**Exercise 2: E-commerce Platform Search Function**

**Understanding Asymptotic Notation**

**Explain Big O notation and how it helps in analyzing algorithms.**

**ANS:**

**Big O Notation: Big O Notation** describes the **upper bound** of an algorithm's running time as the input size increases. It helps in comparing the **efficiency** of algorithms by ignoring constant factors and focusing on how performance scales.

Example:

O(1): Constant Time where Time stays the same regardless of input size.

O(logn): Logarithmic Time where work is reduced with each step (usually via division)

O(n): Linear time where the performance grows linearly with the input size.

**Describe the best, average, and worst-case scenarios for search operations.**

**ANS:**

### **Linear Search**

**Linear Search** is a simple algorithm that sequentially checks each element in the array until it finds the target or reaches the end of the array.

#### **1. Best Case:**

* **Scenario**: The target element is the **first element** in the array.
* **Operations**: Only **one comparison** is needed.
* **Time Complexity**: **O(1)**

#### **2. Average Case:**

* **Scenario**: The target element is **somewhere in the middle** of the array.
* **Operations**: On average, the algorithm checks **n/2 elements**.
* **Time Complexity**: **O(n)**

#### **3. Worst Case:**

* **Scenario**: The target element is the **last element** or **not present** in the array.
* **Operations**: The algorithm checks **all n elements**.
* **Time Complexity**: **O(n)**

### **Binary Search**

**Binary Search** is a fast algorithm for sorted arrays. It repeatedly divides the search range in half to locate the target element.

#### **1. Best Case:**

* **Scenario**: The target element is the **middle element** of the array.
* **Operations**: The algorithm finds the target in the **first comparison**.
* **Time Complexity**: **O(1)**

#### **2. Average Case:**

* **Scenario**: The target element is **randomly located** within the array.
* **Operations**: The algorithm performs around **log₂(n) comparisons**.
* **Time Complexity**: **O(log n)**

#### **3. Worst Case:**

* **Scenario**: The target element is **not present**, or it takes maximum steps to find it.
* **Operations**: The algorithm continues halving until only **one element** remains.
* **Time Complexity**: **O(log n)**

**CODE:**

using System;

public class Program

{

public static void Main(string[] args)

{

Product[] products = new Product[] {

new Product(1, "iPhone 14", "Electronics"),

new Product(2, "Classmate Spiral Notebook", "Stationary"),

new Product(3, "Sony WH-1000XM5 Headphones", "Electronics"),

new Product(4, "Velvet Blackout Curtain", "Home Decor"),

new Product(5, "Boat Rockerz 450", "Electronics"),

new Product(6, "Faber-Castell Sketch Pens", "Stationary"),

new Product(7, "Samsung Galaxy S23", "Electronics"),

new Product(8, "Ikea Study Table", "Furniture"),

new Product(9, "Usha Ceiling Fan", "Home Appliances"),

new Product(10, "Raymond Cotton Shirt", "Clothing"),

new Product(11, "HP Pavilion Laptop", "Electronics"),

new Product(12, "Philips LED Bulb 12W", "Home Appliances"),

new Product(13, "Parker Jotter Pen", "Stationary"),

new Product(14, "Bombay Dyeing Bedspread", "Home Decor"),

new Product(15, "Nike Revolution 6", "Footwear"),

new Product(16, "Wildcraft Backpack 30L", "Accessories"),

new Product(17, "Canon Pixma Printer", "Electronics"),

new Product(18, "Prestige Induction Cooker", "Kitchen Appliances"),

new Product(19, "Adidas Sports Shorts", "Clothing"),

new Product(20, "Philips Hair Trimmer", "Personal Care"),

new Product(21, "Redmi Note 10 Pro", "Electronics"),

new Product(22, "Realme Narzo 50", "Electronics")

};

Console.WriteLine("Welcome to E-Commerce Shopping Platform");

Console.WriteLine("1. Search By Product Name (partial match)");

Console.WriteLine("2. Search for Product ID");

