|  |  |
| --- | --- |
| **Find Rotation Count in C++** | |
| #include <iostream>  #include <vector>  using namespace std;  int findRotationCount(vector<int>& arr) {  int lo = 0;  int hi = arr.size() - 1;  // If the array is not rotated, return 0  if (arr[lo] <= arr[hi]) {  return 0;  }  while (lo <= hi) {  int mid = lo + (hi - lo) / 2;  // Check if mid is the pivot element  if (mid < hi && arr[mid] > arr[mid + 1]) {  return mid + 1;  }  // Check if mid-1 is the pivot element  else if (mid > lo && arr[mid] < arr[mid - 1]) {  return mid;  }  // If arr[lo] <= arr[mid], it means the left half is sorted, so pivot is in the right half  else if (arr[lo] <= arr[mid]) {  lo = mid + 1;  }  // Otherwise, pivot is in the left half  else {  hi = mid - 1;  }  }  return 0; // Should not reach here in a rotated sorted array scenario  }  int main() {  // Hardcoded input  vector<int> arr = {4, 5, 6, 7, 8, 0, 1, 2};  // Call the findRotationCount function to find the rotation count  int ans = findRotationCount(arr);  // Print the rotation count  cout << ans << endl;  return 0;  } | Input: vector<int> arr = {4, 5, 6, 7, 8, 0, 1, 2};  This is a sorted array rotated **5 times**. Let's trace it step-by-step.  **Initial Setup:**   * lo = 0, hi = 7 * Condition: If arr[lo] <= arr[hi], return 0 — not true here (4 > 2)   **🔍 Detailed Step-by-Step Table:**   | **Step** | **lo** | **hi** | **mid** | **arr[mid]** | **arr[mid+1]** | **arr[mid-1]** | **Condition Met** | **Explanation & Action** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 0 | 7 | (0+7)/2 = 3 | 7 | 8 | 6 | arr[lo] <= arr[mid] → 4 <= 7 | Left half is sorted → move right: lo = mid + 1 = 4 | | 2 | 4 | 7 | (4+7)/2 = 5 | 0 | 1 | 8 | arr[mid] < arr[mid-1] → 0 < 8 ✅ | Pivot found → return mid = 5 |  ✅ Final Output: 5 |
| 5 | |