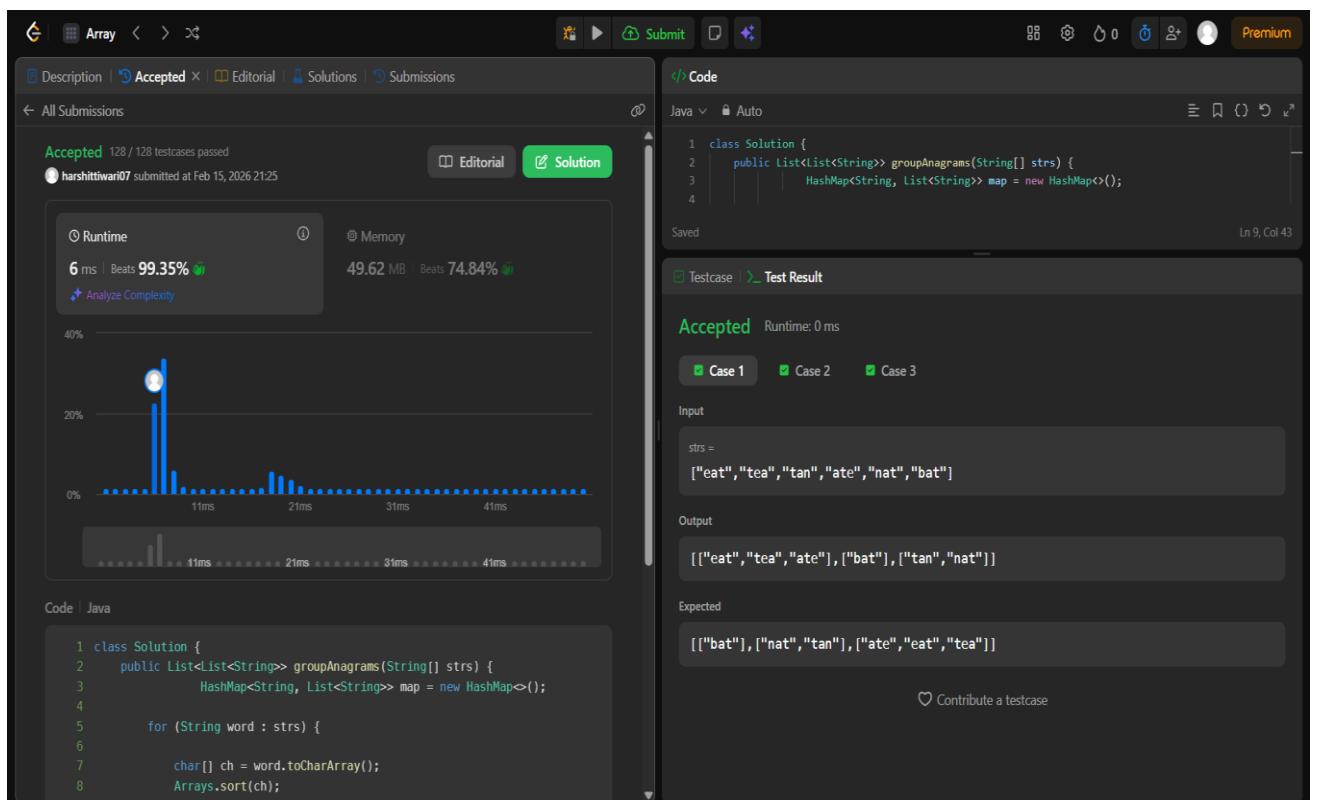
### Example 3:

**Input:** `strs = ["a"]`

**Output:** `[["a"]]`

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

## 5. Solution-



The screenshot shows a LeetCode submission page for the 'Group Anagrams' problem. The code is written in Java and uses a HashMap to group anagrams. It has been accepted with 128/128 testcases passed. The runtime is 6 ms (99.35% beats) and memory usage is 49.62 MB (74.84% beats). The code editor shows the Java implementation, and the test results section shows the input, output, and expected results for three cases.

```
1 class Solution {
2     public List<List<String>> groupAnagrams(String[] strs) {
3         HashMap<String, List<String>> map = new HashMap<>();
4
5         for (String word : strs) {
6             char[] ch = word.toCharArray();
7             Arrays.sort(ch);
8             String sortedWord = new String(ch);
9
10            if (map.containsKey(sortedWord)) {
11                map.get(sortedWord).add(word);
12            } else {
13                List<String> list = new ArrayList<>();
14                list.add(word);
15                map.put(sortedWord, list);
16            }
17        }
18
19        return new ArrayList<>(map.values());
20    }
21}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

Output

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

Expected

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

Contribute a testcase

You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the  $i^{\text{th}}$  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]`.

6.

## Solution-

The screenshot shows a LeetCode problem submission page for a Java solution. The submission was accepted with 114/114 testcases passed. The author is harshittiwari07 and it was submitted at Feb 15, 2026 21:31. The runtime is 0 ms (100.00% beats) and memory usage is 43.39 MB (76.27% beats). The complexity analysis shows a bar chart with a single bar reaching 100%.

**Testcase | Test Result**

**Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

**Input**:  
digits =  
[1,2,3]

**Output**:  
[1,2,4]

**Expected**:  
[1,2,4]

Contribute a testcase

**Code | Java**

