

## **Experiment List for Programming Ability and Logic Building-2**

Name : Ikshu Goel

Batch : 2CSE5

Roll Number : 2410030774

Week : 09/02/26 to 14/02/26

Lecture : 1 & 2

# **EXPERIMENT 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:**

nums = [1,3,5,6], target = 5

2

### Example 2:

nums = [1,3,5,6], target = 2

1

### Example 3:

nums = [1,3,5,6], target = 7

4

### Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- nums contains **distinct** values sorted in **ascending** order.
- $-10^4 \leq \text{target} \leq 10^4$

The screenshot shows a LeetCode problem page for "Search Insert Position". The problem description states: "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." It requires an algorithm with  $O(\log n)$  runtime complexity. The code area contains a Java solution:

```
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) / 2;
7
8             if (nums[mid] == target)
9                 return mid;
10            else if (nums[mid] < target)
11                low = mid + 1;
12            else
13                high = mid - 1;
14        }
15        return low;
16    }
17}
```

The "Testcase" section shows the input: nums = [1,3,5,6], target = 7, and the output: 4. The "Test Result" section shows "Accepted" with a runtime of 0 ms. The constraints listed are:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- nums contains **distinct** values sorted in **ascending** order.
- $-10^4 \leq \text{target} \leq 10^4$

At the bottom, there are statistics: 18.6K submissions, 424 accepted solutions, and 331 online users.

# EXPERIMENT 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:

```
candidates = [2,3,6,7], target = 7  
[[2,2,3],[7]]
```

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:

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

### Example 3:

candidates = [2], target = 1  
[]

### Constraints:

- $1 \leq \text{candidates.length} \leq 30$
- $2 \leq \text{candidates}[i] \leq 40$
- All elements of candidates are **distinct**.
- $1 \leq \text{target} \leq 40$

The screenshot shows a LeetCode problem page for 'Combination Sum'. The problem description is as follows:

**39. Combination Sum**

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 code editor shows a Java solution:

```
import java.util.*;
class Solution {
    List<List<Integer>> ans = new ArrayList<>();
    public List<List<Integer>> combinationSum(int[] c, int target) {
        solve(c, target, 0, new ArrayList<>());
        return ans;
    }
    void solve(int[] c, int target, int i, List<Integer> list) {
        if (target == 0) {
            ans.add(new ArrayList<>(list));
            return;
        }
        if (i == c.length || target < 0) return;
        list.add(c[i]);
        solve(c, target - c[i], i, list);
        list.remove(list.size() - 1);
        solve(c, target, i + 1, list);
    }
}
```

The status bar indicates the code has been saved and is 25 lines long. The test result section shows "Accepted" with runtime 0 ms and three test cases passed.

# EXPERIMENT 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:**

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

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

**Example 2:**

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

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

### Constraints:

- $1 \leq \text{candidates.length} \leq 100$
- $1 \leq \text{candidates}[i] \leq 50$
- $1 \leq \text{target} \leq 30$

The screenshot shows a LeetCode problem page for "40. Combination Sum II". The problem description and examples are visible on the left, and the Java code submission is on the right. The code is a backtracking solution that sorts the candidates and uses a helper function to find all unique combinations that sum up to the target. The submission was accepted with a runtime of 1 ms.

```
import java.util.*;  
class Solution {  
    List<List<Integer>> ans = new ArrayList<>();  
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {  
        Arrays.sort(candidates);  
        solve(candidates, target, 0, new ArrayList<>());  
        return ans;  
    }  
    void solve(int[] arr, int target, int idx, List<Integer> list) {  
        if (target == 0) {  
            ans.add(new ArrayList<>(list));  
            return;  
        }  
        for (int i = idx; i < arr.length; i++) {  
            if (i > idx && arr[i] == arr[i - 1]) continue;  
            if (arr[i] > target) break;  
            list.add(arr[i]);  
            solve(arr, target - arr[i], i + 1, list);  
            list.remove(list.size() - 1);  
        }  
    }  
}
```

Testcase | Test Result  
Accepted Runtime: 1 ms

# EXPERIMENT 4

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:

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

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:

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

## Constraints:

- $1 \leq \text{nums.length} \leq 10^4$

- $0 \leq \text{nums}[i] \leq 100$
- It's guaranteed that you can reach  $\text{nums}[n - 1]$ .

