

Exercise 1:

```
1 using System;
2 using System.Collections.Generic;
3
4 public class Product
5 {
6     public int ProductId { get; set; }
7     public string ProductName { get; set; }
8     public int Quantity { get; set; }
9     public double Price { get; set; }
10
11     public Product(int id, string name, int qty, double price)
12     {
13         ProductId = id;
14         ProductName = name;
15         Quantity = qty;
16         Price = price;
17     }
18
19     public override string ToString()
20     {
21         return $"ID: {ProductId}, Name: {ProductName}, Qty: {Quantity}, Price: {Price}";
22     }
23 }
24
25 public class Inventory
26 {
27     private Dictionary<int, Product> products = new();
28
29     public void AddProduct(Product product)
30     {
31         if (products.ContainsKey(product.ProductId))
32             products[product.ProductId] = product;
33     }
34
35     public void UpdateProduct(int id, int qty, double price)
36     {
37         if (products.ContainsKey(id))
38         {
39             products[id].Quantity = qty;
40             products[id].Price = price;
41         }
42     }
43
44     public void DeleteProduct(int id)
45     {
46         products.Remove(id);
47     }
48
49     public void PrintInventory()
50     {
51         foreach (var p in products.Values)
52             Console.WriteLine(p);
53     }
54 }
55
56 public class Program
57 {
58     public static void Main()
59     {
60         Inventory inventory = new();
61         inventory.AddProduct(new Product(1, "Monitor", 10, 1200));
62         inventory.AddProduct(new Product(2, "Mouse", 50, 50));
63         inventory.UpdateProduct(1, 5, 1150);
64         inventory.DeleteProduct(2);
65         inventory.PrintInventory();
66     }
67 }
```

ID: 1, Name: Monitor, Qty: 5, Price: 1150
--- Code Execution Successful ---

Exercise 2:

```
1 using System;
2
3 public class Product
4 {
5     public int ProductId;
6     public string ProductName;
7     public string Category;
8
9     public Product(int id, string name, string category)
10    {
11        ProductId = id;
12        ProductName = name;
13        Category = category;
14    }
15 }
16
17 public class SearchSystem
18 {
19     public static int LinearSearch(Product[] products, string name)
20     {
21         for (int i = 0; i < products.Length; i++)
22             if (products[i].ProductName == name) return i;
23         return -1;
24     }
25
26     public static int BinarySearch(Product[] products, string name)
27     {
28         int low = 0, high = products.Length - 1;
29         while (low <= high)
30         {
31             int mid = (low + high) / 2;
32             int cmp = string.Compare(products[mid].ProductName, name);
33             if (cmp == 0) return mid;
34             else if (cmp < 0) low = mid + 1;
35             else high = mid - 1;
36         }
37         return -1;
38     }
39 }
40
41 public class Program
42 {
43     public static void Main()
44     {
45         1
46         1
47         --- Code Execution Successful ---
```

Exercise 3:

```
1 using System;
2
3 public class Order
4 {
5     public int OrderId;
6     public string CustomerName;
7     public double TotalPrice;
8
9     public Order(int id, string name, double price)
10    {
11        OrderId = id;
12        CustomerName = name;
13        TotalPrice = price;
14    }
15
16    public override string ToString()
17    {
18        return $"Order {OrderId}, {CustomerName}, {TotalPrice}";
19    }
20 }
21
22 public class Sorter
23 {
24     public static void BubbleSort(Order[] orders)
25     {
26         int n = orders.Length;
27         for (int i = 0; i < n - 1; i++)
28             for (int j = 0; j < n - i - 1; j++)
29                 if (orders[j].TotalPrice > orders[j + 1].TotalPrice)
30                     (orders[j], orders[j + 1]) = (orders[j + 1], orders[j]);
31     }
32
33     public static void QuickSort(Order[] arr, int low, int high)
34     {
35         if (low < high)
36         {
37             int p = Partition(arr, low, high);
38             QuickSort(arr, low, p - 1);
39             QuickSort(arr, p + 1, high);
40         }
41     }
42
43     private static int Partition(Order[] arr, int low, int high)
44     {
45         double pivot = arr[high].TotalPrice;
46         int i = low - 1;
47         for (int j = low; j < high; j++)
48             if (arr[j].TotalPrice < pivot)
49                 (arr[i + 1], arr[j]) = (arr[j], arr[i + 1]);
50         (arr[i + 1], arr[high]) = (arr[high], arr[i + 1]);
51         return i + 1;
52     }
53 }
54
55 public class Program
56 {
57     public static void Main()
58     {
59         Order[] orders = {
60             new Order(1, "Alice", 2000),
61             new Order(2, "Bob", 3000),
62             new Order(3, "Charlie", 1000)
63         };
64
65         Sorter.BubbleSort(orders, 0, orders.Length - 1); // Sort using Bubble Sort
66
67         foreach (var o in orders)
68             Console.WriteLine(o);
69     }
70 }
```

