Majority Element - GeeksforGeeks

Source: https://www.geeksforgeeks.org/majority-element/

Majority Element Last Updated : 22 Jul, 2025 Given an array arr[] of size n, find the element that appears more than ■n/2■ times. If no such element exists, return -1. Examples: Input: arr[] = [1, 1, 2, 1, 3, 5, 1] Output: 1 Explanation: Element 1 appears 4 times. Since ■7/2■ = 3, and 4 > 3, it is the majority element. Input: arr[] = [7] Output: 7 Explanation: Element 7 appears once. Since ■1/2■ = 0, and 1 > 0, it is the majority element. Input: arr[] = [2, 13] Output: -1 Explanation: No element appears more than ■2/2■ = 1 time, so there is no majority element. Try it on GfG Practice Table of Content [Naive Approach] Using Two Nested Loops - O(n^2) Time and O(1) Space [Better Approach - 1] Using Sorting - O(nlog(n)) Time and O(1) Space [Better Approach - 2] Using Hashing - O(n) Time and O(n) Space [Expected Approach] Using Moore's Voting Algorithm - O(n) Time and O(1) Space [Naive Approach] Using Two Nested Loops - O(n^2) Time and O(1) Space The idea is to use nested loops to count frequencies. The outer loop selects each element as a candidate, and the inner loop counts how many times it appears. If any element appears more than n / 2 times, it is the majority element. C++ #include <iostream> #include <vector> using namespace std ; int majorityElement (vector < int >& arr) { int n = arr . size (); // Loop to consider each element as a // candidate for majority for (int i = 0 ; i < n ; i ++) { int count = 0 ; // Inner loop to count the frequency of arr[i] for (int j = 0 ; j < n ; j ++) { if (arr [i] == arr [j]) { count ++ ; } } // Check if count of arr[i] is more // than half the size of the array if (count > n / 2) { return arr [i]; } } // If no majority element found, return -1 return -1 ; } int main () { vector < int > arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; cout << majorityElement (arr) << endl ; return 0 ; } C #include <stdio.h> int majorityElement (int arr [], int n) { // Loop to consider each element as // a candidate for majority for (int i = 0 ; i < n ; i ++) { int count = 0 ; // Inner loop to count the frequency of arr[i] for (int j = 0 ; j < n ; j ++) { if (arr [i] == arr [j]) { count ++ ; } } // Check if count of arr[i] is more // than half the size of the array if (count > n / 2) { return arr [i]; } } // If no majority element found, return -1 return -1 ; } int main () { int arr [] = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , majorityElement (arr , n)); return 0 ; } Java class GfG { static int majorityElement (int [] arr) { int n = arr . length ; // Loop to consider each element as // a candidate for majority for (int i = 0 ; i < n ; i ++) { int count = 0 ; // Inner loop to count the frequency of arr[i] for (int j = 0 ; j < n ; j ++) { if (arr [i] == arr [j]) { count ++ ; } } // Check if count of arr[i] is more // than half the size of the array if (count > n / 2) { return arr [i] ; } } // If no majority element found, return -1 return - 1 ; } public static void main (String [] args) { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; System . out . println (majorityElement (arr)); } } Python def majorityElement (arr): n = len (arr) # Loop to consider each element as # a candidate for majority for i in range (n): count = 0 # Inner loop to count the frequency of arr[i] for j in range (n): if arr [i] == arr [j]: count += 1 # Check if count of arr[i] is more # than half the size of the array if count > n // 2 : return arr [i] # If no majority element found, return -1 return - 1 if __name__ == "__main__" : arr = [1 , 1 , 2 , 1 , 3 , 5 , 1] print (majorityElement (arr)) C# using System ; class GfG { static int majorityElement (int [] arr) { int n = arr . Length ; // Loop to consider each element // as a candidate for majority for (int i = 0 ; i < n ; i ++) { int count = 0 ; // Inner loop to count the frequency of arr[i] for (int j = 0 ; j < n ; j ++) { if (arr [i] == arr [j]) { count ++ ; } } // Check if count of arr[i] is more // than half the size of the array if (count > n / 2) { return arr [i]; } } // If no majority element found, return -1 return - 1 ; } static void Main (string [] args) { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; Console . WriteLine (majorityElement (arr)); } } Javascript function majorityElement (arr) { let n = arr . length ; // Loop to consider each element as // a candidate for majority for (let i = 0 ; i < n ; i ++) { let count = 0 ; // Inner loop to count the frequency of arr[i] for (let j =

0 ; j < n ; j ++) { if (arr [i] === arr [j]) { count ++ ; } } // Check if count of arr[i] is more than // half the size of the array if (count > n / 2) { return arr [i]; } } // If no majority element found, return -1 return - 1 ; } // Driver Code let arr = [1 , 1 , 2 , 1 , 3 , 5 , 1]; console . log (majorityElement (arr)); Output 1 [Better Approach - 1] Using Sorting - O(nlog(n)) Time and O(1) Space The idea is to sort the array so that similar elements are next to each other. Once sorted, go through the array and keep track of how many times each element appears. When you encounter a new element, check if the count of the previous element was more than half the total number of elements in the array. If it was, that element is the majority and should be returned. If no element meets this requirement, no majority element exists. