**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Solution :**

**Program :**

using System;

using System.Collections.Generic;

class Product

{

    public int ProductId { get; set; }

    public string ProductName { get; set; } = "";

    public int Quantity { get; set; }

    public double Price { get; set; }

    public override string ToString()

    {

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

    }

}

class Inventory

{

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

    public void AddProduct(Product product)

    {

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

        {

            products[product.ProductId] = product;

            Console.WriteLine("Product added.");

        }

        else

        {

            Console.WriteLine("Product ID already exists.");

        }

    }

    public void UpdateProduct(Product product)

    {

        if (products.ContainsKey(product.ProductId))

        {

            products[product.ProductId] = product;

            Console.WriteLine("Product updated.");

        }

        else

        {

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

        }

    }

    public void DeleteProduct(int productId)

    {

        if (products.Remove(productId))

        {

            Console.WriteLine("Product deleted.");

        }

        else

        {

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

        }

    }

    public void DisplayAllProducts()

    {

        foreach (var product in products.Values)

        {

            Console.WriteLine(product);

        }

    }

}

class Program

{

    static void Main()

    {

        Inventory inventory = new();

        bool running = true;

        while (running)

        {

            Console.WriteLine("\n1. Add Product\n2. Update Product\n3. Delete Product\n4. View All Products\n5. Exit");

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

            var choice = Console.ReadLine();

            switch (choice)

            {

                case "1":

                    Product p1 = ReadProductDetails();

                    inventory.AddProduct(p1);

                    break;

                case "2":

                    Product p2 = ReadProductDetails();

                    inventory.UpdateProduct(p2);

                    break;

                case "3":

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

                    int delId = int.Parse(Console.ReadLine()!);

                    inventory.DeleteProduct(delId);

                    break;

                case "4":

                    inventory.DisplayAllProducts();

                    break;

                case "5":

                    running = false;

                    break;

                default:

                    Console.WriteLine("Invalid option.");

                    break;

            }

        }

    }

    static Product ReadProductDetails()

    {

        Console.Write("Enter Product ID: ");

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

        Console.Write("Enter Product Name: ");

        string name = Console.ReadLine()!;

        Console.Write("Enter Quantity: ");

        int qty = int.Parse(Console.ReadLine()!);

        Console.Write("Enter Price: ");

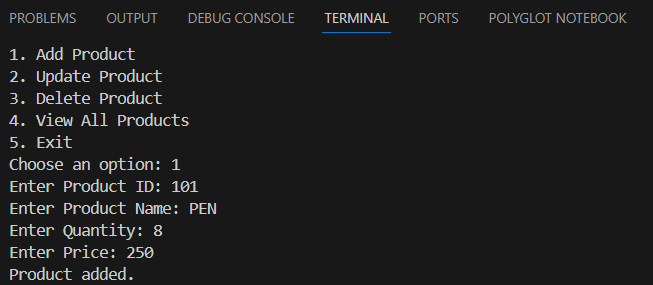
        double price = double.Parse(Console.ReadLine()!);

        return new Product { ProductId = id, ProductName = name, Quantity = qty, Price = price };

    }

}

**Output :**



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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Solution :**

**Program :**

using System;

class Product

{

    public int ProductId { get; set; }

    public string ProductName { get; set; } = "";

    public string Category { get; set; } = "";

    public override string ToString()

    {

        return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";

    }

}

class SearchEngine

{

    public static int LinearSearch(Product[] products, string targetName)

    {

        for (int i = 0; i < products.Length; i++)

        {

            if (products[i].ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))

            {

                return i;

            }

        }

        return -1;

    }

    public static int BinarySearch(Product[] products, string targetName)

    {

        int left = 0, right = products.Length - 1;

        while (left <= right)

        {

            int mid = (left + right) / 2;

            int result = string.Compare(products[mid].ProductName, targetName, true);

            if (result == 0)

                return mid;

            else if (result < 0)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return -1;

    }

}

class Program

{

    static void Main()

