**Algorithms Data Structures**

**Exercise 1: E-commerce Platform Search Function**

**Product.cs :-**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public double Price { get; set; }

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

{

Id = id;

Name = name;

Category = category;

Price = price;

}

public override string ToString()

{

return $"✅ {Name} (ID: {Id}) – {Category} – ₹{Price}";

}

}

**ProductSearchDemo.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**Program.cs :-**

using System;

class Program

{

static void Main()

{

Product[] products = new Product[]

{

new Product(101, "Laptop", "Electronics", 49999),

new Product(102, "Shirt", "Clothing", 999),

new Product(103, "Watch", "Accessories", 1499),

new Product(104, "Camera", "Electronics", 39999),

new Product(105, "Shoes", "Footwear", 1999),

new Product(106, "Mouse", "Electronics", 599),

new Product(107, "Shirt", "Clothing", 899),

new Product(108, "Bag", "Accessories", 1199)

};

Console.WriteLine("\n🔎 Choose Search Type:");

Console.WriteLine("1. Search by Product ID");

Console.WriteLine("2. Search by Product Name");

Console.WriteLine("3. Search by Category");

Console.WriteLine("4. Search by Price Range");

Console.Write("Enter your choice (1–4): ");

string? choice = Console.ReadLine();

switch (choice)

{

case "1":

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

int id = int.Parse(Console.ReadLine() ?? "0");

SearchHelper.SearchById(products, id);

break;

case "2":

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

string name = Console.ReadLine() ?? "";

SearchHelper.SearchByName(products, name);

break;

case "3":

Console.Write("Enter Category: ");

string category = Console.ReadLine() ?? "";

SearchHelper.SearchByCategory(products, category);

break;

case "4":

Console.Write("Enter Min Price: ");

double min = double.Parse(Console.ReadLine() ?? "0");

Console.Write("Enter Max Price: ");

double max = double.Parse(Console.ReadLine() ?? "0");

SearchHelper.SearchByPriceRange(products, min, max);

break;

default:

Console.WriteLine("⚠️ Invalid choice.");

break;

}

Console.WriteLine("\n📁 Results saved to SearchResults.txt");

}

}

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

private static readonly string resultFile = "SearchResults.txt";

public static void LogResult(string content)

{

Console.WriteLine(content);

File.AppendAllText(resultFile, content + Environment.NewLine);

}

public static void LogHeader(string header)

{

File.WriteAllText(resultFile, header + Environment.NewLine);

Console.WriteLine(header);

}

public static void LogFooter()

{

string footer = $"🕒 Generated at: {DateTime.Now}";

File.AppendAllText(resultFile, footer + Environment.NewLine);

Console.WriteLine(footer);

}

}

**SearchHelper.cs :-**

using System;

using System.Diagnostics;

using System.Linq;

public static class SearchHelper

{

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

{

var stopwatch = Stopwatch.StartNew();

var product = products.FirstOrDefault(p => p.Id == id);

stopwatch.Stop();

Logger.LogHeader($"🔍 Search Result – Product ID: {id}");

if (product != null)

Logger.LogResult(product.ToString());

else

Logger.LogResult("❌ Product not found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByName(Product[] products, string name)

{

var stopwatch = Stopwatch.StartNew();

var matches = products

.Where(p => p.Name.Equals(name, StringComparison.OrdinalIgnoreCase))

.ToList();

stopwatch.Stop();

Logger.LogHeader($"🔍 Search Results – Product Name: {name}");

if (matches.Count > 0)

{

foreach (var product in matches)

Logger.LogResult(product.ToString());

}

else

Logger.LogResult("❌ No products found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByCategory(Product[] products, string category)

{

var stopwatch = Stopwatch.StartNew();

var matches = products

.Where(p => p.Category.Equals(category, StringComparison.OrdinalIgnoreCase))

.ToList();

stopwatch.Stop();

Logger.LogHeader($"📂 Search Results – Category: {category}");

if (matches.Count > 0)

{

foreach (var product in matches)

Logger.LogResult(product.ToString());

}

else

Logger.LogResult("❌ No products found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByPriceRange(Product[] products, double minPrice, double maxPrice)

{

var stopwatch = Stopwatch.StartNew();

var filtered = products

.Where(p => p.Price >= minPrice && p.Price <= maxPrice)

.ToList();

stopwatch.Stop();

Logger.LogHeader($"📦 Search Results – Price Range: ₹{minPrice} – ₹{maxPrice}");

foreach (var product in filtered)

