**Exercise 1: Inventory Management System**

Codes :

Product.cs:

namespace InventoryManagementSystem

{

    public class Product

    {

        public int ProductId { get; set; }

        public string ProductName { get; set; }

        public int Quantity { get; set; }

        public double Price { get; set; }

        public Product(int id, string name, int quantity, double price)

        {

            ProductId = id;

            ProductName = name;

            Quantity = quantity;

            Price = price;

        }

        public override string ToString()

        {

            return $"[ID: {ProductId}, Name: {ProductName}, Quantity: {Quantity}, Price: {Price:C2}]";

        }

    }

}

Inventory.cs:

using System;

using System.Collections.Generic;

namespace InventoryManagementSystem

{

    public class Inventory

    {

        private Dictionary<int, Product> products = new Dictionary<int, Product>();

         public void AddProduct(Product product)

        {

            if (!products.ContainsKey(product.ProductId))

            {

                products.Add(product.ProductId, product);

                Console.WriteLine("✅ Product added.");

            }

            else

            {

                Console.WriteLine("⚠️ Product with this ID already exists.");

            }

        }

        public void UpdateProduct(int productId, int quantity, double price)

        {

            if (products.ContainsKey(productId))

            {

                products[productId].Quantity = quantity;

                products[productId].Price = price;

                Console.WriteLine("🔄 Product updated.");

            }

            else

            {

                Console.WriteLine("❌ Product not found.");

            }

        }

        public void DeleteProduct(int productId)

        {

            if (products.Remove(productId))

            {

                Console.WriteLine("🗑️ Product deleted.");

            }

            else

            {

                Console.WriteLine("❌ Product not found.");

            }

        }

        public void DisplayInventory()

        {

            Console.WriteLine("\n📋 Current Inventory:");

            foreach (var product in products.Values)

            {

                Console.WriteLine(product);

            }

        }

    }

}

Program.cs:

using System;

namespace InventoryManagementSystem

{

    class Program

    {

        static void Main(string[] args)

        {

            Inventory inventory = new Inventory();

            inventory.AddProduct(new Product(1, "Laptop", 10, 75000));

            inventory.AddProduct(new Product(2, "Mouse", 100, 500));

            inventory.AddProduct(new Product(3, "Keyboard", 50, 1500));

             inventory.DisplayInventory();

            inventory.UpdateProduct(2, 80, 450);

             inventory.DeleteProduct(1);

            inventory.DisplayInventory();

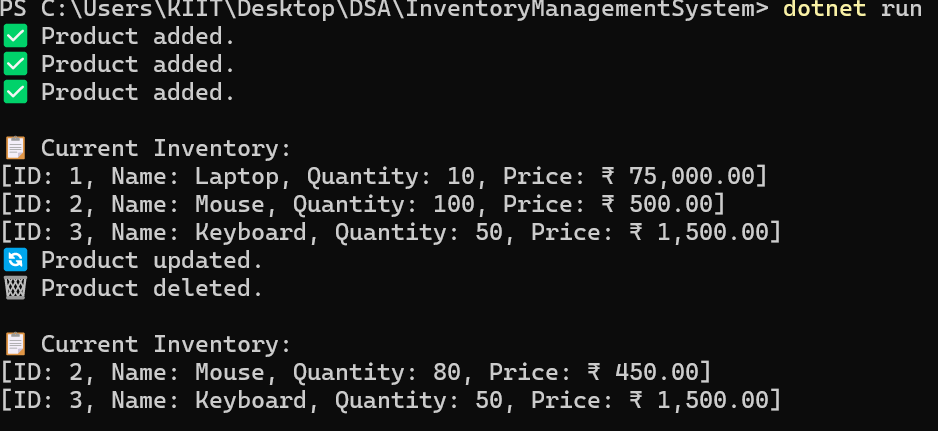
            Console.ReadKey();

        }

    }

}

Output:



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

Codes :

Product.cs:

namespace ECommerceSearchFunction

{

    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}]";

        }

    }

}

SearchHelper.cs:

using System;

namespace ECommerceSearchFunction

{

    public static class SearchHelper

    {

       public static Product LinearSearch(Product[] products, string name)

        {

            foreach (var product in products)