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; int majorityElement (vector < int >& arr) { int n = arr . size (); sort (arr . begin (), arr . end ()); // Potential majority element int candidate = arr [n / 2]; // Count how many times candidate appears int count = 0 ; for (int num : arr) { if (num == candidate) { count ++ ; } } if (count > n / 2) { return candidate ; } // No majority element return -1 ; } int main () { vector < int > arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; cout << majorityElement (arr); return 0 ; } Java import java.util.Arrays ; class GfG { static int majorityElement (int [] arr) { int n = arr . length ; Arrays . sort (arr); // Potential majority element int candidate = arr [n / 2] ; // Count how many times candidate appears int count = 0 ; for (int num : arr) { if (num == candidate) { count ++ ; } } if (count > n / 2) { return candidate ; } // No majority element return - 1 ; } public static void main (String [] args) { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; System . out . println (majorityElement (arr)); } } Python def majorityElement (arr): n = len (arr) arr . sort () # Potential majority element candidate = arr [n // 2] # Count how many times candidate appears count = 0 for num in arr : if num == candidate : count += 1 if count > n // 2 : return candidate else : return - 1 if __name__ == "__main__" : arr = [1 , 1 , 2 , 1 , 3 , 5 , 1] print (majorityElement (arr)) C# using System ; class GfG { static int majorityElement (int [] arr) { int n = arr . Length ; Array . Sort (arr); // Potential majority element int candidate = arr [n / 2]; // Count how many times candidate appears int count = 0 ; foreach (int num in arr) { if (num == candidate) { count ++ ; } } if (count > n / 2) { return candidate ; } else { return - 1 ; } } static void Main () { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; Console . WriteLine (majorityElement (arr)); } } Javascript function majorityElement (arr) { let n = arr . length ; arr . sort ((a , b) => a - b); // Potential majority element let candidate = arr [Math . floor (n / 2)]; // Count how many times candidate appears let count = 0 ; for (let num of arr) { if (num === candidate) { count ++ ; } } if (count > Math . floor (n / 2)) { return candidate ; } else { return - 1 ; } } // Driver Code let arr = [1 , 1 , 2 , 1 , 3 , 5 , 1]; console . log (majorityElement (arr)); Output 1 [Better Approach - 2] Using Hashing - O(n) Time and O(n) Space The idea is to use a hash map to track frequencies and identify the majority element in a single pass. Step By Step Implementations: Initialize an empty hash map. Traverse the array and update the count of each element. After each update, check if the count exceeds n / 2. If found, return that element immediately. If no such element exists after the loop, return -1. C++ #include <iostream> #include <unordered_map> #include <vector> using namespace std ; int majorityElement (vector < int > & arr) { int n = arr . size (); unordered_map < int , int > countMap ; // Traverse the array and count occurrences using the hash map for (int num : arr) { countMap [num] ++ ; // Check if current element count exceeds n / 2 if (countMap [num] > n / 2) { return num ; } } // If no majority element is found, return -1 return -1 ; } int main () { vector < int > arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; cout << majorityElement (arr) << endl ; return 0 ; } Java import java.util.HashMap ; import java.util.Map ; class GfG { static int majorityElement (int [] arr) { int n = arr . length ; Map < Integer , Integer > countMap = new HashMap <> (); // Traverse the array and count occurrences using the hash map for (int num : arr) { countMap . put (num , countMap . getOrDefault (num , 0) + 1); // Check if current element count exceeds n / 2 if (countMap . get (num) > n / 2) { return num ; } } // If no majority element is found, return -1 return - 1 ; } public static void main (String [] args) { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; System . out . println (majorityElement (arr)); } } Python from collections import defaultdict def majorityElement (arr): n = len (arr) countMap = defaultdict (int) # Traverse the list and count occurrences using the hash map for num in arr : countMap [num] += 1 # Check if current element count exceeds n / 2 if countMap [num] > n / 2 : return num # If no majority element is found, return -1 return - 1 if __name__ == "__main__" : arr = [1 , 1 , 2 , 1 , 3 , 5 , 1] print (majorityElement (arr)) C# using System ; using System.Collections.Generic ; class GfG { static int majorityElement (int [] arr) { int n = arr . Length ; Dictionary < int , int > countMap = new Dictionary < int , int > (); // Traverse the array and count occurrences using the hash map foreach (int num in arr) { if (countMap . ContainsKey (num)) countMap [num] ++ ; else countMap [num] = 1 ; // Check if current element count exceeds n / 2 if (countMap [num] > n / 2) { return num ; } } // If no majority element is found, return -1 return - 1 ; } public static void Main () { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; Console . WriteLine (majorityElement (arr

