# Find Unique CodeStudio

You are given an array of integers, where every element appears exactly twice except for one element. You need to find and return the unique element.

Input: [2, 4, 6, 8, 2, 4, 6]

Output: 8

Input: [1, 3, 5, 7, 9, 1, 3, 5]

Output: 7

Note:

The given array will always have exactly one element that appears only once, while all other elements appear exactly twice.

**Approach 1: Using a tested loop to compare all the array elements with each other.**

The approach uses a nested loop to compare each element in the array with all the other elements. If a duplicate is found, it breaks out of the inner loop. If no duplicate is found, the element is considered unique and returned.

This approach has a **time complexity of O(n^2)** since it requires nested iterations through the array.

This solution does not require any additional space besides the input array. It uses a constant amount of space, resulting in a **space complexity of O(1).**

**Approach 2: Using a XOR operation to find the Unique element.**

The approach utilizes the XOR (^) operation to find the unique element. It performs an XOR operation on all the elements in the array. As XOR of a number with itself is 0, and XOR is associative, the final result will be the unique element.

This approach has a **time complexity of O(n)** as it requires a single iteration through the array.

This solution also does not require any additional space besides the input array. It performs the bitwise XOR operation on the elements in-place and does not use any data structures or extra memory. Hence, it has a **space complexity of O(1).**

**Approach 3: Using an unordered map to store the frequency of each element in the array.**

The approach uses an unordered map to store the frequency of each element in the array. It then iterates over the map and returns the element with a frequency of 1.

This approach has a **time complexity of O(n)** as it requires a single iteration through the array.

it also requires additional space to store the map, resulting in a **space complexity of O(n).**

**Approach 2 using XOR operator solution is the most optimal one.** It has a linear time complexity of O(n) and requires minimal additional space.