            {

                if (product.ProductName.Equals(name, StringComparison.OrdinalIgnoreCase))

                    return product;

            }

            return null;

        }

        public static Product BinarySearch(Product[] products, string name)

        {

            int left = 0;

            int right = products.Length - 1;

            while (left <= right)

            {

                int mid = (left + right) / 2;

                int comparison = string.Compare(products[mid].ProductName, name, StringComparison.OrdinalIgnoreCase);

                if (comparison == 0)

                    return products[mid];

                else if (comparison < 0)

                    left = mid + 1;

                else

                    right = mid - 1;

            }

            return null;

        }

    }

}

Program.cs:

using System;

namespace ECommerceSearchFunction

{

    class Program

    {

        static void Main(string[] args)

        {

            Product[] products = new Product[]

            {

                new Product(101, "Laptop", "Electronics"),

                new Product(102, "Headphones", "Electronics"),

                new Product(103, "Shoes", "Footwear"),

                new Product(104, "Shirt", "Apparel"),

                new Product(105, "Watch", "Accessories")

            };

             Console.WriteLine("== Linear Search ==");

            Product resultLinear = SearchHelper.LinearSearch(products, "Shoes");

            Console.WriteLine(resultLinear != null ? resultLinear.ToString() : "Product not found");

            Array.Sort(products, (p1, p2) => string.Compare(p1.ProductName, p2.ProductName));

            Console.WriteLine("\n== Binary Search ==");

            Product resultBinary = SearchHelper.BinarySearch(products, "Shoes");

            Console.WriteLine(resultBinary != null ? resultBinary.ToString() : "Product not found");

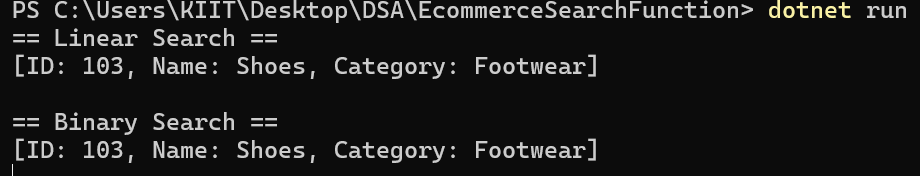
            Console.ReadKey();

        }

    }

}

Output:



**Exercise 3: Sorting Customer Orders**

Codes :

Order.cs:

namespace CustomerOrderSorting

{

    public class Order

    {

        public int OrderId { get; set; }

        public string CustomerName { get; set; }

        public double TotalPrice { get; set; }

        public Order(int orderId, string customerName, double totalPrice)

        {

            OrderId = orderId;

            CustomerName = customerName;

            TotalPrice = totalPrice;

        }

        public override string ToString()

        {

            return $"[Order ID: {OrderId}, Customer: {CustomerName}, Total: ₹{TotalPrice:F2}]";

        }

    }

}

SortHelper.cs:

using System;

namespace CustomerOrderSorting

{

    public static class SortHelper

    {

         public static void BubbleSort(Order[] orders)

        {

            int n = orders.Length;

            for (int i = 0; i < n - 1; i++)

            {

                for (int j = 0; j < n - i - 1; j++)

                {

                    if (orders[j].TotalPrice > orders[j + 1].TotalPrice)

                    {

                        var temp = orders[j];

                        orders[j] = orders[j + 1];

                        orders[j + 1] = temp;

                    }

                }

            }

        }

         public static void QuickSort(Order[] orders)

        {

            QuickSortRecursive(orders, 0, orders.Length - 1);

        }

        private static void QuickSortRecursive(Order[] orders, int low, int high)

        {

            if (low < high)

            {

                int pivotIndex = Partition(orders, low, high);

                QuickSortRecursive(orders, low, pivotIndex - 1);

                QuickSortRecursive(orders, pivotIndex + 1, high);

            }

        }

        private static int Partition(Order[] orders, int low, int high)

        {

            double pivot = orders[high].TotalPrice;

            int i = low - 1;

            for (int j = low; j < high; j++)

            {

                if (orders[j].TotalPrice < pivot)

                {

                    i++;

                    (orders[i], orders[j]) = (orders[j], orders[i]);

                }

            }

            (orders[i + 1], orders[high]) = (orders[high], orders[i + 1]);

            return i + 1;