    {

        Product[] products = new Product[]

        {

            new Product { ProductId = 1, ProductName = "Mouse", Category = "Electronics" },

            new Product { ProductId = 2, ProductName = "Keyboard", Category = "Electronics" },

            new Product { ProductId = 3, ProductName = "Shoes", Category = "Fashion" },

            new Product { ProductId = 4, ProductName = "T-Shirt", Category = "Fashion" },

            new Product { ProductId = 5, ProductName = "Laptop", Category = "Electronics" }

        };

        Console.Write("Enter product name to search: ");

        string searchName = Console.ReadLine()!;

        Console.WriteLine("\n--- Linear Search ---");

        int index1 = SearchEngine.LinearSearch(products, searchName);

        if (index1 != -1)

            Console.WriteLine("Product found: " + products[index1]);

        else

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

        Console.WriteLine("\n--- Binary Search (requires sorted array) ---");

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

        int index2 = SearchEngine.BinarySearch(products, searchName);

        if (index2 != -1)

            Console.WriteLine("Product found: " + products[index2]);

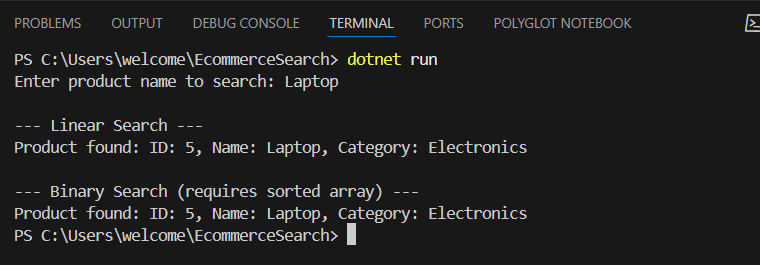
        else

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

    }

}

**Output :**



**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**Solution :**

**Program :**

using System;

class Order

{

public int OrderId { get; set; }

public string CustomerName { get; set; } = "";

public double TotalPrice { get; set; }

public override string ToString()

{

return $"OrderID: {OrderId}, Customer: {CustomerName}, TotalPrice: {TotalPrice}";

}

}

class Sorting

{

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)

{

(orders[j], orders[j + 1]) = (orders[j + 1], orders[j]);

}

}

}

}

public static void QuickSort(Order[] orders, int low, int high)

{

if (low < high)

{

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

QuickSort(orders, low, pivotIndex - 1);

QuickSort(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;

}

}

class Program

{

static void PrintOrders(Order[] orders)

{

foreach (var order in orders)

{

Console.WriteLine(order);

}

}

static void Main()

{

Order[] orders = new Order[]

{

new Order { OrderId = 101, CustomerName = "Alice", TotalPrice = 950.5 },

new Order { OrderId = 102, CustomerName = "Bob", TotalPrice = 1200.0 },

new Order { OrderId = 103, CustomerName = "Charlie", TotalPrice = 450.75 },

new Order { OrderId = 104, CustomerName = "Diana", TotalPrice = 2300.0 }

};

Console.WriteLine("Original Orders:");

PrintOrders(orders);

// Bubble Sort

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

Sorting.BubbleSort(bubbleSorted);

Console.WriteLine("\nOrders Sorted by Bubble Sort (Ascending TotalPrice):");

PrintOrders(bubbleSorted);

// Quick Sort

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

Sorting.QuickSort(quickSorted, 0, quickSorted.Length - 1);