)); } } Javascript function majorityElement (arr) { const n = arr . length ; const countMap = new Map (); // Traverse the array and count occurrences using the hash map for (const num of arr) { countMap . set (num , (countMap . get (num) || 0) + 1); // Check if current element count exceeds n / 2 if (countMap . get (num) > n / 2) { return num ; } } // If no majority element is found, return -1 return - 1 ; } // Driver Code const arr = [1 , 1 , 2 , 1 , 3 , 5 , 1]; console . log (majorityElement (arr)); Output 1 [Expected Approach] Using Moore's Voting Algorithm - O(n) Time and O(1) Space The idea is to use the Boyer-Moore Voting Algorithm to efficiently find a potential majority element by canceling out different elements. If a majority element exists, it will remain as the candidate. Then verify it. This is a two-step process: The first step gives the element that may be the majority element in the array. If there is a majority element in an array, then this step will definitely return majority element, otherwise, it will return candidate for majority element. Check if the element obtained from the above step is the majority element. This step is necessary as there might be no majority element. Step By Step Approach: Initialize a candidate variable and a count variable. Traverse the array once: -> If count is zero , set the candidate to the current element and set count to one. -> If the current element equals the candidate, increment count. -> If the current element differs from the candidate, decrement count. Traverse the array again to count the occurrences of the candidate. If the candidate's count is greater than n / 2 , return the candidate as the majority element. C++ #include <iostream> #include <vector> using namespace std ; int majorityElement (vector < int >& arr) { int n = arr . size (); int candidate = -1 ; int count = 0 ; // Find a candidate for (int num : arr) { if (count == 0) { candidate = num ; count = 1 ; } else if (num == candidate) { count ++ ; } else { count -- ; } } // Validate the candidate count = 0 ; for (int num : arr) { if (num == candidate) { count ++ ; } } // If count is greater than n / 2, return the // candidate; otherwise, return -1 if (count > n / 2) { return candidate ; } else { return -1 ; } } int main () { vector < int > arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; cout << majorityElement (arr) << endl ; return 0 ; } C #include <stdio.h> int majorityElement (int arr [], int n) { int candidate = -1 ; int count = 0 ; // Find a candidate for (int i = 0 ; i < n ; i ++) { if (count == 0) { candidate = arr [i]; count = 1 ; } else if (arr [i] == candidate) { count ++ ; } else { count -- ; } } // Validate the candidate count = 0 ; for (int i = 0 ; i < n ; i ++) { if (arr [i] == candidate) { count ++ ; } } // If count is greater than n / 2, return // the candidate; otherwise, return -1 if (count > n / 2) { return candidate ; } else { return -1 ; } } int main () { int arr [] = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%d \n " , majorityElement (arr , n)); return 0 ; } Java class GfG { static int majorityElement (int [] arr) { int n = arr . length ; int candidate = - 1 ; int count = 0 ; // Find a candidate for (int num : arr) { if (count == 0) { candidate = num ; count = 1 ; } else if (num == candidate) { count ++ ; } else { count -- ; } } // Validate the candidate count = 0 ; for (int num : arr) { if (num == candidate) { count ++ ; } } // If count is greater than n / 2, return // the candidate; otherwise, return -1 if (count > n / 2) { return candidate ; } else { return - 1 ; } } public static void main (String [] args) { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; System . out . println (majorityElement (arr)); } } Python def majorityElement (arr): n = len (arr) candidate = - 1 count = 0 # Find a candidate for num in arr : if count == 0 : candidate = num count = 1 elif num == candidate : count += 1 else : count -= 1 # Validate the candidate count = 0 for num in arr : if num == candidate : count += 1 # If count is greater than n / 2, return # the candidate; otherwise, return -1 if count > n // 2 : return candidate else : return - 1 if __name__ == "__main__" : arr = [1 , 1 , 2 , 1 , 3 , 5 , 1] print (majorityElement (arr)) C# using System ; class GfG { static int majorityElement (int [] arr) { int n = arr . Length ; int candidate = - 1 ; int count = 0 ; // Find a candidate foreach (int num in arr) { if (count == 0) { candidate = num ; count = 1 ; } else if (num == candidate) { count ++ ; } else { count -- ; } } // Validate the candidate count = 0 ; foreach (int num in arr) { if (num == candidate) { count ++ ; } } // If count is greater than n / 2, return // the candidate; otherwise, return -1 if (count > n / 2) { return candidate ; } else { return - 1 ; } } static void Main () { int [] arr = { 1 , 1 , 2 , 1 , 3 , 5 , 1 }; Console . WriteLine (majorityElement (arr)); } } Javascript function majorityElement (arr) { const n = arr . length ; let candidate = - 1 ; let count = 0 ; // Find a candidate for (const num of arr) { if (count === 0) { candidate = num ; count = 1 ; } else if (num === candidate) { count ++ ; } else { count -- ; } } // Validate the candidate count = 0 ; for (const num of arr) { if (num === candidate) { count ++ ; } } // If count is greater than n / 2, return // the candidate; otherwise, return -1 if (count > n / 2) { return candidate ; } else { return - 1 ; } } // Driver Code const arr = [1 , 1 , 2 , 1 , 3 , 5 , 1]; console . log (majorityElement (arr)); Output 1 Comment Article Tags: Article Tags: DSA Arrays Majority Element Moore's Voting Algorithm Microsoft Amazon Samsung D-E-Shaw Accolite MakeMyTrip FactSet + 7 More