        }

    }

}

Program.cs:

using System;

namespace CustomerOrderSorting

{

     class Program

    {

        static void Main(string[] args)

        {

            Order[] orders = new Order[]

            {

                new Order(101, "Alice", 5500.75),

                new Order(102, "Bob", 12000.00),

                new Order(103, "Charlie", 4200.50),

                new Order(104, "Daisy", 8500.00),

                new Order(105, "Eve", 3000.00)

            };

            Console.WriteLine("🧾 Original Orders:");

            DisplayOrders(orders);

            Console.WriteLine("\n🔵 Bubble Sort by Total Price:");

            Order[] bubbleSorted = (Order[])orders.Clone();

            SortHelper.BubbleSort(bubbleSorted);

            DisplayOrders(bubbleSorted);

            Console.WriteLine("\n⚡ Quick Sort by Total Price:");

            Order[] quickSorted = (Order[])orders.Clone();

            SortHelper.QuickSort(quickSorted);

            DisplayOrders(quickSorted);

             Console.ReadKey();

        }

         static void DisplayOrders(Order[] orders)

        {

            foreach (var order in orders)

            {

                Console.WriteLine(order);

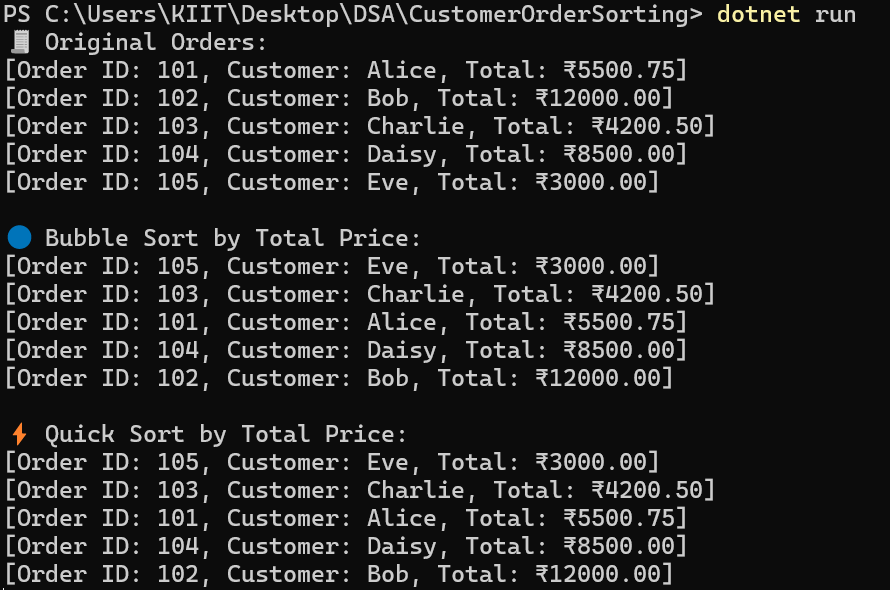
            }

        }

    }

}

Output:



**Exercise 4: Employee Management System**

Codes :

Employee.cs:

public class Employee

{

    public int EmployeeId { get; set; }

    public string Name { get; set; }

    public string Position { get; set; }

    public double Salary { get; set; }

    public Employee(int id, string name, string position, double salary)

    {

        EmployeeId = id;

        Name = name;

        Position = position;

        Salary = salary;

    }

    public override string ToString()

    {

        return $"ID: {EmployeeId}, Name: {Name}, Position: {Position}, Salary: ₹{Salary}";

    }

}

EmployeeManager.cs:

using System;

public class EmployeeManager

{

    private Employee[] employees;

    private int count;

    public EmployeeManager(int size)

    {

        employees = new Employee[size];

        count = 0;

    }

    public void AddEmployee(Employee emp)

    {

        if (count < employees.Length)

        {

            employees[count++] = emp;

            Console.WriteLine("Employee added.");

        }

        else

        {

            Console.WriteLine("Employee list is full.");

        }

    }

    public void SearchEmployee(int id)

    {

        for (int i = 0; i < count; i++)

        {

            if (employees[i].EmployeeId == id)

