Here’s an explanation and implementation of the solution, focusing on using a set for O(n) efficiency.

### Explanation

The \*\*primary insight\*\* is that a set automatically enforces uniqueness. This means:

1. Adding an item to a set will only succeed if that item isn’t already present.

2. Checking if an item exists in the set is an O(1) operation.

### Steps to Optimize Using a Set:

1. Create a `HashSet<char>` to keep track of letters encountered so far.

2. Loop through the input string. For each character:

- If it already exists in the set, return `false` immediately.

- Otherwise, add it to the set.

3. If the loop completes without finding duplicates, return `true`.

---

### Optimized Implementation

```csharp

public static class UniqueLettersSolution

{

public static void Run()

{

var test1 = "abcdefghjiklmnopqrstuvwxyz"; // Expect True because all letters unique

Console.WriteLine(AreUniqueLetters(test1));

var test2 = "abcdefghjiklanopqrstuvwxyz"; // Expect False because 'a' is repeated

Console.WriteLine(AreUniqueLetters(test2));

var test3 = ""; // Expect True because it's an empty string

Console.WriteLine(AreUniqueLetters(test3));

}

/// <summary>

/// Determine if all letters in the text are unique.

/// </summary>

/// <param name="text">Text to check for unique letters.</param>

/// <returns>true if all letters are unique, otherwise false.</returns>

private static bool AreUniqueLetters(string text)

{

var found = new HashSet<char>();

foreach (var letter in text)

{

// Check if the letter is already in the set

if (found.Contains(letter))

return false;

// Add the letter to the set

found.Add(letter);

}

return true; // All letters are unique

}

}

```

---

### Alternative Implementation (Size Comparison Method)

This approach uses the unique property of a set to compare sizes:

1. Create a `HashSet<char>` from the input string.

2. Compare the size of the set to the length of the string.

- If the sizes are equal, the string has unique letters.

- If not, it contains duplicates.

```csharp

private static bool AreUniqueLettersAlternate(string text)

{

var unique = new HashSet<char>(text);

return unique.Count == text.Length;

}

```

---

### Key Points:

- \*\*Performance\*\*: Both implementations run in O(n) time due to the properties of sets.

- \*\*Behavior of `HashSet`\*\*: Adding and checking for existence in a `HashSet` are O(1) operations, making it ideal for this problem.

- \*\*Comparison\*\*: The first method short-circuits as soon as a duplicate is found, potentially saving time. The second method completes processing before checking sizes.