

**EXPERIMENT LIST FOR PROGRAMMING ABILITY AND LOGIC
BUILDING – 21**

NAME : IKSHU GOEL

BATCH : 2CSE5

ROLL NUMBER : 2410030774

WEEK : 19/1/26 - 25/1/26

LECTURE : 2

EXPERIMENT 1

Given an array **arr**, rotate the array by one position in clockwise direction.

Examples:

Input: arr[] = [1, 2, 3, 4, 5]

Output: [5, 1, 2, 3, 4]

Explanation: If we rotate arr by one position in clockwise 5 come to the front and

remaining those are shifted to the end.

The screenshot shows a web-based Java code editor interface. The code editor has a dark theme with light-colored syntax highlighting. The code itself is a Java class named `Solution` with a single method `rotate`. The code rotates an array by shifting elements from index 1 to index 0 and setting the last element as the first. The code editor includes a toolbar with tabs for Java (selected), Start Timer, and various file operations. On the left, there's an Output Window and a Compilation Results section showing "Compilation Completed". The input field contains "1 2 3 4 5" and the output field shows "5 1 2 3 4", which matches the expected output. At the bottom, there are buttons for Custom Input, Compile & Run, and Submit.

```
1- class Solution {  
2-     public void rotate(int[] arr) {  
3-         int last = arr[arr.length - 1];  
4-         for (int i = arr.length - 1; i > 0; i--) {  
5-             arr[i] = arr[i - 1];  
6-         }  
7-         arr[0] = last;  
8-     }  
9- }  
10- }  
11- }  
12- }
```

EXPERIMENT 2

You are given an integer array **arr[]**. You need to find the **maximum** sum of a

subarray (containing at least one element) in the array **arr[]**.

Note : A **subarray** is a continuous part of an array.

Examples:

Input: arr[] = [2, 3, -8, 7, -1, 2, 3]

Output: 11

Explanation: The subarray [7, -1, 2, 3] has the largest sum 11.

The screenshot shows a web browser window for 'Kadane's Algorithm | Practice' on the GeeksforGeeks website. The page includes a navigation bar with links for Offers Ending, Courses, Tutorials, Practice, and Jobs. A search bar is at the top right. The main area has tabs for Problem, Editorial, Submissions, and Comments. Below these are sections for Output Window and Compilation Results. The Compilation Completed section shows the following code:

```
Java (21) Start Timer
1- class Solution {
2-     long maxSubarraySum(int[] arr) {
3-         long maxSoFar = arr[0];
4-         long curr = arr[0];
5-
6-         for (int i = 1; i < arr.length; i++) {
7-             curr = Math.max(arr[i], curr + arr[i]);
8-             maxSoFar = Math.max(maxSoFar, curr);
9-         }
10-     }
11-     return maxSoFar;
12- }
```

The input field contains 'arr[] = [1, 2, 3, -2, 5]'. The output field shows '9'. The expected output field also shows '9'. At the bottom are buttons for Custom Input, Compile & Run, and Submit.

EXPERIMENT 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

Search Insert Position - LeetCode

leetcode.com/problems/search-insert-position/?envType=problem-list-v2&envId=array

Array

Description Editorial Solutions Submissions

35. Search Insert Position

Easy Topics Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

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

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Constraints:

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

18.5K 413 335 Online

Code

```
Java Auto
```

```
1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int l = 0, r = nums.length - 1;
4
5         while (l <= r) {
6             int mid = (l + r) / 2;
7             if (nums[mid] == target)
8                 return mid;
9             else if (nums[mid] < target)
10                l = mid + 1;
11             else
12                r = mid - 1;
13         }
14         return l;
15     }
16 }
```

Saved Ln 15, Col 6

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

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

target =

```
5
```

Output

335 Online

Search Insert Position - LeetCode

leetcode.com/problems/search-insert-position/?envType=problem-list-v2&envId=array

Array

Description Editorial Solutions Submissions

35. Search Insert Position

Easy Topics Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

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

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Constraints:

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

18.5K 413 336 Online

Code

```
Java Auto
```

```
1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int l = 0, r = nums.length - 1;
4
5         while (l <= r) {
6             int mid = (l + r) / 2;
7             if (nums[mid] == target)
8                 return mid;
9             else if (nums[mid] < target)
10                l = mid + 1;
11             else
12                r = mid - 1;
13         }
14         return l;
15     }
16 }
```

Saved Ln 15, Col 6

Testcase Test Result

target =

```
5
```

Output

```
2
```

Expected

```
2
```

Contribute a testcase

EXPERIMENT 4

Given an array of integers `nums` and an integer `target`, return *indices of the two*

numbers such that they add up to target.