            {

                Console.WriteLine("Employee found: " + employees[i]);

                return;

            }

        }

        Console.WriteLine("Employee not found.");

    }

    public void TraverseEmployees()

    {

        if (count == 0)

        {

            Console.WriteLine("No employees to display.");

            return;

        }

        for (int i = 0; i < count; i++)

        {

            Console.WriteLine(employees[i]);

        }

    }

    public void DeleteEmployee(int id)

    {

        for (int i = 0; i < count; i++)

        {

            if (employees[i].EmployeeId == id)

            {

                for (int j = i; j < count - 1; j++)

                {

                    employees[j] = employees[j + 1];

                }

                employees[--count] = null;

                Console.WriteLine("Employee deleted.");

                return;

            }

        }

        Console.WriteLine("Employee not found.");

    }

}

Program.cs:

using System;

class Program

{

    static void Main()

    {

        EmployeeManager manager = new EmployeeManager(5); // Array size = 5

        bool exit = false;

         while (!exit)

        {

            Console.WriteLine("\nEmployee Management System");

            Console.WriteLine("1. Add Employee");

            Console.WriteLine("2. Search Employee");

            Console.WriteLine("3. Traverse Employees");

            Console.WriteLine("4. Delete Employee");

            Console.WriteLine("5. Exit");

            Console.Write("Choose an option: ");

            string choice = Console.ReadLine();

             switch (choice)

            {

                case "1":

                    Console.Write("Enter ID: ");

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

                    Console.Write("Enter Name: ");

                    string name = Console.ReadLine();

                    Console.Write("Enter Position: ");

                    string position = Console.ReadLine();

                    Console.Write("Enter Salary: ");

                    double salary = double.Parse(Console.ReadLine());

                    manager.AddEmployee(new Employee(id, name, position, salary));

                    break;

                     case "2":

                    Console.Write("Enter ID to search: ");

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

                    manager.SearchEmployee(searchId);

                    break;

                case "3":

                    manager.TraverseEmployees();

                    break;

                     case "4":

                    Console.Write("Enter ID to delete: ");

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

                    manager.DeleteEmployee(deleteId);

                    break;

                case "5":

                    exit = true;

                    break;

                     default:

                    Console.WriteLine("Invalid choice.");

                    break;

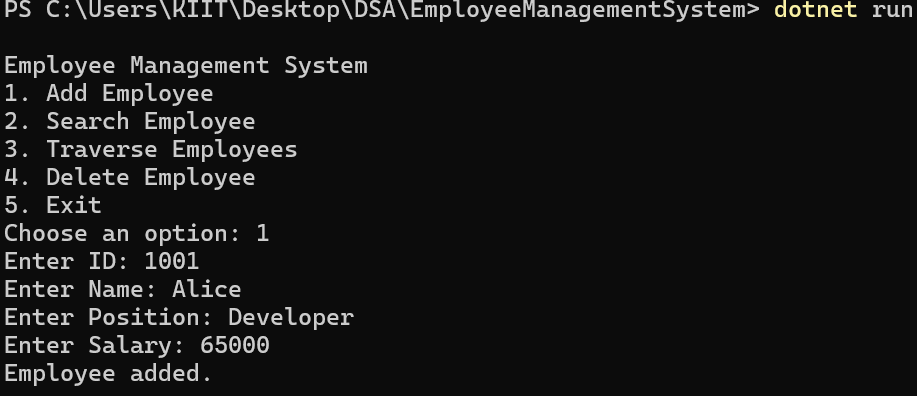
            }

        }

    }

}

Output:



**Exercise 5: Task Management System**

Codes :

Task.cs:

public class Task

{

    public int TaskId { get; set; }

    public string TaskName { get; set; }

    public string Status { get; set; }

    public Task(int id, string name, string status)

    {

        TaskId = id;

        TaskName = name;

        Status = status;

    }

     public override string ToString()

    {

        return $"[ID: {TaskId}, Name: {TaskName}, Status: {Status}]";

    }

}

TaskNode.cs:

public class TaskNode

{

    public Task Data { get; set; }

    public TaskNode Next { get; set; }

    public TaskNode(Task task)

    {

        Data = task;

        Next = null;

    }

}

TaskLinkedList.cs:

public class TaskLinkedList

{

    private TaskNode head;

    public void AddTask(Task task)

