**1.Understand Asymptotic Notation**

Big O notation is used to describe how an algorithm running time or space requirements grow as the input size increases. It helps us compare algorithms and understand their efficiency.  
  
For search algorithms:  
- Linear Search: O(n) in the worst case.  
- Binary Search: O(log n) in the worst case, but requires a sorted array.

**2.Setup**

We define a Product class with properties that we want to search by, such as productId, productName, and category. We will use an array of Product objects for both linear and binary search.

**3.Implementation**

using System;  
using System.Linq;  
  
public class Product  
{  
 public int ProductId { get; set; }  
 public string ProductName { get; set; }  
 public string Category { get; set; }  
  
 public Product(int id, string name, string category)  
 {  
 ProductId = id;  
 ProductName = name;  
 Category = category;  
 }  
  
 public override string ToString()  
 {  
 return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";  
 }  
}  
  
class Program  
{  
 public static Product LinearSearch(Product[] products, string searchName)  
 {  
 foreach (var product in products)  
 {  
 if (product.ProductName.Equals(searchName, StringComparison.OrdinalIgnoreCase))  
 return product;  
 }  
 return null;  
 }  
  
 public static Product BinarySearch(Product[] products, string searchName)  
 {  
 int left = 0;  
 int right = products.Length - 1;  
  
 while (left <= right)  
 {  
 int mid = (left + right) / 2;  
 int comparison = string.Compare(products[mid].ProductName, searchName, StringComparison.OrdinalIgnoreCase);  
  
 if (comparison == 0)  
 return products[mid];  
 else if (comparison < 0)  
 left = mid + 1;  
 else  
 right = mid - 1;  
 }  
 return null;  
 }  
  
 static void Main(string[] args)  
 {  
 Product[] products = new Product[]  
 {  
 new Product(1, "Laptop", "Electronics"),  
 new Product(2, "T-Shirt", "Apparel"),  
 new Product(3, "Keyboard", "Electronics"),  
 new Product(4, "Mug", "Kitchen"),  
 new Product(5, "Mouse", "Electronics"),  
 };  
  
 Console.WriteLine("Enter product name to search:");  
 string searchTerm = Console.ReadLine();  
  
 Console.WriteLine("\n Linear Search Result:");  
 var result1 = LinearSearch(products, searchTerm);  
 Console.WriteLine(result1 != null ? result1.ToString() : "Product not found!");  
  
 var sortedProducts = products.OrderBy(p => p.ProductName).ToArray();  
  
 Console.WriteLine("\n Binary Search Result:");  
 var result2 = BinarySearch(sortedProducts, searchTerm);  
 Console.WriteLine(result2 != null ? result2.ToString() : "Product not found!");  
 }  
}

**4.Analysis**

Time Complexity Comparison:  
- Linear Search: O(n) in the worst case (goes through all items).  
- Binary Search: O(log n) in the worst case (keeps cutting the array in half).

Output:

