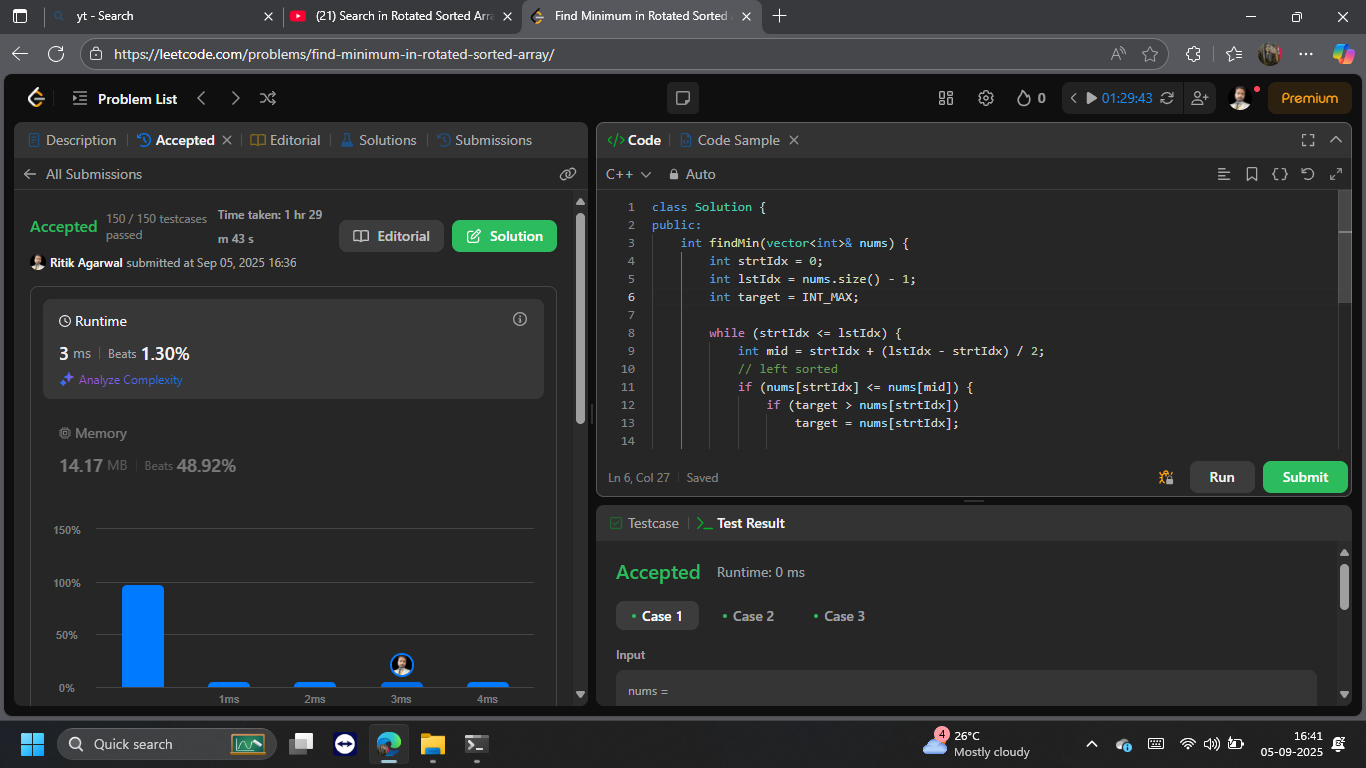
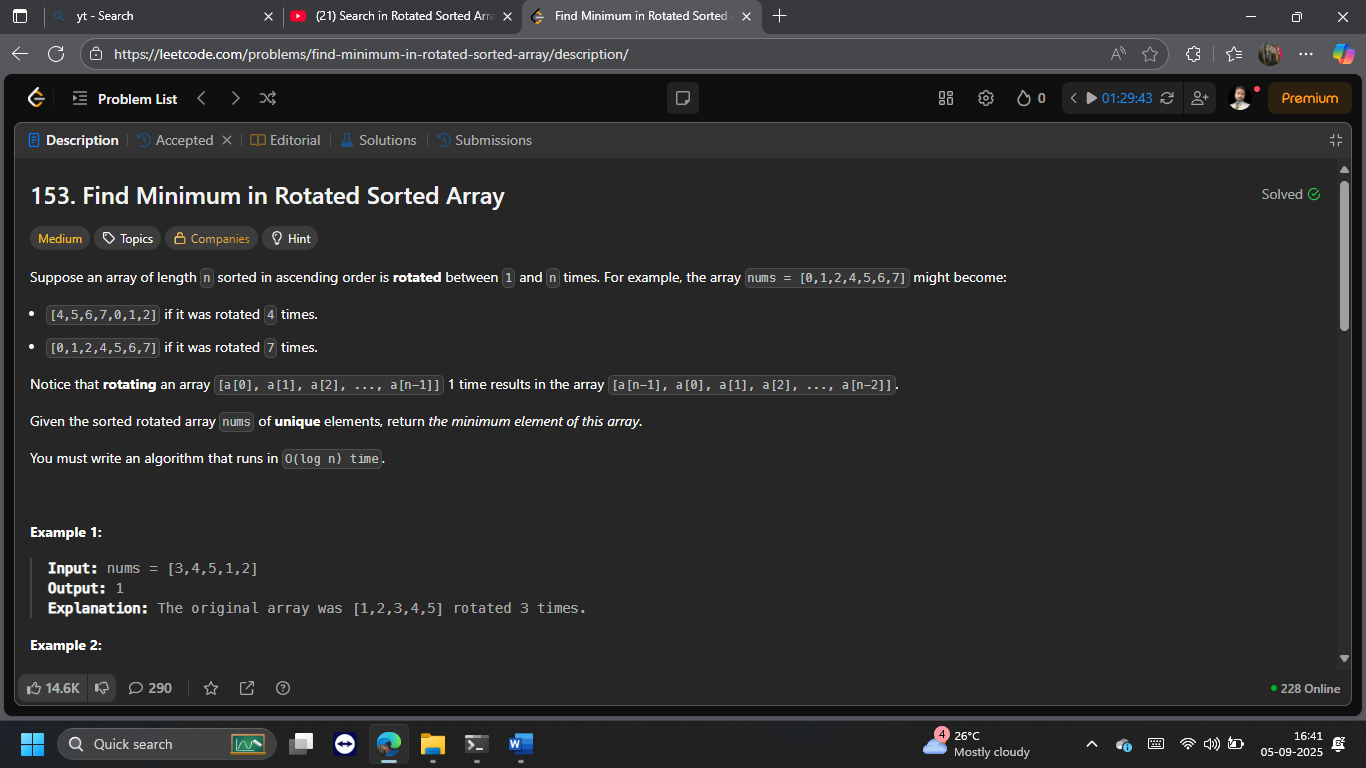
**Leet Code 153 : Find Maximum in a Rotated Sorted Array**