Console.WriteLine("3. Search for Products in a Category (partial match)");

int choice;

if (int.TryParse(Console.ReadLine(), out choice))

{

if (choice == 1)

{

Console.WriteLine("Enter name of the product:");

string input = Console.ReadLine();

SearchByNamePartial(products, input);

}

else if (choice == 2)

{

Console.WriteLine("Enter the ID of the product:");

string idInput = Console.ReadLine();

int id;

if (int.TryParse(idInput, out id))

{

BinarySearchByID(products, id);

}

else

{

Console.WriteLine("Invalid product ID entered.");

}

}

else if (choice == 3)

{

Console.WriteLine("Enter the name of the Category:");

string categoryName = Console.ReadLine();

SearchByCategoryPartial(products, categoryName);

}

else

{

Console.WriteLine("Invalid choice. Please select 1, 2, or 3.");

}

}

else

{

Console.WriteLine("Invalid input. Please enter a valid number.");

}

Console.WriteLine("Press Enter to exit...");

Console.ReadLine();

}

public static void SearchByNamePartial(Product[] products, string namePart)

{

bool found = false;

foreach (var product in products)

{

if (product.productName.IndexOf(namePart, StringComparison.OrdinalIgnoreCase) >= 0)

{

Console.WriteLine($"Product: {product.productName} (ID: {product.productID}, Category: {product.Category}) is Available");

found = true;

}

}

if (!found)

{

Console.WriteLine("No matching products found.");

}

}

public static void SearchByCategoryPartial(Product[] products, string categoryPart)

{

bool found = false;

foreach (var product in products)

{

if (product.Category.IndexOf(categoryPart, StringComparison.OrdinalIgnoreCase) >= 0)

{

Console.WriteLine($"Product: {product.productName} (ID: {product.productID}, Category: {product.Category}) is Available");

found = true;

}

}

if (!found)

{

Console.WriteLine("No products found in this category.");

}

}

public static void BinarySearchByID(Product[] products, int id)

{

Array.Sort(products, (a, b) => a.productID.CompareTo(b.productID));

int left = 0;

int right = products.Length - 1;

while (left <= right)

{

int mid = left + (right - left) / 2;

if (products[mid].productID == id)

{

Console.WriteLine($"Product: {products[mid].productName} (ID: {products[mid].productID}, Category: {products[mid].Category}) is Available");

return;

}

else if (products[mid].productID < id)

{

left = mid + 1;

}

else

{

right = mid - 1;

}

}

Console.WriteLine("Product Not Available");

}

}

public class Product

{

public int productID { get; set; }

public string productName { get; set; }

public string Category { get; set; }

public Product(int productID, string productName, string category)

{

this.productID = productID;

this.productName = productName;