The screenshot shows a LeetCode problem page for "Jump Game II".

**Description:** 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
```

**Constraints:**

- $1 \leq \text{nums.length} \leq 10^4$

16.4K Submissions | 272 Favorites | 210 Online

**Code:**

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

Saved Ln 16, Col 1

**Testcase | Test Result**

**Accepted** Runtime: 0 ms

Case 1 Case 2

Input

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

Output

# EXPERIMENT 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"]]`

## Constraints:

- $1 \leq \text{strs.length} \leq 10^4$
- $0 \leq \text{strs[i].length} \leq 100$
- `strs[i]` consists of lowercase English letters.

The screenshot shows a LeetCode problem page for "Group Anagrams".

**Description:** 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"]]
```

**Code:**

```
1 import java.util.*;
2
3 class Solution {
4     public List<List<String>> groupAnagrams(String[] strs) {
5         Map<String, List<String>> map = new HashMap<>();
6
7         for (String s : strs) {
8             char[] ch = s.toCharArray();
9             Arrays.sort(ch);
10            String key = new String(ch);
11
12            map.putIfAbsent(key, new ArrayList<>());
13            map.get(key).add(s);
14        }
15        return new ArrayList<>(map.values());
16    }
17 }
18
```

**Testcase:** Accepted Runtime: 0 ms

**Input:**

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

**Output:**

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

# EXPERIMENT 6

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:

`digits = [1,2,3]`

`[1,2,4]`

The array represents the integer 123.

Incrementing by one gives  $123 + 1 = 124$ .

Thus, the result should be `[1,2,4]`.

### Example 2:

`digits = [4,3,2,1]`

`[4,3,2,2]`

The array represents the integer 4321.

Incrementing by one gives  $4321 + 1 = 4322$ .

Thus, the result should be `[4,3,2,2]`.

### Example 3:

`digits = [9]`

`[1,0]`

The array represents the integer 9.

Incrementing by one gives  $9 + 1 = 10$ .

Thus, the result should be `[1,0]`.

### Constraints:

- $1 \leq \text{digits.length} \leq 100$
- $0 \leq \text{digits}[i] \leq 9$
- `digits` does not contain any leading 0's.

The screenshot shows a LeetCode problem page for "Plus One". The problem description states: "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 shows input [1,2,3] and output [1,2,4]. Example 2 shows input [4,3,2,1] and output [4,3,2,2]. Example 3 shows input [9] and output [1,0]. The Java code submitted is:

```
public int[] plusOne(int[] d) {
    for (int i = d.length - 1; i >= 0; i--) {
        if (d[i] < 9) {
            d[i]++;
            return d;
        }
        d[i] = 0;
    }
    int[] res = new int[d.length + 1];
    res[0] = 1;
    return res;
}
```

The test results show the code was accepted with a runtime of 0 ms, passing all three test cases.

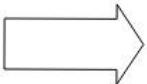
# EXPERIMENT 7

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



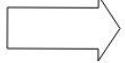
1	0	1
0	0	0
1	0	1

matrix =

```
[[1,1,1],[1,0,1],[1,1,1]]  
[[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
0	4	5	0
0	3	1	0

matrix =

```
[[0,1,2,0],[3,4,5,2],[1,3,1,5]]  
[[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

### Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix[0].length}$
- $1 \leq m, n \leq 200$
- $-2^{31} \leq \text{matrix}[i][j] \leq 2^{31} - 1$

The screenshot shows a LeetCode problem page for "Set Matrix Zeroes".