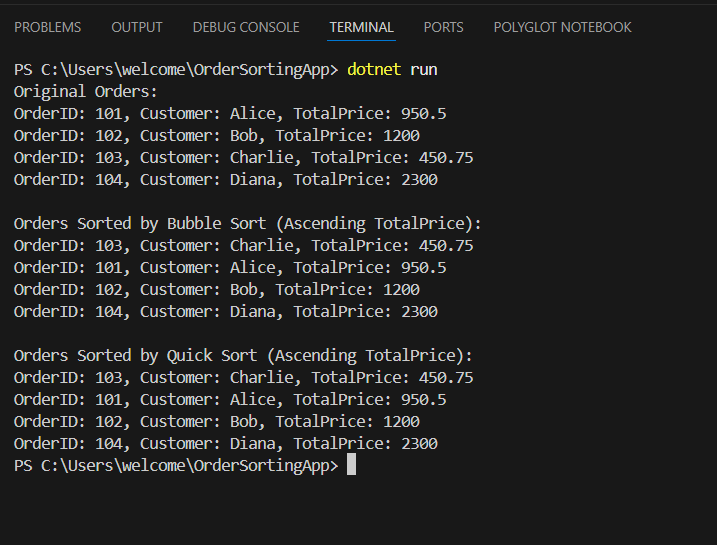
Console.WriteLine("\nOrders Sorted by Quick Sort (Ascending TotalPrice):");

PrintOrders(quickSorted);

}

}

**Output :**



**Exercise 4: Employee Management System**

**Scenario:**

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

**Solution :**

**Program :**

using System;

class Employee

{

public int EmployeeId { get; set; }

public string Name { get; set; } = "";

public string Position { get; set; } = "";

public double Salary { get; set; }

public override string ToString()

{

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

}

}

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 DisplayAll()

{

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

{

Console.WriteLine(employees[i]);

}

}

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 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.");

}

}

class Program

{

static void Main()

{

EmployeeManager manager = new(100);

bool running = true;

while (running)

{

Console.WriteLine("\n1. Add Employee\n2. Search Employee\n3. Display All Employees\n4. Delete Employee\n5. Exit");

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

string? option = Console.ReadLine();

switch (option)

{

case "1":

Employee emp = new();

Console.Write("Enter ID: ");

emp.EmployeeId = int.Parse(Console.ReadLine()!);

Console.Write("Enter Name: ");

emp.Name = Console.ReadLine()!;

Console.Write("Enter Position: ");

emp.Position = Console.ReadLine()!;

Console.Write("Enter Salary: ");

emp.Salary = double.Parse(Console.ReadLine()!);

manager.AddEmployee(emp);

break;

case "2":

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

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

manager.SearchEmployee(searchId);

break;

case "3":

manager.DisplayAll();

break;

case "4":

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

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

manager.DeleteEmployee(deleteId);

break;

case "5":

running = false;

break;

default:

Console.WriteLine("Invalid option.");

break;

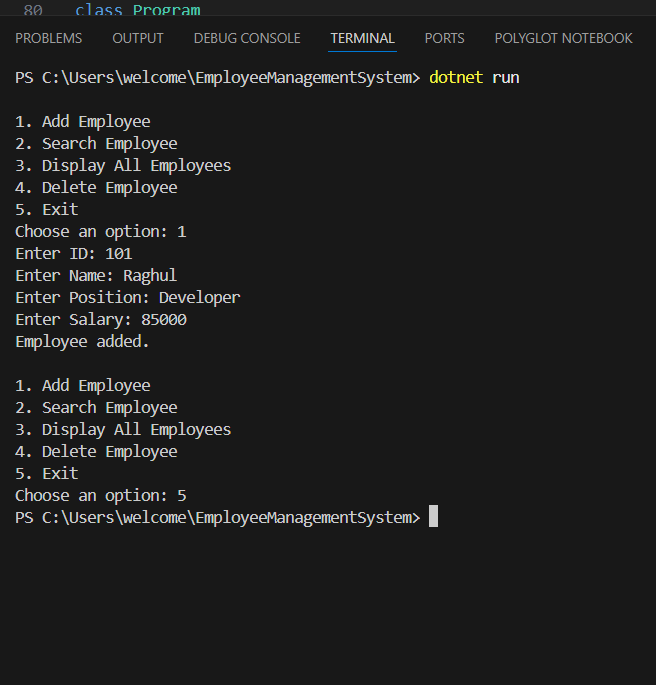
}

}

}

}

**Output :**



**Exercise 5: Task Management System**

**Scenario:**

**You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.**

**Solution :**

**Program :**

using System;

class Task

{

public int TaskId { get; set; }

public string TaskName { get; set; } = "";

public string Status { get; set; } = "";

public override string ToString()