    {

        TaskNode newNode = new TaskNode(task);

        if (head == null)

        {

            head = newNode;

            return;

        }

        TaskNode current = head;

        while (current.Next != null)

        {

            current = current.Next;

        }

        current.Next = newNode;

    }

    public void DisplayTasks()

    {

        if (head == null)

        {

            Console.WriteLine("Task list is empty.");

            return;

        }

         TaskNode current = head;

        while (current != null)

        {

            Console.WriteLine(current.Data);

            current = current.Next;

        }

    }

    public Task SearchTask(int id)

    {

        TaskNode current = head;

        while (current != null)

        {

            if (current.Data.TaskId == id)

                return current.Data;

            current = current.Next;

        }

        return null;

    }

    public void DeleteTask(int id)

    {

        if (head == null) return;

        if (head.Data.TaskId == id)

        {

            head = head.Next;

            return;

        }

        TaskNode current = head;

        while (current.Next != null && current.Next.Data.TaskId != id)

        {

            current = current.Next;

        }

         if (current.Next != null)

        {

            current.Next = current.Next.Next;

        }

    }

}

Program.cs:

class Program

{

    static void Main(string[] args)

    {

        TaskLinkedList taskList = new TaskLinkedList();

         taskList.AddTask(new Task(1, "Design Database", "Pending"));

        taskList.AddTask(new Task(2, "Implement API", "Pending"));

        taskList.AddTask(new Task(3, "Write Tests", "Completed"));

        Console.WriteLine("All Tasks:");

        taskList.DisplayTasks();

        Console.WriteLine("\nSearching Task with ID 2:");

        Task foundTask = taskList.SearchTask(2);

        Console.WriteLine(foundTask != null ? foundTask.ToString() : "Task not found.");

        Console.WriteLine("\nDeleting Task with ID 1.");

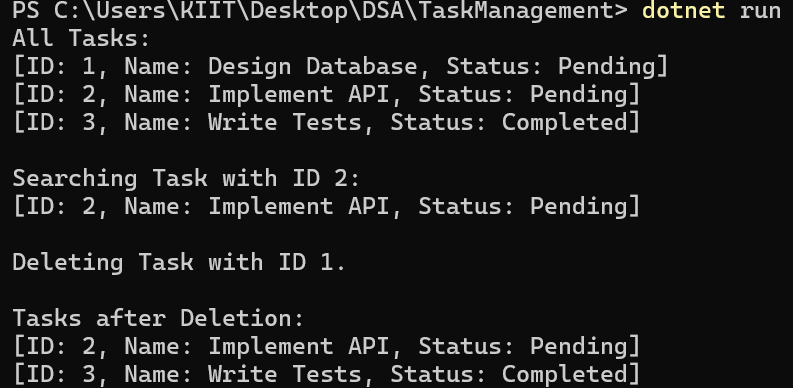
        taskList.DeleteTask(1);

        Console.WriteLine("\nTasks after Deletion:");

        taskList.DisplayTasks();

    }

}

Output:

**Exercise 6: Library Management System**

Codes :

Book.cs:

namespace LibraryManagementSystem

{

     public class Book

    {

        public int BookId { get; set; }

        public string Title { get; set; }

        public string Author { get; set; }

        public Book(int id, string title, string author)

        {

            BookId = id;

            Title = title;

            Author = author;

        }

        public override string ToString()

        {

            return $"[ID: {BookId}, Title: \"{Title}\", Author: {Author}]";

        }

    }

}

SearchHelper.cs:

using System;

using System.Collections.Generic;

namespace LibraryManagementSystem

{

    public static class SearchHelper

    {

        public static Book LinearSearch(List<Book> books, string title)

        {

            foreach (Book book in books)

            {

                if (book.Title.Equals(title, StringComparison.OrdinalIgnoreCase))

                    return book;

            }

            return null;

        }

         public static Book BinarySearch(List<Book> books, string title)

        {

            int left = 0, right = books.Count - 1;

            while (left <= right)

            {

                int mid = (left + right) / 2;

                int comparison = string.Compare(books[mid].Title, title, true);

                if (comparison == 0)

                    return books[mid];

                else if (comparison < 0)

                    left = mid + 1;

                else

                    right = mid - 1;