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

**Example 1:**

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

**Code:**

```

4   boolean[] col = new boolean[m[0].length];
5
6   for (int i = 0; i < m.length; i++) {
7       for (int j = 0; j < m[0].length; j++) {
8           if (m[i][j] == 0) {
9               row[i] = true;
10              col[j] = true;
11           }
12
13           for (int i = 0; i < m.length; i++) {
14               for (int j = 0; j < m[0].length; j++) {
15                   if (row[i] || col[j])
16                       m[i][j] = 0;
17               }
18           }
19

```

**Test Result:** Accepted Runtime: 0 ms  
Case 1 Case 2

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

# EXPERIMENT 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

matrix =

`[[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3`

`true`

#### Example 2:

1	3	5	7
10	11	16	20
23	30	34	60

matrix =

`[[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13`

`false`

## Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix[i].length}$
- $1 \leq m, n \leq 100$
- $-10^4 \leq \text{matrix}[i][j], \text{target} \leq 10^4$

The screenshot shows the LeetCode platform displaying problem 74. Search a 2D Matrix. The problem description and constraints are visible at the top. Below the code editor, there is a grid of three rows and four columns of numbers:

1	3	5	7
10	11	16	20
23	30	34	60

The code editor contains a Java solution using binary search:

```
int n = matrix.length;
int m = matrix[0].length;

int low = 0, high = n * m - 1;

while (low <= high) {
    int mid = (low + high) / 2;
    int val = matrix[mid / m][mid % m];

    if (val == target) return true;
    else if (val < target) low = mid + 1;
    else high = mid - 1;
}
return false;
```

The test result section shows the code was accepted with a runtime of 0 ms, passing both Case 1 and Case 2.

# EXPERIMENT 9

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:

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

## Example 2:

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

## Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 300$
- $\text{nums}[i]$  is either 0, 1, or 2.

The screenshot shows a LeetCode problem page for "Sort Colors".

**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.

**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`
- `nums[i]` is either `0`, `1`, or `2`.

**Code:**

```
class Solution {
    public void sortColors(int[] nums) {
        int c0 = 0, c1 = 0, c2 = 0;

        for (int x : nums) {
            if (x == 0) c0++;
            else if (x == 1) c1++;
            else c2++;
        }

        int i = 0;
        while (c0-- > 0) nums[i++] = 0;
        while (c1-- > 0) nums[i++] = 1;
        while (c2-- > 0) nums[i++] = 2;
    }
}
```

**Test Result:** Accepted, Runtime: 0 ms

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

# EXPERIMENT 10

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:

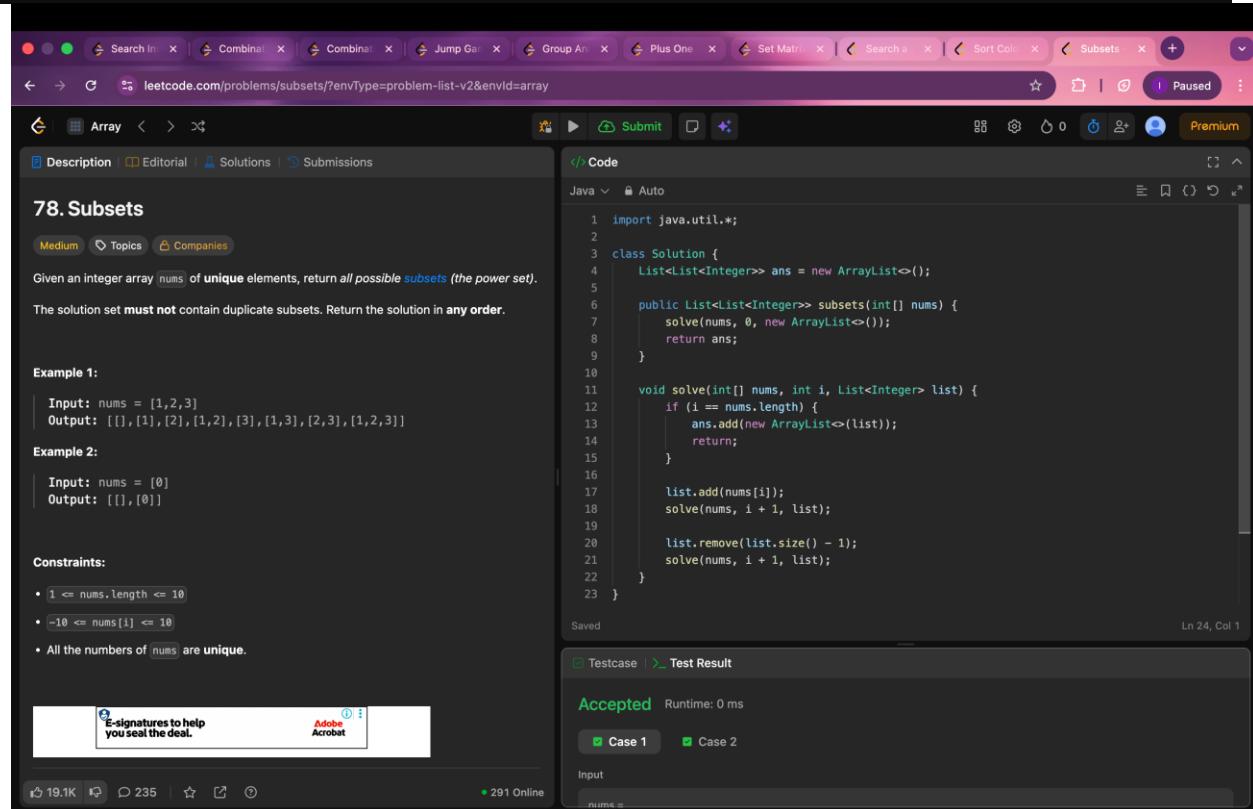
`nums = [1,2,3]`  
`[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]`

### Example 2:

`nums = [0]`  
`[[],[0]]`

### Constraints:

- $1 \leq \text{nums.length} \leq 10$
- $-10 \leq \text{nums}[i] \leq 10$
- All the numbers of `nums` are **unique**.



The screenshot shows a LeetCode problem page for "78. Subsets". The problem statement is identical to the one above. The "Code" section contains the following Java code:

```
1 import java.util.*;
2
3 class Solution {
4     List<List<Integer>> ans = new ArrayList<>();
5
6     public List<List<Integer>> subsets(int[] nums) {
7         solve(nums, 0, new ArrayList<>());
8         return ans;
9     }
10
11    void solve(int[] nums, int i, List<Integer> list) {
12        if (i == nums.length) {
13            ans.add(new ArrayList<>(list));
14            return;
15        }
16
17        list.add(nums[i]);
18        solve(nums, i + 1, list);
19
20        list.remove(list.size() - 1);
21        solve(nums, i + 1, list);
22    }
23 }
```

The "Test Result" section shows "Accepted" with a runtime of 0 ms, having passed both "Case 1" and "Case 2". The input field shows "nums = [1,2,3]".

# EXPERIMENT 11

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

board =

`[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"`  
`true`

### Example 2:

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

board =

`[["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"`  
`true`

### Example 3:

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

board =

```
[[ "A", "B", "C", "E"], [ "S", "F", "C", "S"], [ "A", "D", "E", "E"]], word = "ABCB"
false
```

### Constraints:

- $m == \text{board.length}$
- $n = \text{board}[i].length$
- $1 \leq m, n \leq 6$
- $1 \leq \text{word.length} \leq 15$
- board and word consists of only lowercase and uppercase English letters.

The screenshot shows a LeetCode problem page for "Word Search". The problem description and example 1 are visible. Example 1 shows a 3x4 grid of characters:

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

The Java code implements a depth-first search (DFS) algorithm to find the word "ABCED" in the board. The code is as follows:

```

Java
public boolean exist(char[][] b, String w, int i, int j, int k) {
    if (k == w.length()) return true;
    if (i < 0 || j < 0 || i >= b.length || j >= b[0].length || b[i][j] != w.charAt(k))
        return false;
    char temp = b[i][j];
    b[i][j] = '#';
    boolean found = dfs(b, w, i + 1, j, k + 1) ||
                    dfs(b, w, i - 1, j, k + 1) ||
                    dfs(b, w, i, j + 1, k + 1) ||
                    dfs(b, w, i, j - 1, k + 1);
    b[i][j] = temp;
    return found;
}

```

The code is accepted with a runtime of 0 ms. The input for the test case is:

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

Example 2: [redacted]

17.5K views, 332 upvotes, 292 online users.