Order 2, Bob, 3000
Order 3, Charlie, 1000
Order 1, Alice, 2000

--- Code Execution Successful ---

Exercise 4:

```
1 using System;
2
3 public class Employee
4 {
5     public int Id;
6     public string Name;
7     public string Position;
8     public double Salary;
9
10    public Employee(int id, string name, string pos, double salary)
11    {
12        Id = id;
13        Name = name;
14        Position = pos;
15        Salary = salary;
16    }
17
18    public override string ToString() => $"Id: {Id} - Name: {Name} - Position: {Position} - Salary: {Salary}";
19 }
20
21 public class Company
22 {
23     private Employee[] employees = new Employee[10];
24     private int count = 0;
25
26     public void AddEmployee(Employee e) -- employees[count++] = e;
27
28     public void Traverse()
29     {
30         for (int i = 0; i < count; i++)
31             Console.WriteLine(employees[i]);
32     }
33
34     public Employee Search(int id)
35     {
36         for (int i = 0; i < count; i++)
37             if (employees[i].Id == id)
38                 return employees[i];
39         return null;
40     }
41
42     public void Delete(int id)
43     {
44         for (int i = 0; i < count; i++)
45         {
46             if (employees[i].Id == id)
47             {
48                 for (int j = i; j < count - 1; j++)
49                     employees[j] = employees[j + 1];
50                 count--;
51                 break;
52             }
53         }
54     }
55 }
56
57 public class Program
58 {
59     public static void Main()
60     {
61         Company c = new();
62         c.AddEmployee(new Employee(1, "John", "Manager", 8000));
63         c.AddEmployee(new Employee(2, "Jane", "Dev", 6000));
64         c.Delete(1);
65         c.Traverse();
66     }
67 }
```

Id: 2 - Name: Bob - Position: Manager - Salary: 3000

--- Code Execution Successful ---

Exercise 5:

using System;

```
public class Task
{
    public int TaskId;
    public string TaskName;
    public string Status;
    public Task Next;

    public Task(int id, string name, string status)
    {
        TaskId = id;
        TaskName = name;
        Status = status;
        Next = null;
    }

    public override string ToString() => $"ID: {TaskId}, Name: {TaskName}, Status: {Status}";
}

public class TaskLinkedList
{
    private Task head;

    public void AddTask(Task task)
    {
        if (head == null)
        {
            head = task;
        }
        else
        {
            Task temp = head;
            while (temp.Next != null)
            {
                temp = temp.Next;
            }
            temp.Next = task;
        }
    }

    public Task SearchTask(int id)
    {
        Task temp = head;
        while (temp != null)
        {
            if (temp.TaskId == id)
            {
                return temp;
            }
            temp = temp.Next;
        }
        return null;
    }

    public void TraverseTasks()
    {
        Task temp = head;
        while (temp != null)
        {
            Console.WriteLine(temp.ToString());
            temp = temp.Next;
        }
    }
}
```

```

        Console.WriteLine(temp);
        temp = temp.Next;
    }
}

public void DeleteTask(int id)
{
    if (head == null) return;

    if (head.TaskId == id)
    {
        head = head.Next;
        return;
    }

    Task temp = head;
    while (temp.Next != null && temp.Next.TaskId != id)
        temp = temp.Next;

    if (temp.Next != null)
        temp.Next = temp.Next.Next;
}

}

public class Program
{
    public static void Main()
    {
        TaskLinkedList list = new();
        list.AddTask(new Task(1, "Design UI", "Pending"));
        list.AddTask(new Task(2, "Fix Bugs", "In Progress"));
        list.TraverseTasks();

        Console.WriteLine("After Deleting Task 1:");
        list.DeleteTask(1);
        list.TraverseTasks();
    }
}