[ Time taken: 1 hr 29 m 43 s ]

Here’s a structured write-up you can use in your **MS Word document** to showcase your solution to the LeetCode problem **“Search in Rotated Sorted Array”**. This includes the problem statement, your C++ implementation, key learnings, and documentation notes — all formatted for clarity and professionalism.

**🧩 Problem Statement**

**Title**: Search in Rotated Sorted Array  
**Difficulty**: Medium  
**Source**: [LeetCode Problem 33](https://leetcode.com/problems/search-in-rotated-sorted-array/)

You are given an integer array nums sorted in ascending order (with distinct values), which may have been rotated at an unknown pivot index k such that the resulting array is:

[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]

Given this array and an integer target, return the index of target if it is in nums, or -1 if it is not.

**Constraints**:

* 1 <= nums.length <= 5000
* -10⁴ <= nums[i], target <= 10⁴
* All values in nums are unique
* Must achieve **O(log n)** time complexity

**💻 C++ Implementation**

class Solution {

public:

int search(vector<int>& nums, int target) {

// Binary search concept can only be applied if array is sorted.

// In a rotated array, either the left part or the right part is sorted.

int strtIdx = 0;

int lstIdx = nums.size() - 1;

while (strtIdx <= lstIdx) {

int mid = strtIdx + (lstIdx - strtIdx) / 2;

if (nums[mid] == target) {

return mid;

}

if (nums[strtIdx] <= nums[mid]) { // Left half is sorted

if (nums[strtIdx] <= target && target <= nums[mid]) {

lstIdx = mid - 1;

} else {

strtIdx = mid + 1;

}

} else { // Right half is sorted

if (nums[mid] <= target && target <= nums[lstIdx]) {

strtIdx = mid + 1;

} else {

lstIdx = mid - 1;

}

}

}

return -1;

}

};

**🧠 Key Learnings**

**✅ Binary Search Adaptation**

* In a rotated sorted array, **at least one half is always sorted**.
* Use binary search to identify the sorted half and decide which side to search next.

**⚠️ Edge Case Handling**

* The condition nums[strtIdx] <= nums[mid] ensures correct detection of the sorted half.
* This logic handles both rotated and non-rotated arrays without special cases.

**🛠️ Debugging Insight**

* A failed test case like nums = [3, 1], target = 1 revealed the importance of inclusive boundary checks.
* Fixing the condition from < to <= resolved the issue and passed all test cases.

**📘 Documentation Notes**

* This solution is part of my DSA repository for IIIT Bhopal.
* Includes README with problem description, solution walkthrough, and edge case analysis.
* Demonstrates how small logical tweaks can make or break binary search in rotated arrays.
* Emphasizes clarity, correctness, and structured learning — core principles of my coding practice.