{

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

}

}

class TaskNode

{

public Task Task { get; set; }

public TaskNode? Next { get; set; }

public TaskNode(Task task)

{

Task = task;

Next = null;

}

}

class TaskLinkedList

{

private TaskNode? head;

public void AddTask(Task task)

{

TaskNode newNode = new(task);

if (head == null)

{

head = newNode;

}

else

{

TaskNode current = head;

while (current.Next != null)

current = current.Next;

current.Next = newNode;

}

Console.WriteLine("Task added.");

}

public void DisplayTasks()

{

TaskNode? current = head;

while (current != null)

{

Console.WriteLine(current.Task);

current = current.Next;

}

}

public void SearchTask(int id)

{

TaskNode? current = head;

while (current != null)

{

if (current.Task.TaskId == id)

{

Console.WriteLine("Task found: " + current.Task);

return;

}

current = current.Next;

}

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

}

public void DeleteTask(int id)

{

TaskNode? current = head;

TaskNode? previous = null;

while (current != null)

{

if (current.Task.TaskId == id)

{

if (previous == null)

{

head = current.Next;

}

else

{

previous.Next = current.Next;

}

Console.WriteLine("Task deleted.");

return;

}

previous = current;

current = current.Next;

}

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

}

}

class Program

{

static void Main()

{

TaskLinkedList taskList = new();

bool running = true;

while (running)

{

Console.WriteLine("\n1. Add Task\n2. Search Task\n3. Display All Tasks\n4. Delete Task\n5. Exit");

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

string? option = Console.ReadLine();

switch (option)

{

case "1":

Task task = new();

Console.Write("Enter Task ID: ");

task.TaskId = int.Parse(Console.ReadLine()!);

Console.Write("Enter Task Name: ");

task.TaskName = Console.ReadLine()!;

Console.Write("Enter Task Status: ");

task.Status = Console.ReadLine()!;

taskList.AddTask(task);

break;

case "2":

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

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

taskList.SearchTask(searchId);

break;

case "3":

taskList.DisplayTasks();

break;

case "4":

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

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

taskList.DeleteTask(deleteId);

break;

case "5":

running = false;

break;

default:

Console.WriteLine("Invalid option.");

break;

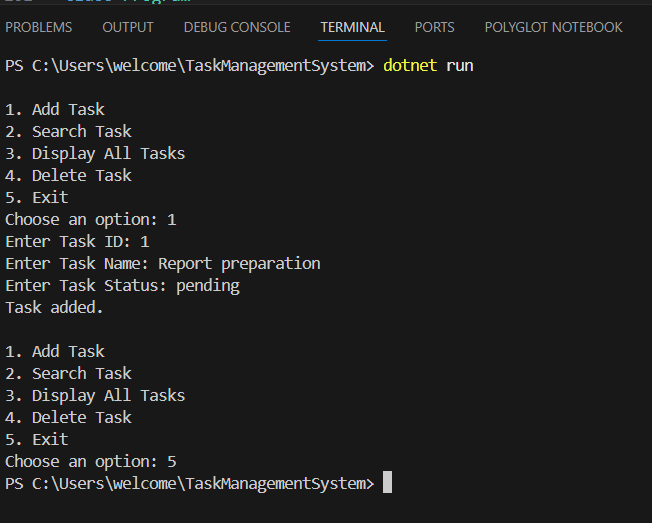
}

}

}

}

**Output :**



**Exercise 6: Library Management System**

**Scenario:**

You are developing a library management system where users can search for books by title or author.