```

Output

```

ID: 1, Name: Design UI, Status: Pending
ID: 2, Name: Fix Bugs, Status: In Progress
After Deleting Task 1:
ID: 2, Name: Fix Bugs, Status: In Progress

=== Code Execution Successful ===

```

Exercise 6:

```

1 using System;
2
3 public class Book
4 {
5     public int BookId;
6     public string Title;
7     public string Author;
8
9     public Book(int id, string title, string author)
10    {
11        BookId = id;
12        Title = title;
13        Author = author;
14    }
15
16    public override string ToString() => $"BookId: {BookId} - Title: {Author}";
17 }
18
19 public class Library
20 {
21     public static int LinearSearch(Book[] books, string title)
22     {
23         for (int i = 0; i < books.Length; i++)
24             if (books[i].Title == title)
25                 return i;
26         return -1;
27     }
28
29     public static int BinarySearch(Book[] books, string title)
30     {
31         int low = 0, high = books.Length - 1;
32         while (low <= high)
33         {
34             int mid = (low + high) / 2;
35             int cmp = String.Compare(books[mid].Title, title, StringComparison.Ordinal);
36             if (cmp < 0)
37                 low = mid + 1;
38             else if (cmp > 0)
39                 high = mid - 1;
40             else
41                 return mid;
42         }
43         return -1;
44     }
45 }
46
47 public class Program
48 {
49     public static void Main()
50     {
51         Book[] books = {
52             new Book(1, "Programming", "Gerard Attacher"),
53             new Book(101, "Java Basics", "Janet Haining"),
54             new Book(102, "Python Crash Course", "Eric Matthes")
55         };
56
57         Array.Sort(books, (a, b) => a.Title.CompareTo(b.Title)); // for binary search
58
59         int low = 0, high = books.Length - 1; // 0 to 2
60         int mid = 1; // 1
61
62         Console.WriteLine($"Linear Search Found at: {Library.LinearSearch(books, title)}");
63         Console.WriteLine($"Binary Search Found at: {Library.BinarySearch(books, title)}");
64     }
65 }

```

Linear Search Found at: 1
Binary Search Found at: 1

Code Execution Successful ==>

Exercise 7:

```

1 using System;
2
3 public class Forecasting
4 {
5     // Simple recursive growth: Future = current * (1 + rate)^years
6     public static double PredictFutureValue(double value, double rate, int years)
7     {
8         if (years == 0) return value;
9         return (1 + rate) * PredictFutureValue(value, rate, years - 1);
10    }
11
12    // Optimized version with memoization (optional)
13    public static double PredictWithMemo(double value, double rate, int years, double[] memo)
14    {
15        if (years == 0) return value;
16        if (memo[years] != 0) return memo[years];
17        return memo[years] = (1 + rate) * PredictWithMemo(value, rate, years - 1, memo);
18    }
19 }
20
21 public class Program
22 {
23     public static void Main()
24     {
25         double current = 1000; // 1000
26         double rate = 0.1; // 10%
27         int years = 5;
28
29         Console.WriteLine("Recursive Forecast: " +
30             Forecasting.PredictFutureValue(current, rate, years));
31
32         double[] memo = new double[years + 1];
33         Console.WriteLine("Memoized Forecast: " +
34             Forecasting.PredictWithMemo(current, rate, years, memo));
35     }
36 }
37

```

Recursive Forecast: 1610.51
Memoized Forecast: 1610.51

Code Execution Successful ==>