Logger.LogResult(product.ToString());

if (filtered.Count == 0)

Logger.LogResult("❌ No products found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

}

**OUTPUT :-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**SerachResult.txt :-**

📦 Search Results – Price Range: ₹500 – ₹1000

✅ Shirt (ID: 102) – Clothing – ₹999

✅ Mouse (ID: 106) – Electronics – ₹599

✅ Shirt (ID: 107) – Clothing – ₹899

⌛ Time Complexity: O(n), Time Taken: 0.9334 ms

🕒 Generated at: 6/21/2025 9:04:32 PM

**Time Complexity :**

**By Product ID / Name / Category (Linear Search):**

* **Best Case:** O(1)
* **Average Case:** O(n)
* **Worst Case:** O(n)

**By Price Range:**

* **Best Case:** O(1)
* **Average Case:** O(n)
* **Worst Case:** O(n)

**By Product ID (Binary Search on Sorted Data):**

* **Best Case:** O(1)
* **Average Case:** O(log n)
* **Worst Case:** O(log n)

**Exercise 2: Financial Forecasting**

**Program.cs :-**

using System;

using System.Diagnostics;

using System.IO;

class Program

{

static double CalculateForecast(double amount, double rate, int years)

{

// Base case

if (years == 0)

return amount;

// Recursive case: compound annually

return CalculateForecast(amount \* (1 + rate / 100), rate, years - 1);

}

static void Main()

{

Console.Write("Enter initial amount: ");

double initialAmount = double.Parse(Console.ReadLine() ?? "0");

Console.Write("Enter number of years to forecast: ");

int years = int.Parse(Console.ReadLine() ?? "0");

Console.Write("How many growth rates do you want to compare? ");

int rateCount = int.Parse(Console.ReadLine() ?? "0");

double[] rates = new double[rateCount];

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

{

Console.Write($"Enter growth rate {i + 1} (as %): ");

rates[i] = double.Parse(Console.ReadLine() ?? "0");

}

string filePath = "forecast.txt";

using StreamWriter writer = new StreamWriter(filePath);

writer.WriteLine($"📈 Forecast for Initial Amount: ₹{initialAmount} over {years} years:");

writer.WriteLine(new string('-', 50));

var stopwatch = Stopwatch.StartNew();

foreach (double rate in rates)

{

double result = CalculateForecast(initialAmount, rate, years);

string line = $"Growth Rate {rate}% → ₹{result:F2}";

Console.WriteLine(line);

writer.WriteLine(line);

}

stopwatch.Stop();

Console.WriteLine("✔ Forecast saved to forecast.txt");

writer.WriteLine(new string('-', 50));

writer.WriteLine($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

}

}

**FinancialForecasting.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**Time Complexity :**

**Calculate Forecast (n years):**

* **Best Case:** O(n)
* **Average Case:** O(n)
* **Worst Case:** O(n)

**Recursion :-**

The recursive algorithm used in the financial forecasting tool calculates the future value by compounding the amount year by year. Each recursive call represents one year of compounding, and the recursion continues until the base case is reached (when years = 0). This leads to a **time complexity of O(n)**, where n is the number of years. The best, average, and worst cases all have the same complexity because the number of calls is always directly proportional to the number of years provided.

**OUTPUT :-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Forecast.txt :-**

📈 Forecast for Initial Amount: ₹10000 over 5 years:

--------------------------------------------------

Growth Rate 5% → ₹12762.82

Growth Rate 10% → ₹16105.10

--------------------------------------------------

⌛ Time Complexity: O(n), Time Taken: 3.4291 ms

**Optimization :-**

The original recursive approach to financial forecasting calculates the compounded value by calling the function once for each year. While functionally correct, recursion introduces additional overhead due to repeated function calls and memory stack usage. For large values of years, this can lead to performance issues or even stack overflow errors. A simple and effective way to optimize the solution without using the direct formula is to replace the recursion with an iterative approach. By using a for loop to compound the amount year by year, we maintain the same time complexity of **O(n)** while improving efficiency and eliminating the risk of stack overflow. The iteration is more memory-friendly and easier to debug or extend, making it ideal for practical scenarios where input size could grow.