```
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         }
9     }
10 }
```

Given an  $m \times n$  integer matrix  $\text{matrix}$ , 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

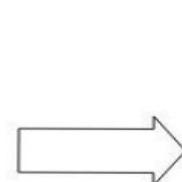
Take Home

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

7.

## Solution-

The screenshot shows a LeetCode problem page for an Array problem. The problem statement is identical to the one above. The solution has been accepted with 202/202 testcases passed by user harshittiwari07 at Feb 15, 2026 21:34. The runtime is 1 ms (Beats 60.26%) and memory usage is 47.35 MB (Beats 70.32%). The complexity analysis is shown as "Analyze Complexity".

**Runtime:** 1 ms | Beats 60.26% | Analyze Complexity

**Memory:** 47.35 MB | Beats 70.32% | Analyze Complexity

**Testcase:** Accepted | Test Result

**Case 1:** Accepted | Runtime: 0 ms

**Input:** matrix = [[1,1,1],[1,0,1],[1,1,1]]

**Output:** [[1,0,1],[0,0,0],[1,0,1]]

**Expected:** [[1,0,1],[0,0,0],[1,0,1]]

**Code:** Java

```
1 class Solution {
2     public void setZeroes(int[][] matrix) {
3         int m = matrix.length;
4         int n = matrix[0].length;
5
6         boolean firstRowZero = false;
7         boolean firstColZero = false;
```

8.

You are given an  $m \times n$  integer matrix  $\text{matrix}$  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

**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

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

- Description:** Accepted 133 / 133 testcases passed by harshtiwari07 at Feb 15, 2026 21:39.
- Code:** Java code for a Solution class with a searchMatrix method.
- Runtime:** 0 ms | Beats 100.00%.
- Memory:** 43.94 MB | Beats 61.79%.
- Complexity:** Analyze Complexity (chart showing ~100% runtime vs 1ms-4ms).
- Test Result:** Accepted | Runtime: 0 ms | Case 1, Case 2.
- Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]] | target = 3 | Output = true.

```
1 class Solution {
2     public boolean searchMatrix(int[][] matrix, int target) {
3         int m = matrix.length;
4         int n = matrix[0].length;
5
6         int low = 0;
7         int high = (m*n) - 1;
8
9         while (low <= high){
10             int mid = low + (high - low)/2;
11
12             int row = mid / n;
13             int col = mid % n;
```

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]`

9.

### Example 2:

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

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

The screenshot shows a Java solution for the Dutch National Flag problem. The code implements the Dutch National Flag algorithm (LSD Radix Sort) to sort an array of integers 0, 1, and 2. The solution is accepted, having passed 89/89 test cases. The runtime is 0 ms (Beats 100.00%) and memory usage is 43.70 MB (Beats 41.66%). The complexity analysis chart shows a single dominant bar at 0 ms. The test result section shows the input `[2,0,2,1,1,0]`, output `[0,0,1,1,2,2]`, and expected output `[0,0,1,1,2,2]`.

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

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]]

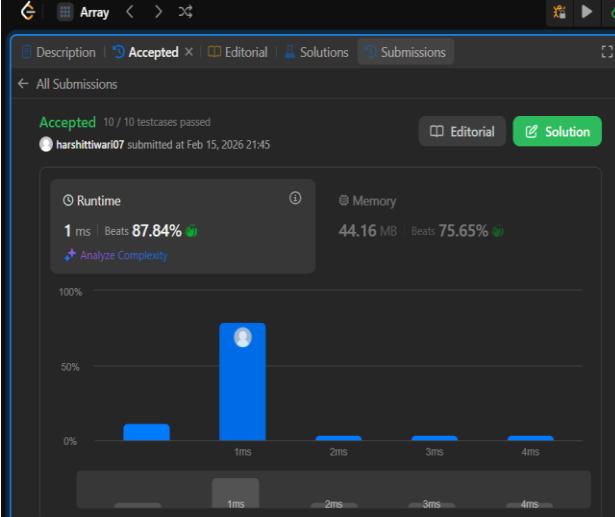
### Example 2:

**Input:** nums = [0]

**Output:** [[], [0]]

10.

## Solution-



Accepted 10 / 10 testcases passed  
harshittiwari07 submitted at Feb 15, 2026 21:45

Runtime: 1 ms | Beats 87.84%  
Memory: 44.16 MB | Beats 75.65%

Analyze Complexity

Code | Java