You may assume that each input would have **exactly one solution**, and you may

not use the *same* element twice.

You can return the answer in any order.

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9`

Output: [0,1]

Explanation: Because $\text{nums}[0] + \text{nums}[1] == 9$, we return [0, 1].

The screenshot shows a LeetCode problem page for "Two Sum". The problem description asks for indices of two numbers in an array that add up to a target value. The Java code uses a hash map to store the numbers and their indices. It iterates through the array, calculates the complement (target - current number), and checks if it's already in the map. If yes, it returns the current index and the index from the map. If not, it adds the current number and its index to the map.

```
1 import java.util.*;
2
3 class Solution {
4     public int[] twoSum(int[] nums, int target) {
5         HashMap<Integer, Integer> map = new HashMap<>();
6
7         for (int i = 0; i < nums.length; i++) {
8             int rem = target - nums[i];
9             if (map.containsKey(rem)) {
10                 return new int[]{map.get(rem), i};
11             }
12             map.put(nums[i], i);
13         }
14     }
15 }
```

The code is saved and has a runtime of 0 ms. The test result shows all three cases (Case 1, Case 2, Case 3) are accepted. The input provided is [2,7,11,15] and the target is 9.

Two Sum - LeetCode

leetcode.com/problems/two-sum/?envType=problem-list-v2&envId=array

Paused

Array

Description | Editorial | Solutions | Submissions

1. Two Sum

Easy | Topics | Companies | Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have **exactly one solution**, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- $2 \leq \text{nums.length} \leq 10^4$

67K | 1.8K | 2990 Online | Contribute a testcase

Java

```
import java.util.*;
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int rem = target - nums[i];
            if (map.containsKey(rem)) {
                return new int[]{map.get(rem), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{};
    }
}
```

Ln 17, Col 1

Testcase | Test Result

target = 9

Output

[0,1]

Expected

[0,1]

EXPERIMENT 5

You are given an array **arr[]** of non-negative numbers. Each number tells you the **maximum number of steps** you can jump forward from that position.

For example:

- If **arr[i] = 3**, you can jump to index **i + 1, i + 2, or i + 3** from position **i**.
- If **arr[i] = 0**, you **cannot jump forward** from that position.

Your task is to find the **minimum number of jumps** needed to move from the **first** position in the array to the **last** position.

Note: Return **-1** if you can't reach the end of the array.

Examples :

Input: arr[] = [1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9]

Output: 3

Explanation: First jump from 1st element to 2nd element with value 3. From here

we jump to 5th element with value 9, and from here we will jump to the last.

Minimum Jumps | Practice | +

geeksforgeeks.org/problems/minimum-number-of-jumps-1587115620/1

Offers Ending Courses Tutorials Practice Jobs

Search... Problem Editorial Submissions Comments

Java (21) Start Timer

Output Window X

Compilation Results Custom Input

Compilation Completed

Case 1

Input: arr[] = 1 3 5 8 9 2 6 7 6 8 9

Your Output: 3

Expected Output: 3

```
1- class Solution {
2-     public int minJumps(int[] arr) {
3-         if (arr.length <= 1)
4-             return 0;
5-         if (arr[0] == 0)
6-             return -1;
7-
8-         int maxReach = arr[0];
9-         int steps = arr[0];
10-        int jumps = 1;
11-
12-        for (int i = 1; i < arr.length; i++) {
13-            if (i == arr.length - 1)
14-                return jumps;
15-
16-            maxReach = Math.max(maxReach, i + arr[i]);
17-            steps--;
18-
19-            if (steps == 0) {
20-                jumps++;
21-                if (i >= maxReach)
22-                    return -1;
23-                steps = maxReach - i;
24-            }
25-        }
26-        return -1;
27-    }
28-}
```

Custom Input Compile & Run Submit