            }

            return null;

        }

    }

}

Program.cs:

using System;

using System.Collections.Generic;

namespace LibraryManagementSystem

{

    class Program

    {

        static void Main(string[] args)

        {

          List<Book> library = new List<Book>

            {

                new Book(101, "C Programming", "Dennis Ritchie"),

                new Book(102, "Data Structures", "Mark Weiss"),

                new Book(103, "Algorithms", "Robert Sedgewick"),

                new Book(104, "Operating Systems", "Andrew Tanenbaum"),

                new Book(105, "Computer Networks", "James Kurose")

            };

            Console.WriteLine("== Linear Search for 'Algorithms' ==");

            Book linearResult = SearchHelper.LinearSearch(library, "Algorithms");

            Console.WriteLine(linearResult != null ? linearResult.ToString() : "Book not found.");

            library.Sort((b1, b2) => string.Compare(b1.Title, b2.Title));

            Console.WriteLine("\n== Binary Search for 'Operating Systems' ==");

            Book binaryResult = SearchHelper.BinarySearch(library, "Operating Systems");

            Console.WriteLine(binaryResult != null ? binaryResult.ToString() : "Book not found.");

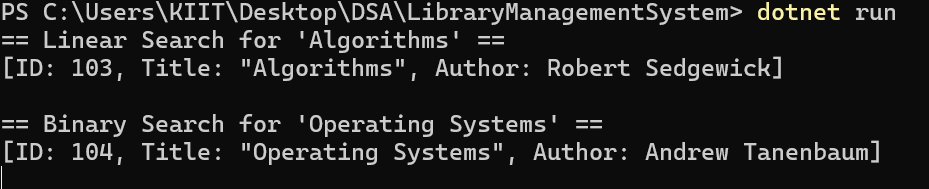
            Console.ReadKey();

        }

    }

}

Output:



**Exercise 7: Financial Forecasting**

Codes :

ForecastCalculator.cs:

namespace FinancialForecasting

{

    public static class ForecastCalculator

    {

        public static double CalculateFutureValue(double presentValue, double growthRate, int years)

        {

            if (years == 0)

                return presentValue;

            return (1 + growthRate) \* CalculateFutureValue(presentValue, growthRate, years - 1);

        }

         public static double CalculateFutureValueIterative(double presentValue, double growthRate, int years)

        {

            double result = presentValue;

            for (int i = 0; i < years; i++)

            {

                result \*= (1 + growthRate);

            }

            return result;

        }

    }

}

Program.cs:

using System;

namespace FinancialForecasting

{

    class Program

    {

         static void Main(string[] args)

        {

            Console.WriteLine("📊 Financial Forecasting Tool");

            Console.Write("Enter Present Value: ");

            double presentValue = double.Parse(Console.ReadLine());

            Console.Write("Enter Annual Growth Rate (in %): ");

            double growthRatePercent = double.Parse(Console.ReadLine());

            double growthRate = growthRatePercent / 100.0;

            Console.Write("Enter Number of Years: ");

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

            double futureValueRecursive = ForecastCalculator.CalculateFutureValue(presentValue, growthRate, years);

            Console.WriteLine($"\n[Recursive] Future Value after {years} years: {futureValueRecursive:C2}");

            double futureValueIterative = ForecastCalculator.CalculateFutureValueIterative(presentValue, growthRate, years);

            Console.WriteLine($"[Iterative] Future Value after {years} years: {futureValueIterative:C2}");

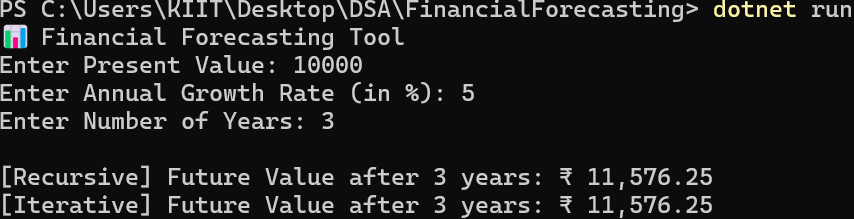
            Console.ReadKey();

        }

    }

}

Output:

.