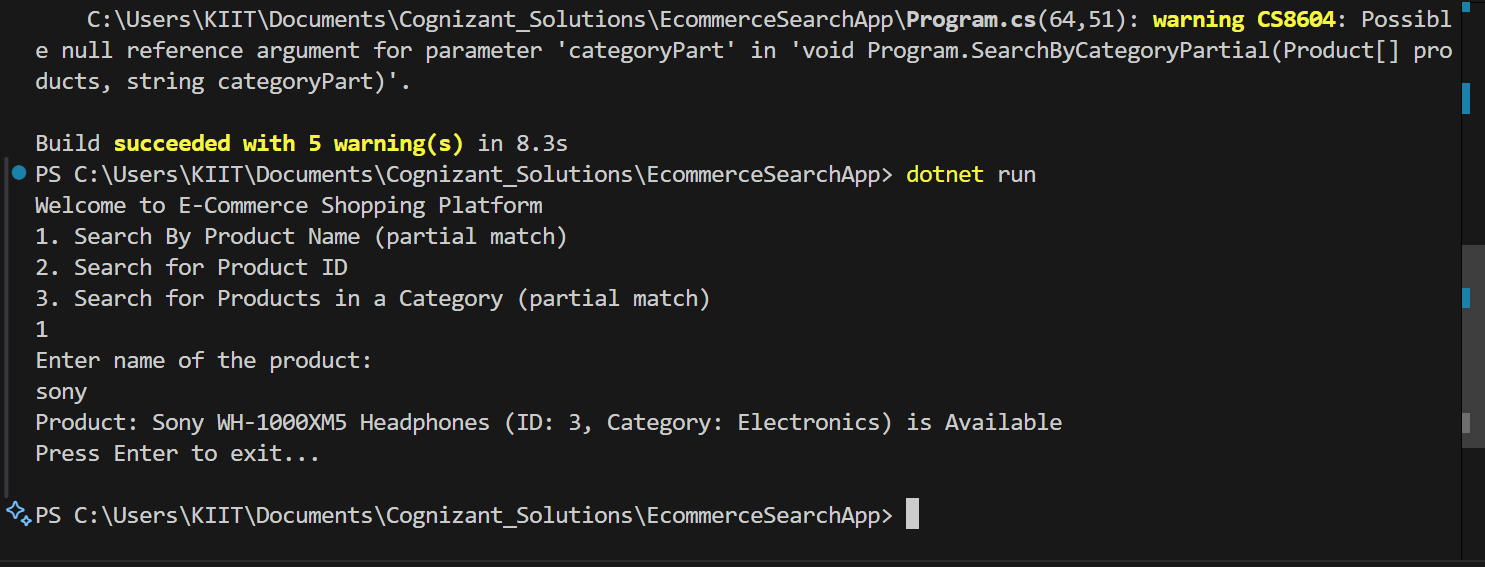
this.Category = category;

