#### Questions for this assignment

Scenario: You need to implement a collection that ensures uniqueness of elements and provides fast look-up operations.

Question: When would you use HashSet over other collection classes such as List or ArrayList in C#? What are the main advantages of HashSet?

Scenario: You need to implement a collection that supports Last-In-First-Out (LIFO) or First-In-First-Out (FIFO) operations.

Question: When would you use Stack or Queue over other collection classes such as List or ArrayList in C#? What are the main differences between Stack and Queue?

Scenario: You need to search for an element in a collection based on its value or a specific condition.

Question: What is the difference between IndexOf() and BinarySearch() methods in List class in C#? When would you use each of them?

Scenario: You need to insert an element into a specific position in a List or insert a range of elements into a specific position in a List.

Question: What are the differences between Insert() and InsertRange() methods in the List class in C#? When would you use each of them?

Scenario: You need to find elements in a List based on a specific condition.

Question: What are the differences between Exists(), Find(), and FindAll() methods in the List class in C#? When would you use each of them?

Scenario: You need to implement custom sorting logic for a collection of objects.

Question: When would you use IComparable or IComparer in C# to implement custom sorting logic? What are the differences between them?

Scenario: You need to implement a collection that can be enumerated and modified.

Question: When would you use IEnumerable<T> or ICollection in C# to create a custom collection? What are the differences between them?

Scenario: You need to remove elements from a collection based on certain conditions.

Question: What are the differences between Remove(), RemoveRange(), and RemoveAll() methods in the List class in C#?

1. Scenario: You need to implement a collection that ensures uniqueness of elements and provides fast look-up operations.

Question: When would you use HashSet over other collection classes such as List or ArrayList in C#? What are the main advantages of HashSet?

HashSet is a collection class in C# that stores unique elements and provides fast look-up operations. You would use HashSet when you need to ensure that the elements in the collection are unique and you need fast access to elements based on their values. HashSet uses hash codes and equality comparisons to ensure the uniqueness of elements, which provides fast look-up performance compared to List or ArrayList.

The main advantages of HashSet are:

* **Ensures uniqueness of elements:** HashSet ensures that each element in the collection is unique based on its value, and it automatically handles duplicate elements for you.
* **Fast look-up operations:** HashSet uses hash codes to quickly locate elements in the collection, providing fast performance for look-up operations.
* **No duplicates:** HashSet automatically handles duplicate elements and ensures that each element is stored only once, saving memory and improving performance.
* **Dynamic resizing:** HashSet automatically resizes itself when the number of elements exceeds its capacity, ensuring efficient memory usage.
* **Efficient operations:** HashSet provides efficient operations for set operations such as union, intersection, and difference, making it suitable for various set-based scenarios.

Scenario: You need to implement a collection that supports Last-In-First-Out (LIFO) or First-In-First-Out (FIFO) operations.

Question: When would you use Stack or Queue over other collection classes such as List or ArrayList in C#? What are the main differences between Stack and Queue?

Stack and Queue are collection classes in C# that provide support for Last-In-First-Out (LIFO) and First-In-First-Out (FIFO) operations, respectively. You would use Stack when you need to implement a collection that follows the Last-In-First-Out (LIFO) principle, where the last element added to the collection is the first one to be removed. You would use Queue when you need to implement a collection that follows the First-In-First-Out (FIFO) principle, where the first element added to the collection is the first one to be removed.

The main differences between Stack and Queue are:

* **Order of elements:** In Stack, the order of elements is based on the Last-In-First-Out (LIFO) principle, whereas in Queue, the order of elements is based on the First-In-First-Out (FIFO) principle.
* **Operations:** Stack provides methods such as Push() to add elements to the top of the stack and Pop() to remove elements from the top of the stack. Queue provides methods such as Enqueue() to add elements to the end of the queue and Dequeue() to remove elements from the front of the queue.
* **Access to elements:** Stack provides access to the top element of the stack using the Peek() method without removing it, whereas Queue provides access to the front element of the queue using the Peek() method without removing it.
* **Performance:** Stack and Queue are optimized for their respective operations, with Stack providing fast performance for LIFO operations and Queue providing fast performance for FIFO operations.

Scenario: You need to search for an element in a collection based on its value or a specific condition.

Question: What is the difference between IndexOf() and BinarySearch() methods in List class in C#? When would you use each of them?

IndexOf() and BinarySearch() are methods in the List class in C# that are used to search for an element in a collection based on its value or a specific condition.

The main differences between them are:

* **Search algorithm:** IndexOf() uses a linear search algorithm that iterates over the elements in the collection one by one to find the matching element, whereas BinarySearch() uses a binary search algorithm that requires the collection to be sorted and can search for the element in a more efficient way by dividing the search range in half at each step.
* **Performance:** BinarySearch() is generally faster than IndexOf() for large collections because of its more efficient search algorithm. However, for small collections or collections that are not sorted, IndexOf() may perform equally well or even better.
* **Sorting requirement:** BinarySearch() requires the collection to be sorted in ascending order according to the default comparer or a custom comparer provided, whereas IndexOf() does not require any sorting.
* **Return value:** IndexOf() returns the index of the first occurrence of the matching element in the collection, or -1 if the element is not found, whereas BinarySearch() returns the index of the matching element in the collection if found, or a bitwise complement of the index of the next larger element if not found. This behavior of BinarySearch() can be confusing and needs to be handled carefully in the code.

Scenario: You need to insert an element into a specific position in a List or insert a range of elements into a specific position in a List.

Question: What are the differences between Insert() and InsertRange() methods in the List class in C#? When would you use each of them?