## EXPERIMENT 12

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:

`nums = [1,0,-1,0,-2,2], target = 0`  
`[[−2,−1,1,2],[−2,0,0,2],[−1,0,0,1]]`

#### Example 2:

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

#### Constraints:

- $1 \leq \text{nums.length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

The screenshot shows a LeetCode problem page for "18. 4Sum". The Java code provided is as follows:

```
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         Arrays.sort(nums);
7         for (int i = 0; i < n - 3; i++) {
8             if (i > 0 && nums[i] == nums[i - 1]) continue;
9             for (int j = i + 1; j < n - 2; j++) {
10                 if (j > i + 1 && nums[j] == nums[j - 1]) continue;
11                 int left = j + 1;
12                 int right = n - 1;
13                 while (left < right) {
14                     long sum = (long) nums[i] + nums[j] + nums[left] + nums[right];
15                     if (sum == target) {
16                         result.add(Arrays.asList(nums[i], nums[j], nums[left], nums[right]));
17                         while (left < right && nums[left] == nums[left + 1]) left++;
18                         while (left < right && nums[right] == nums[right - 1]) right--;
19                         left++;
20                         right--;
21                     }
22                 }
23             }
24         }
25     }
26 }
27
```

The code is saved and has an accepted status with a runtime of 1 ms.

# EXPERIMENT 13

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 [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 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:

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

4

### Example 2:

nums = [4,5,6,7,0,1,2], target = 3

-1

### Example 3:

nums = [1], target = 0

-1

### Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- All values of nums are **unique**.
- nums is an ascending array that is possibly rotated.
- $-10^4 \leq \text{target} \leq 10^4$

The screenshot shows a LeetCode problem page for "Search in Rotated Sorted Array".

**Description:** 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 `[nums[0], nums[1], ..., nums[n-1], nums[0], nums[1], ..., nums[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`.

**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`

**Constraints:** `1 \leq \text{nums.length} \leq 5000`, `0 \leq \text{nums}[i] \leq 1000`, all elements in `nums` are **unique**, and `nums` is **guaranteed** to be **rotated** at some index.

**Code:**

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

**Testcase:** Accepted Runtime: 0 ms  
Case 1 Case 2 Case 3

# EXPERIMENT 14

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:

nums = [5,7,7,8,8,10], target = 8  
[3,4]

### Example 2:

nums = [5,7,7,8,8,10], target = 6  
[-1,-1]

### Example 3:

nums = [], target = 0  
[-1,-1]

### Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- nums is a non-decreasing array.

- $-10^9 \leq \text{target} \leq 10^9$

**34. Find First and Last Position of Element in Sorted Array**

**Medium** | Topics | Companies

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with  $O(\log n)$  runtime complexity.

**Example 1:**

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

**Example 2:**

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

**Example 3:**

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

**Constraints:**

- $0 \leq \text{nums.length} \leq 10^5$

23.1K Submissions | 354 Favorites | 313 Online | Accepted Runtime: 0 ms

```

1 class Solution {
2     public int[] searchRange(int[] nums, int target) {
3         int[] ans = {-1, -1};
4
5         for (int i = 0; i < nums.length; i++) {
6             if (nums[i] == target) {
7                 ans[0] = i;
8                 break;
9             }
10        }
11
12        for (int i = nums.length - 1; i >= 0; i--) {
13            if (nums[i] == target) {
14                ans[1] = i;
15                break;
16            }
17        }
18
19        return ans;
20    }
21 }
22

```

Saved | Ln 22, Col 1

**Testcase** | **Test Result**

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

```
nums =
15.7.7.8.8.101
```