# Module 4

### What to learn

Collection
Array
Stack Queue
Generic Collection
List
Stack
Queue
Dictionary (map key value pair)
Using Generic collection Interface (ICollection Interface)
Operation on collection sort, search, Index, Add, count remove.

### Practice Exercise

**Practice 1**
Store a product information in map object. Key will be a product item and value will be the price of that product. Search the product by product name.
**Practice 2**

## Collections

Write a program to find duplicate elements in a collection and return them along with their frequency.
Implement a method to merge two collections of different types into a single collection of key-value pairs.
Create a collection of objects representing employees and group them by their department using LINQ.
Implement a solution to filter out items from a collection based on multiple conditions (e.g., price > 100 and category = "Electronics").
Write a function to flatten a collection of nested collections into a single collection.

**Practice 3**

## Array

Implement a method to find the subarray with the maximum sum from a given array.

Write a program to rotate an array by k positions to the right without using extra space.
Create a solution to find all unique triplets in an array that sum to a specific value.
Develop a function that takes an array of integers and reorders it such that even numbers come before odd numbers.
Write a method to determine if two arrays are permutations of each other.

Practice 4

## Stack and Queue

Using a stack, write a function to check if a given string of brackets is balanced.
Design a queue implementation where the maximum element can be retrieved in O(1) time.
Simulate a browser's forward and backward navigation functionality using two stacks.
Write a program to reverse the order of elements in a queue without using additional queues.
Implement a priority queue to process customer orders based on urgency levels.

Practice 5

## Generic Collections

Implement a generic method to merge two collections into one while removing duplicates.
Create a generic method to find the median of a collection of numeric types.
Write a program to implement a custom generic collection that supports filtering, sorting, and mapping.
Design a generic collection to handle caching with time-to-live (TTL) for each entry.
Implement a generic method to shuffle the elements of a collection randomly.

Practice 6

## List

Write a method to find and remove all duplicate elements from a list without using LINQ.
Develop a function to split a list into sublists of a given size.
Create a program to merge two sorted lists into one sorted list.
Write a method to shift elements of a list cyclically to the left by a given number of positions.

Implement a function to find the intersection of two lists without using built-in methods.

Practice 7

## Stack

Write a program to sort a stack using only another stack and no additional data structures.
Create a method to evaluate a postfix mathematical expression using a stack.
Design a stack that supports retrieving the minimum element in O(1) time.
Implement a function to reverse a string using a stack.
Write a stack-based solution to convert an infix expression to a postfix expression.

Practice 8

## Queue

Implement a circular queue that supports dynamic resizing.
Create a method to merge two sorted queues into a single sorted queue.
Write a program to simulate a ticket booking system using a queue, prioritizing VIP customers.
Develop a solution to reverse the first k elements of a queue without altering the rest.
Design a queue-based implementation for task scheduling with priorities.

Practice 9

## Dictionary (Map Key-Value Pair)

Implement a method to find the most frequently occurring value in a dictionary.
Write a function to merge two dictionaries, combining values of common keys into a list.
Create a program to group words by their lengths using a dictionary.
Design a dictionary to store student grades and calculate the average grade for each student.
Write a method to find all keys in a dictionary whose values satisfy a specific condition.

Practice 10

## Using Generic Collection Interfaces (ICollection)

Implement a method using ICollection to remove items from a collection based on a predicate.
Write a function that accepts an ICollection and returns the second largest element.

Create a generic method using ICollection to compute the union of two collections.
Develop a method to clone an ICollection into a new collection of the same type.
Write a function using ICollection to implement pagination for large data sets.

Practice 11

## Operations on Collection (Sort, Search, Index, Add, Count, Remove)

Write a method to sort a collection of employees by multiple fields (e.g., age and then name).
Create a function to search for an element in a sorted collection using binary search.
Implement a method to insert an item at a specific index in a collection and maintain sorted order.
Write a solution to count the occurrences of each unique element in a collection.
Develop a method to remove items from a collection that are not present in another collection.