Insert() and InsertRange() are methods in the List class in C# that are used to insert an element or a range of elements into a specific position in a collection.

The differences between them are:

* **Insert():** This method inserts an element at the specified index in the collection. It takes two arguments, the index at which the element should be inserted, and the value of the element to be inserted. The existing elements from the specified index to the end of the collection are shifted to the right to make room for the new element.
* **InsertRange():** This method inserts a range of elements at the specified index in the collection. It takes two arguments, the index at which the range of elements should be inserted, and a collection of elements to be inserted. The existing elements from the specified index to the end of the collection are shifted to the right to make room for the new elements.

You would use Insert() when you want to insert a single element into a specific position in the collection.

You would use InsertRange() when you want to insert a range of elements into a specific position in the collection, for example, when you have a collection of elements that you want to insert into another collection at a specific index.

Scenario: You need to find elements in a List based on a specific condition.

Question: What are the differences between Exists(), Find(), and FindAll() methods in the List class in C#? When would you use each of them?

Exists(), Find(), and FindAll() are methods in the List class in C that are used to find elements in a collection based on a specific condition.

The differences between them are:

* **Exists():** This method checks if any element in the collection satisfies a specific condition. It takes a predicate as an argument, which is a delegate that represents the condition to be checked, and returns a boolean value indicating whether any element satisfies the condition or not.
* **Find():** This method returns the first element in the collection that satisfies a specific condition. It takes a predicate as an argument, which is a delegate that represents the condition to be checked, and returns the first element that satisfies the condition. If no element satisfies the condition, it returns the default value of the element type.
* **FindAll():** This method returns all elements in the collection that satisfy a specific condition. It takes a predicate as an argument, which is a delegate that represents the condition to be checked, and returns a new List containing all the elements that satisfy the condition.

You would use Exists() when you only need to check if any element satisfies a specific condition and you are only interested in a boolean result.

You would use Find() when you need to find the first element that satisfies a specific condition and you are only interested in a single element.

You would use FindAll() when you need to find all elements that satisfy a specific condition and you want to retrieve them as a new List.

Scenario: You need to implement custom sorting logic for a collection of objects.

Question: When would you use IComparable or IComparer in C# to implement custom sorting logic? What are the differences between them?

IComparable and IComparer are interfaces in C# that are used to implement custom sorting logic for a collection of objects.

The differences between them are:

* **IComparable:** This interface is implemented by objects that can be compared to other objects of the same type. It defines a CompareTo() method that compares the current object with another object and returns an integer value indicating the relative order of the objects. It is typically used when the natural order of objects can be determined based on their intrinsic properties, such as comparing numbers or strings based on their values.
* **IComparer:** This interface is implemented by objects that can compare two other objects of different types or instances of the same type that do not implement IComparable. It defines a Compare() method that takes two objects as arguments and returns an integer value indicating the relative order of the objects. It is typically used when you need to implement custom sorting logic for objects that do not have a natural order or when you want to override the default comparison behavior of objects that implement IComparable.

You would use IComparable when the objects in your collection have a natural order that can be determined based on their intrinsic properties, and you want to implement sorting logic based on that natural order.

You would use IComparer when you need to implement custom sorting logic for objects that do not have a natural order, or when you want to override the default comparison behavior of objects.

Scenario: You need to implement a collection that can be enumerated and modified.

Question: When would you use IEnumerable<T> or ICollection in C# to create a custom collection? What are the differences between them?

IEnumerable<T> and ICollection are interfaces in C# that are used to create custom collections that can be enumerated and modified.

The differences between them are:

* **IEnumerable<T>:** This interface represents a collection of objects that can be enumerated, which means you can iterate over the objects in the collection using a foreach loop or other enumeration methods. It defines a single method GetEnumerator() that returns an `IEnumerator<T GetEnumerator() that allows you to iterate over the elements in the collection.
* **ICollection:** This interface represents a collection of objects that can be enumerated and modified, which means you can both iterate over the objects in the collection and perform modifications such as adding, removing, or updating elements. It inherits from IEnumerable<T> and adds additional methods for modifying the collection, such as Add(), Remove(), Clear(), etc.

You would use IEnumerable<T> when you want to create a read-only collection that can be enumerated, but you do not need to modify the collection.

You would use ICollection when you need to create a collection that can be enumerated and modified, allowing you to perform operations such as adding, removing, or updating elements in the collection.

Scenario: You need to remove elements from a collection based on certain conditions.

Question: What are the differences between Remove(), RemoveRange(), and RemoveAll() methods in the List class in C#?

Remove(), RemoveRange(), and RemoveAll() are methods in the List class in C# that are used to remove elements from a collection based on certain conditions, but they have different behaviors:

* **Remove():** This method removes the first occurrence of a specific element from the collection and returns a boolean value indicating whether the element was removed or not. It removes only the first occurrence of the element, even if there are multiple occurrences of the same element in the collection.
* **RemoveRange():** This method removes a range of elements from the collection based on the specified starting index and count. It modifies the collection by removing the elements in the specified range and shifting the remaining elements to fill the gap. It returns void and does not provide any information about the elements that were removed.
* **RemoveAll():** This method removes all elements from the collection that satisfy a given condition, which is specified as a predicate function. It returns the number of elements that were removed from the collection.

You would use Remove() when you need to remove only the first occurrence of a specific element from the collection. You would use RemoveRange() when you need to remove a range of elements from the collection based on their indices. You would use RemoveAll() when you need to remove all elements from the collection that satisfy a given condition.