```
class Program
  {
    /*
     * Two Sum
     *
     * Given an array of integers, return indices of the two numbers such that they add up to a specific target.
     * You may assume that each input would have exactly one solution, and you may not use the same element twice.
     *
     * Example:
     * Given nums = [2, 7, 11, 15], target = 9,
     * Because nums[0] + nums[1] = 2 + 7 = 9,
     * return [0, 1].
     */
    static void Main(string[] args)
    {
      int[] nums = { 1, 7, 8, 15 };
      int target = 9;

      Console.WriteLine("The answer to the given question:");
      Console.WriteLine($"[{TwoSum(nums, target)[0]}, {TwoSum(nums, target)[1]}]\n");

      int[] nums2 = { 1, 3, 5, 7, 8, 11 };

      Console.WriteLine("The answer to the given question:");
      Console.WriteLine($"[{TwoSum2(nums2, target)[0]}, {TwoSum2(nums2, target)[1]}]\n");
    }


    /*
     * Approach 1 - Brute Force
     * Loop through each element x and y in the given array
     * Find if there is the sum of x and y that equals to the target. (x + y = target)
     */
    public static int[] TwoSum(int[] nums, int target)
    {
      int[] result = new int[2];

      for(int i=0; i < nums.Length; i++)     // Time complexity = n
      {
        for(int j=i+1; j<nums.Length; j++)  // Time complexity = n^2
        {
          if(nums[i] + nums[j] == target)
          {
            result[0] = i;
            result[1] = j;
          }
        }
      }
      return result;  // Time complexity = O(n^2), Space complexity = O(1)
    }
```

```
    /*
     * Approach 2 - Using Dictionary
     * Using Hash Table is the best way to maintain a mapping of each element in the array to its index.
     * See Details: https://www.tutorialsteacher.com/csharp/csharp-hashtable
     */
    public static int[] TwoSum2(int[] nums, int target)
    {
        var dict = new Dictionary<int, int>();

        for(int i=0; i < nums.Length; i++)  // Time complexity = n, Space complexity = n
        {                                   // The extra space required depends on the n of items stored in the dictionary.
            dict.Add(nums[i], i);      // {key, value}: {1, 0}, {7, 1}, {8, 2}, {15, 3}
        }

        for(int i=0; i < nums.Length; i++)  // Time complexity = n
        {
            int complement = target - nums[i];  // i=0, target=9, nums[0]=1. Therefore, complement= 9-1 =8;
            if (dict.ContainsKey(complement))   // {8, 2};
            {
                return new int[] { i, dict[complement] };   // since dict[8] = 2, it returns {0, 2}.
            }
        }
        return null;    // As a result, Time Complexity = O(n), and Space Complexity = O(n)
    }
}
```