### Assignment Exercise

**Assignment 1**

Rambo Rental Bikes is looking for developing a system to calculate the rentals of the bikes. System should accept the customer details, bike details and calculate the rental charges. DESCRIPTION OF PROJECTS System allows users to add customer details with bike rented. It computes rent for each customer. Systems displays the Bike details with summation of days of hire and rental payment. FUNCTIONALITY AND TASK Define a class called Mobike with the following description: Instance variables/data members: BikeNumber – to store the bike's number PhoneNumber – to store the phone number of the customer Name – to store the name of the customer Days – to store the number of days the bike is taken on rent o charge – to calculate and store the rental charge Member methods: void Input( ) – to input and store the detail of the customer. void Compute( ) – to compute the rental charge void display( ) – to display the details in the following format: Bike No. PhoneNo No. of days Charge The rent for a mobike is charged on the following basis: First five days Rs 500 per day Next five days Rs 400 per day Rest of the days Rs 200 per day Use collection Framework to store 10 Customer Details. Implement List operation add, delete, edit and search functionality

**Assignment 2**

# Assignment: Comprehensive Data Structures and Collections in C#

**Objective**:
The goal of this assignment is to apply and integrate concepts from **Collections**, **Arrays**, **Stacks**, **Queues**, **Generic Collections**, and **Dictionaries** to solve real-world problems. Each task focuses on specific functionality and should be completed in 3–4 hours.

## Instructions:

Use **C#** for implementing the solutions.
Each question represents a task that builds towards a unified application.
Submit your code along with brief explanations of your logic for each task.
Make sure your code is modular and follows best practices.

# Scenario:

You are developing a **library management system** that involves various data structures to manage books, users, and borrowing records. Implement the following tasks as part of the system:

# Tasks

## Task 1: Manage Book Collection (Collections)

Create a collection to store book details (Book ID, Title, Author, and Availability).
Implement methods to:

- Add a new book.
- Find books by a specific author.
- Remove a book by its ID.

## Task 2: Efficient Search in Book List (Array)

Use an array to store book IDs (integer values).
Implement a function to:

- Find the index of a given Book ID using **binary search**.
- Check if the array is sorted; if not, sort it before performing the binary search.

## Task 3: Book Borrowing (Stack and Queue)

**Stack**: Implement a "Recent Borrowed Books" feature to track the last 5 books borrowed by any user. Use a stack to store and display them in reverse order of borrowing.
**Queue**: Create a waiting queue for books that are currently unavailable.
Implement methods to:

Add a user to the waiting queue.
Serve the next user in line when the book becomes available.

## Task 4: Generic Collection for Borrowing Records (Generic Collections)

Implement a **generic class** BorrowingRecord<T> to store borrowing details for any entity type (e.g., books, journals).
Add methods to:

Record a new borrowing transaction.
Retrieve all borrowings for a specific user.

## Task 5: User Management (List)

Use a list to manage user details (User ID, Name, and Borrow Count).
Implement methods to:

Find the user with the maximum number of borrowings.
Remove users with zero borrowings from the list.

## Task 6: Recently Returned Books (Stack)

Create a stack to store the details of books returned recently.
Implement a method to transfer all books from this stack to the "Available Books" collection when the stack reaches 10 items.

## Task 7: Track User Reservations (Queue)

Implement a queue-based system to manage user reservations for books.
Provide methods to:

Add a reservation for a user.
Remove a reservation once it has been fulfilled.

## Task 8: Book Search by Keywords (Dictionary)

Use a dictionary where the key is a keyword (e.g., "science", "history"), and the value is a list of book titles.
Implement methods to:

Add a new keyword and its associated books.
Search for all books under a specific keyword.

## Task 9: Collection Operations (Sort, Search, Remove)

Implement methods to:

Sort the book collection alphabetically by title.
Search for a book by title using a linear search.
Remove books that haven't been borrowed in the last year.

## Task 10: Integrating Everything (Final Application)

Library Management System
---

1. Add a Book
2. Search Book by Author
3. Borrow a Book
4. Return a Book
5. View Borrowing Records
6. Manage User Accounts
7. View Waiting Queue
8. Search Books by Keyword
9. Sort Book Collection
10. Exit

Online Reference

No online Reference

## .NET Core Web API

## WEB API (old)

## Authentication And Authorization (WEBAPI)(old)

## FullStackDevelopment_WIth_Dotnet_AND_Angular