```
class Program
  {
    /*
     * Single Number
     * Given a non-empty array of integers, every element appears twice except for one. Find that single one.
     *
     * Example 1:
     * Input: [2,2,1]
     * Output: 1
     *
     * Example 2:
     * Input: [4,1,2,1,2]
     * Output: 4
     */
    static void Main(string[] args)
    {
      int[] inputArr = { 2, 2, 1 };

      Console.WriteLine("What is the answer to the given question?");
      Console.WriteLine($"{SingleNumber(inputArr)}");
      Console.WriteLine($"{SingleNumber2(inputArr)}");

      int[] inputArr_2 = { 4, 1, 2, 1, 2 };

      Console.WriteLine("What is the answer to the given question?");
      Console.WriteLine($"{SingleNumber(inputArr_2)}");
      Console.WriteLine($"{SingleNumber2(inputArr_2)}");
    }

    /*
     * Approach 1 - Using List Operation
     * 1. Iterate over all the elements in nums
     * 2. If some number in nums is new to array, add it
     * 3. If some number is already in the array, remove it
     */
    public static int SingleNumber(int[] nums)
    {
      List<int> result = new List<int>();    // Space Complexity = n

      for(int i=0; i<nums.Length; i++)
      {
        if (!result.Contains(nums[i]))      // Time Complexity = n
        {
          result.Add(nums[i]);
        }
        else                                // Time Complexity = n
        {
          result.Remove(nums[i]);
        }
      }
      return result[0];                     // Time Complexity = n*n = O(n^2), Space Complexity = O(n)
    }
    /*
     * Approach 2 - Using HashSet
     * 1. Iterate through all elements in nums and set up key/value pair.
     * 2. Return the element which appeared only once.
     */
     public static int SingleNumber2(int[] nums)
    {
        HashSet<int> map = new HashSet<int>();

        for(int i=0; i<nums.Length; i++)   // Time Complexity = n
        {
            if (map.Contains(nums[i]))      // Time Complexity of HashSet = 1
            {
                map.Remove(nums[i]);        // 2. In map, the duplicated elements will be deleted.
                                            //      As a result, map = {4};
            }
            else
            {
                map.Add(nums[i]);           // 1. map = {4, 1, 2}
            }
        }
        return map.First();                 // Time Complexity = n*1 = O(n), Space Complexity = O(n)
    }
}
```