}

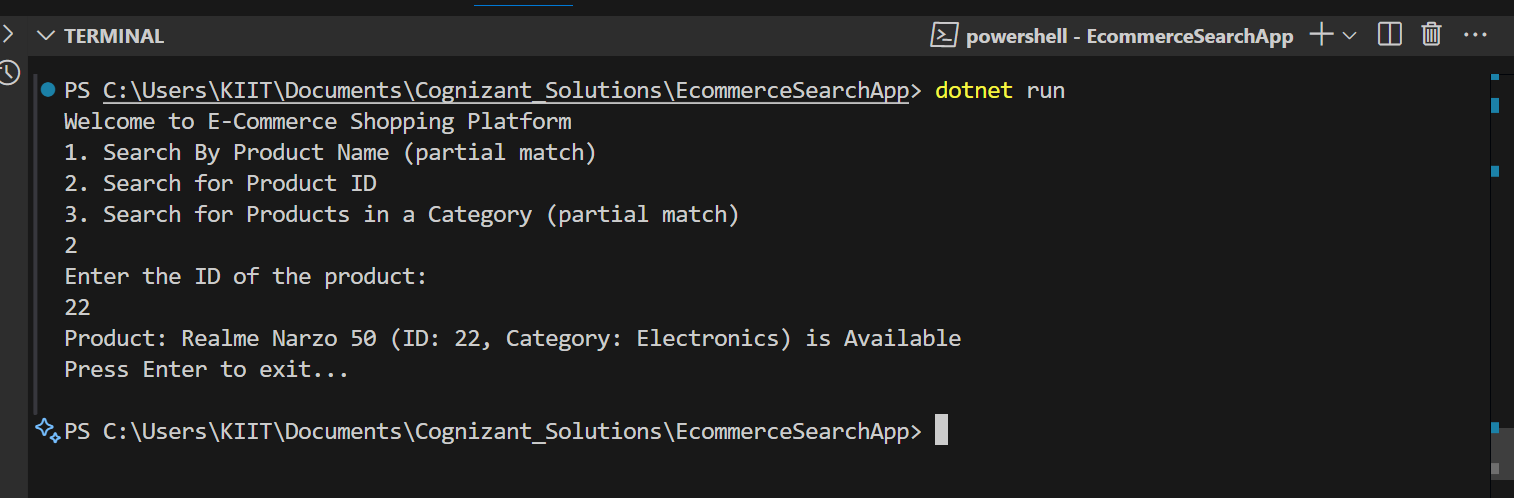
}

**OUTPUT:**

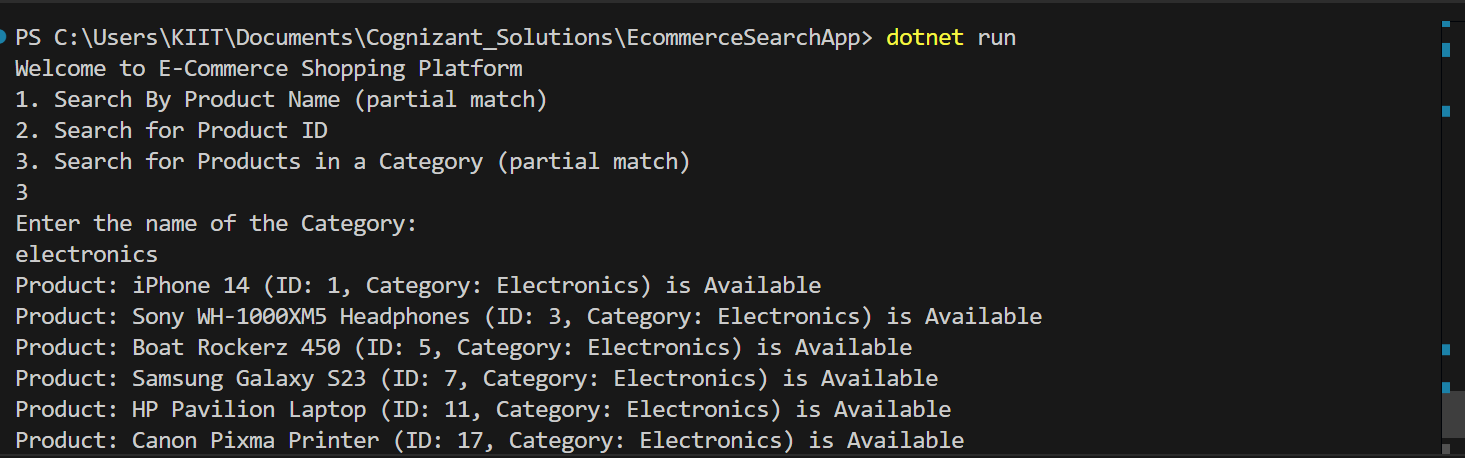
**Output By Using Name:**



**Output by Using ID:**



**Output By Using Product Category**



**ANALYSIS:**

1. **Compare the time complexity of linear and binary search algorithms**

**ANS:**

### **1. Linear Search**

* **How it works**: Checks each element in the array one by one until it finds the target or reaches the end.
* **Applicable to**: **Unsorted** data or list of elements
* **Best Case**:
  + Target is at the beginning.
  + **Time Complexity**: O(1)
* **Average Case**:
  + Target is somewhere in the middle.
  + **Time Complexity**: O(n)
* **Worst Case**:
  + Target is at the end or not present.
  + **Time Complexity**: O(n)

### **2. Binary Search**

* **How it works**: Repeatedly divide the **sorted** array into halves to locate the target.
* **Applicable to**: **Sorted** data only.
* **Best Case**:
  + Target is exactly at the middle index.
  + **Time Complexity**: O(1)
* **Average Case**:
  + Target is somewhere in the array.
  + **Time Complexity**: O(log n)
* **Worst Case**:
  + Target not present or at extreme ends.
  + **Time Complexity**: O(log n)

**2.Discuss which algorithm is more suitable for your platform and why.**

**ANS:**

For my e-commerce platform:

* It is suitable to Use **Linear Search** when searching by:
  + **Product Name** (partial match)
  + **Category** (partial match)
* It is suitable to Use **Binary Search** only when:
  + Searching by **Product ID**, **and** my data is **sorted by ID**

### **Why Linear Search is suitable for names/categories:**

* Users may **type only part** of a product name or category (e.g., "Note", "Electro").
* Binary search needs **exact matching** and **sorted data** — not practical for partial matches.
* So, **linear search is the only flexible and accurate option** for partial string matching.