**Exercise 3: Employee Management System**

**Program.cs :-**

using System;

using System.Diagnostics;

class Program

{

static Employee?[] employees = new Employee?[100];

static int count = 0;

static void AddEmployee(Employee emp)

{

var stopwatch = Stopwatch.StartNew();

if (count >= employees.Length)

{

stopwatch.Stop();

Logger.Log($"❌ Employee array is full. [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

employees[count++] = emp;

stopwatch.Stop();

Logger.Log($"✅ Employee added: {emp.EmployeeId}, {emp.Name}, {emp.Position}, ₹{emp.Salary} [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void SearchEmployee(int id)

{

var stopwatch = Stopwatch.StartNew();

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

{

var emp = employees[i];

if (emp != null && emp.EmployeeId == id)

{

stopwatch.Stop();

Logger.Log($"🔍 Found Employee ID {id}: {emp.Name}, {emp.Position}, ₹{emp.Salary} [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

}

stopwatch.Stop();

Logger.Log($"❌ Employee ID {id} not found. [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void TraverseEmployees()

{

var stopwatch = Stopwatch.StartNew();

if (count == 0)

{

stopwatch.Stop();

Logger.Log($"ℹ️ No employees to display. [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

Logger.Log($"📋 Traversing Employee List: [Time Complexity: O(n)]");

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

{

var emp = employees[i];

if (emp != null)

{

Logger.Log($"ID: {emp.EmployeeId}, Name: {emp.Name}, Position: {emp.Position}, Salary: ₹{emp.Salary}");

}

}

stopwatch.Stop();

Logger.Log($"✅ Traversal Complete. Duration: {stopwatch.ElapsedMilliseconds} ms");

}

static void DeleteEmployee(int id)

{

var stopwatch = Stopwatch.StartNew();

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

{

var emp = employees[i];

if (emp != null && emp.EmployeeId == id)

{

string deletedInfo = $"ID: {emp.EmployeeId}, Name: {emp.Name}";

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

{

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

}

employees[--count] = null;

stopwatch.Stop();

Logger.Log($"🗑 Deleted Employee → {deletedInfo} [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

}

stopwatch.Stop();

Logger.Log($"❌ Employee ID {id} not found for deletion. [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void Main()

{

AddEmployee(new Employee(1, "Alekya", "Developer", 65000));

AddEmployee(new Employee(2, "Ravi", "Manager", 85000));

AddEmployee(new Employee(3, "Sara", "Analyst", 55000));

TraverseEmployees();

Console.WriteLine();

SearchEmployee(2);

Console.WriteLine();

DeleteEmployee(2);

TraverseEmployees();

}

}

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

private static readonly string logPath = "employee\_log.txt";

static Logger()

{

File.AppendAllText(logPath, $"\n--- Log Started at {DateTime.Now} ---\n");

}

public static void Log(string message)

{

string logMessage = $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] {message}";

Console.WriteLine(logMessage);

File.AppendAllText(logPath, logMessage + Environment.NewLine);

}

}

**EmployeeManagement.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**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 employeeId, string name, string position, double salary)

{

EmployeeId = employeeId;

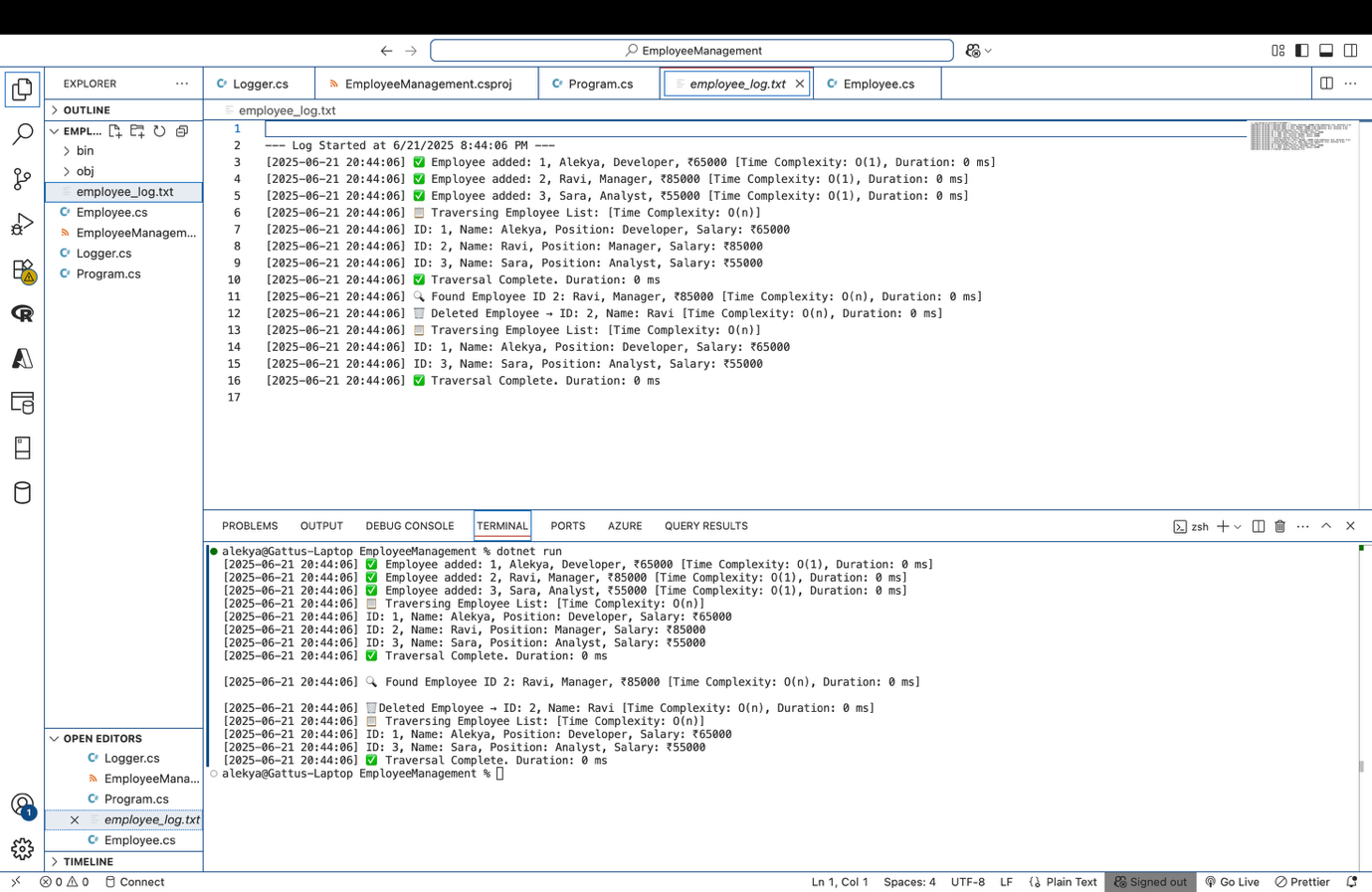
Name = name;

Position = position;

Salary = salary;

} }

**OUTPUT :-**



**Employee\_log.txt :-**

--- Log Started at 6/21/2025 8:44:06 PM ---

[2025-06-21 20:44:06] ✅ Employee added: 1, Alekya, Developer, ₹65000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] ✅ Employee added: 2, Ravi, Manager, ₹85000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] ✅ Employee added: 3, Sara, Analyst, ₹55000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] 📋 Traversing Employee List: [Time Complexity: O(n)]

[2025-06-21 20:44:06] ID: 1, Name: Alekya, Position: Developer, Salary: ₹65000

[2025-06-21 20:44:06] ID: 2, Name: Ravi, Position: Manager, Salary: ₹85000

[2025-06-21 20:44:06] ID: 3, Name: Sara, Position: Analyst, Salary: ₹55000

[2025-06-21 20:44:06] ✅ Traversal Complete. Duration: 0 ms

[2025-06-21 20:44:06] 🔍 Found Employee ID 2: Ravi, Manager, ₹85000 [Time Complexity: O(n), Duration: 0 ms]

[2025-06-21 20:44:06] 🗑 Deleted Employee → ID: 2, Name: Ravi [Time Complexity: O(n), Duration: 0 ms]

[2025-06-21 20:44:06] 📋 Traversing Employee List: [Time Complexity: O(n)]

[2025-06-21 20:44:06] ID: 1, Name: Alekya, Position: Developer, Salary: ₹65000

[2025-06-21 20:44:06] ID: 3, Name: Sara, Position: Analyst, Salary: ₹55000

[2025-06-21 20:44:06] ✅ Traversal Complete. Duration: 0 ms

**Time Complexity :-**

**Add Employee to Array:**

* **Best Case:** O(1)
* **Average Case:** O(1)
* **Worst Case:** O(1)

**Search Employee by ID (Linear Search):**

* **Best Case:** O(1)
* **Average Case:** O(n)
* **Worst Case:** O(n)

**Delete Employee by ID (Search + Shift):**

* **Best Case:** O(1)
* **Average Case:** O(n)
* **Worst Case:** O(n)

**Traverse All Employees:**

* **Best Case:** O(n)
* **Average Case:** O(n)
* **Worst Case:** O(n)