```
1 class Solution {  
2     public List<List<Integer>> subsets(int[] nums) {  
3         List<List<Integer>> ans = new ArrayList<>();  
4         solve(0, nums, new ArrayList<>(), ans);  
5         return ans;  
6     }  
7     void solve(int index, int[] nums, List<Integer> temp, List<List<Integer>> ans) {  
8         ans.add(new ArrayList<>(temp));  
9         for (int i = index; i < nums.length; i++) {  
10             temp.add(nums[i]);  
11             solve(i + 1, nums, temp, ans);  
12             temp.remove(temp.size() - 1);  
13         }  
14     }  
15 }
```

Code

```
Java
```

Java

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums = [1,2,3]

Output

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

Expected

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

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 = "ABCED"

**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

```
Java | Auto
1 class Solution {
2     public boolean exist(char[][] board, String word) {
3         int m = board.length;
4         int n = board[0].length;
5
6         for (int i = 0; i < m; i++) {
7             for (int j = 0; j < n; j++) {
8                 if (dfs(board, word, i, j, 0)) {
9                     return true;
10                }
11            }
12        }
13    }
14
15    private boolean dfs(char[][] board, String word, int i, int j, int index) {
16        if (index == word.length()) {
17            return true;
18        }
19
20        if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != word.charAt(index)) {
21            return false;
22        }
23
24        char temp = board[i][j];
25        board[i][j] = '#';
26
27        boolean found = dfs(board, word, i + 1, j, index + 1) ||
28                      dfs(board, word, i - 1, j, index + 1) ||
29                      dfs(board, word, i, j + 1, index + 1) ||
30                      dfs(board, word, i, j - 1, index + 1);
31
32        board[i][j] = temp;
33
34        return found;
35    }
36}
37
38 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

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

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]]

## Solution-

The screenshot shows a LeetCode problem page for "Word Search". The top navigation bar includes tabs for "Description", "Accepted", "Editorial", "Solutions", and "Submissions". The "Accepted" tab is selected. The main content area displays the following information:

- Status:** Accepted (88 / 88 testcases passed)
- Author:** harshitiwan07 submitted at Feb 15, 2026 21:48
- Runtime:** 129 ms | Beats 81.51%
- Memory:** 43.22 MB | Beats 30.01%
- Analyze Complexity:** A chart showing execution time distribution across various input sizes.
- Code (Java):** The provided Java solution for the problem.
- Test Result:** All test cases passed (Case 1, Case 2, Case 3).
- Input:** board = [[“A”, “B”, “C”, “E”], [“S”, “F”, “C”, “S”], [“A”, “D”, “E”, “E”]] and word = “ABCCED”.
- Output:** true.

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

13.

```

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

```

Accepted Runtime: 0 ms  
Memory: 43.91 MB

Testcase | Test Result  
Accepted Runtime: 0 ms  
Case 1 Case 2 Case 3

Input:  
nums = [4,5,6,7,0,1,2]  
target = 0

Output: 4

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]

14.

The screenshot shows a LeetCode submission page for a Java solution. The submission was accepted with 88/88 testcases passed. The runtime is 0 ms (Beats 100.00%) and memory usage is 48.58 MB (Beats 11.72%). The code editor displays the following Java code:

```
1 class Solution {
2     public int[] searchRange(int[] nums, int target) {
3         int first = findFirst(nums, target);
4         int last = findLast(nums, target);
5
6         return new int[]{first, last};
7     }
8
9     int findFirst(int[] nums, int target) {
10        int low = 0, high = nums.length - 1;
11        int ans = -1;
12
13        while (low <= high) {
14            int mid = (low + high) / 2;
15
16            if (nums[mid] == target) {
17                ans = mid;
18                high = mid - 1;
19            } else if (nums[mid] < target) {
20                low = mid + 1;
21            } else {
22                high = mid - 1;
23            }
24        }
25
26        return ans;
27    }
28
29    int findLast(int[] nums, int target) {
30        int low = 0, high = nums.length - 1;
31        int ans = -1;
32
33        while (low <= high) {
34            int mid = (low + high) / 2;
35
36            if (nums[mid] == target) {
37                ans = mid;
38                low = mid + 1;
39            } else if (nums[mid] < target) {
40                low = mid + 1;
41            } else {
42                high = mid - 1;
43            }
44        }
45
46        return ans;
47    }
48}
```

The test result section shows Case 1, Case 2, and Case 3 all accepted with 0 ms runtime. The input is [5,7,7,8,8,10] and the target is 8, resulting in the output [3,4].