### **Why Binary Search is suitable for Product ID:**

* Product IDs are **unique integers** and can be sorted easily.
* Binary search is **much faster** (O(log n)) than linear search (O(n)) for large datasets. So, for exact ID searching, binary search is ideal.

**Exercise 7: Financial Forecasting**

**Understanding Recursive Algorithms:**

**Recursion: Recursion** is a technique in programming where a function calls **itself** to solve a smaller version of the same problem.

It’s like solving a big problem by **breaking it into smaller sub-problems**, and then solving each piece step by step.

Every recursive function must have a **base case**, which acts like a stopping point — without it, the function would call itself forever!

**CODE:**  
using System;

namespace FinancialForecasting

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter The Present Sum:");

int presentSum = int.Parse(Console.ReadLine());

Console.WriteLine("Enter the rate in percent (e.g.,2 for 2%):");

double rate = double.Parse(Console.ReadLine()) / 100;

Console.WriteLine("Enter the Time Period (in years):");

int time = int.Parse(Console.ReadLine());

double futureValue = Forecasting(presentSum, rate, time);

Console.WriteLine($"Forecasted Future Value after {time} years: Rs{futureValue:F2}");

}

public static double Forecasting(double presentSum, double rate, int time)

{

if (time == 0)

{

return presentSum;

}

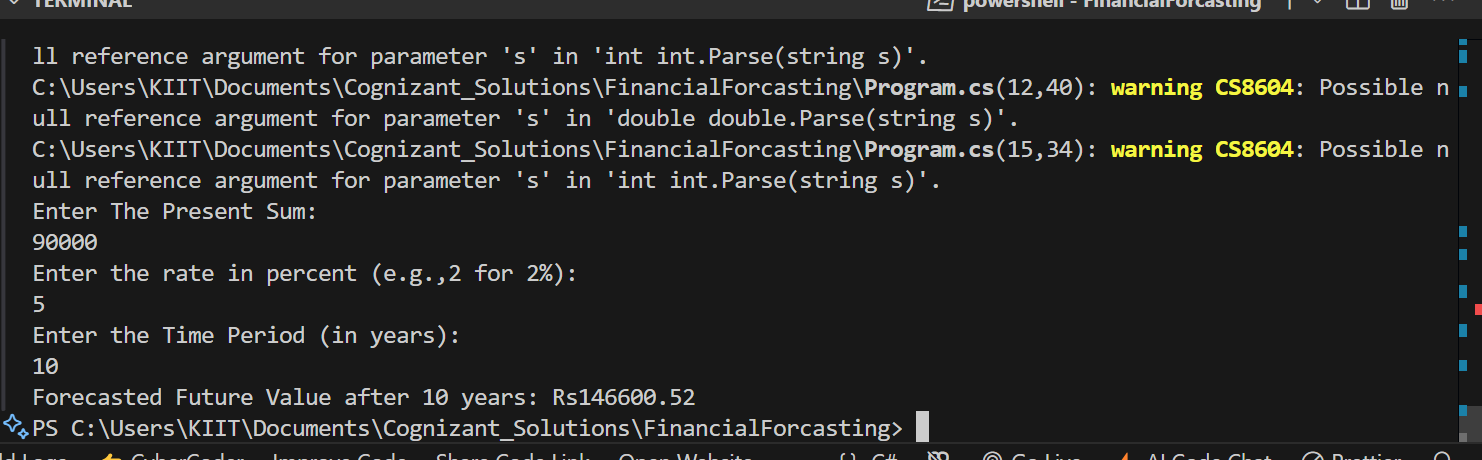
return Forecasting(presentSum \* (1 + rate), rate, time - 1);

}

}

}

**OUTPUT: (Exercise-7)**



|  |
| --- |
|  |

|  |
| --- |
|  |