**Solution :**

using System;

class Book

{

    public int BookId { get; set; }

    public string Title { get; set; } = "";

    public string Author { get; set; } = "";

    public override string ToString()

    {

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

    }

}

class BookSearch

{

    public static int LinearSearch(Book[] books, string targetTitle)

    {

        for (int i = 0; i < books.Length; i++)

        {

            if (books[i].Title.Equals(targetTitle, StringComparison.OrdinalIgnoreCase))

            {

                return i;

            }

        }

        return -1;

    }

    public static int BinarySearch(Book[] books, string targetTitle)

    {

        int low = 0;

        int high = books.Length - 1;

        while (low <= high)

        {

            int mid = (low + high) / 2;

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

            if (comparison == 0)

                return mid;

            else if (comparison < 0)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return -1;

    }

}

class Program

{

    static void Main()

    {

        Book[] books = new Book[]

        {

            new Book { BookId = 1, Title = "Data Structures", Author = "Mark Allen" },

            new Book { BookId = 2, Title = "Algorithms", Author = "Thomas Cormen" },

            new Book { BookId = 3, Title = "C# Programming", Author = "Andrew Troelsen" },

            new Book { BookId = 4, Title = "Operating Systems", Author = "Abraham Silberschatz" },

            new Book { BookId = 5, Title = "Computer Networks", Author = "Andrew Tanenbaum" }

        };

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

        string target = Console.ReadLine()!;

        // Linear Search

        Console.WriteLine("\n--- Linear Search ---");

        int index1 = BookSearch.LinearSearch(books, target);

        if (index1 != -1)

            Console.WriteLine("Book Found: " + books[index1]);

        else

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

        // Binary Search - requires sorted array

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

        Array.Sort(books, (a, b) => string.Compare(a.Title, b.Title, StringComparison.OrdinalIgnoreCase));

        int index2 = BookSearch.BinarySearch(books, target);

        if (index2 != -1)

            Console.WriteLine("Book Found: " + books[index2]);

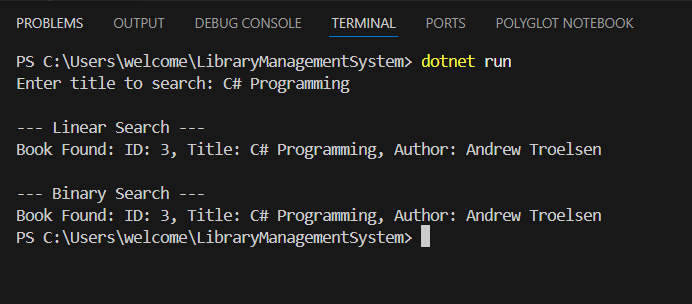
        else

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

    }

}

**Output :**



**Exercise 7: Financial Forecasting**

**Scenario:**

**You are developing a financial forecasting tool that predicts future values based on past data.**

**Solution :**

**Program :**

using System;

class Forecast

{

// Recursive method to calculate future value

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

{

if (years == 0)

return presentValue;

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

}

// Optimized version using memoization (optional extension)

public static double PredictWithMemo(double presentValue, double growthRate, int years, double[] memo)

{

if (years == 0)

return presentValue;

if (memo[years] != 0)

return memo[years];

memo[years] = (1 + growthRate) \* PredictWithMemo(presentValue, growthRate, years - 1, memo);

return memo[years];

}

}

class Program

{

static void Main()

{

Console.Write("Enter Present Value (e.g., 1000): ");

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

Console.Write("Enter Annual Growth Rate (as decimal, e.g., 0.05 for 5%): ");

double growthRate = double.Parse(Console.ReadLine()!);

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

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

Console.WriteLine("\n--- Using Simple Recursion ---");

double futureValue = Forecast.PredictFutureValue(presentValue, growthRate, years);

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

Console.WriteLine("\n--- Using Memoized Recursion ---");

double[] memo = new double[years + 1];

double memoValue = Forecast.PredictWithMemo(presentValue, growthRate, years, memo);

Console.WriteLine($"Predicted Future Value after {years} years (memoized): {memoValue:F2}");

}

}

**Output :**

