# Short Answer:

## Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples to each question).

1. Describe the Collections Type. Explain each Collection type in detail, especially pointing out the scenarios for using certain Collection type

Collections standardize the way of which the objects are handled by your program. In other words, it contains a set of classes to contain elements in a generalized manner. With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc.

Diagram

Description automatically generated

[Collections in C# - GeeksforGeeks](https://www.geeksforgeeks.org/collections-in-c-sharp/?ref=lbp)

● Group of objects

● It is not specified whether they are

○ Ordered / not ordered

○ Duplicated / not duplicated

● Following constructors are common to all classes

implementing Collection

○ T()

○ T(){...}

○ T(Collection c)

Collections

● Generic Collections

○ Generic Collections work on the specific type that is specified in the program whereas

non-generic collections work on the object type.

○ Using System.Collections.Generic;

○ List, LinkedList, Dictionary, HashSet, SortedList, Stack, Queue

● List

○ List class is a collection that can be used for specific types.

○ List is a class that is similar to an array, but the size is not fixed

○ Elements can be added / removed at runtime.

○ Ex. List<int> al = new List<int>();

● List

○ access

○ Remove()

○ RemoveAt()

○ Contains()

○ IndexOf()

○ LastIndexOf()

In C#, List is a generic collection which is used to store the elements or objects in the form of a list and it is defined under System.Collection.Generic namespace. It provides the same functionality like ArrayList, but there is only one difference i.e., a list is a generic whereas ArrayList is a non-generic collection. It is dynamic in nature means the size of the list grows, according to the need.

Important Points:

The List class implements the ICollection<T>, IEnumerable<T>, IList<T>, IReadOnlyCollection<T>, IReadOnlyList<T>, ICollection, IEnumerable, and IList interface.

It can accept null as a valid value for reference types and also allows duplicate elements.

If the Count becomes equal to Capacity, then the capacity of the List increased automatically by reallocating the internal array. The existing elements will be copied to the new array before the addition of the new element.

The elements present in the list are not sorted by default and elements are accessed by zero-based index.

How to create a List?

A List class has 3 constructors which are used to create a list and the constructors are as follows:

List<T>(): This constructor is used to create an instance of the List<T> class that is empty and has the default initial capacity.

List<T>(IEnumerable): This constructor is used to create an instance of the List<T> class that contains elements copied from the specified collection and has sufficient capacity to accommodate the number of elements copied.

List<T>(Int32): This constructor is used to create an instance of the List<T> class that is empty and has the specified initial capacity.

● LinkedList

○ a general-purpose linked list (doubly linked)

○ LinkedList<T> provides separate nodes of type LinkedListNode<T>, so

insertion and removal are O(1) operations

Text

Description automatically generated Diagram

Description automatically generated with medium confidence

A LinkedList is a linear data structure which stores element in the non-contiguous location. The elements in a linked list are linked with each other using pointers. Or in other words, LinkedList consists of nodes where each node contains a data field and a reference(link) to the next node in the list. In C#, LinkedList is the generic type of collection which is defined in System.Collections.Generic namespace. It is a doubly linked list, therefore, each node points forward to the Next node and backward to the Previous node. It is a dynamic collection which grows, according to the need of your program. It also provides fast inserting and removing elements.

Important Points:

The LinkedList class implements the ICollection<T>, IEnumerable<T>, IReadOnlyCollection<T>, ICollection, IEnumerable, IDeserializationCallback, and ISerializable interfaces.

It also supports enumerators.

You can remove nodes and reinsert them, either in the same list or in another list, which results in no additional objects allocated on the heap.

Every node in a LinkedList<T> object is of the type LinkedListNode<T>.

It does not support chaining, splitting, cycles, or other features that can leave the list in an inconsistent state.

If the LinkedList is empty, the First and Last properties contain null.

The capacity of a LinkedList is the number of elements the LinkedList can hold.

In LinkedList, it is allowed to store duplicate elements but of the same type.

[Linked List Implementation in C# - GeeksforGeeks](https://www.geeksforgeeks.org/linked-list-implementation-in-c-sharp/?ref=lbp)

In C#, Dictionary is a generic collection which is generally used to store key/value pairs. The working of Dictionary is quite similar to the non-generic hashtable. The advantage of Dictionary is, it is generic type. Dictionary is defined under System.Collections.Generic namespace. It is dynamic in nature means the size of the dictionary is grows according to the need.

Important Points:

The Dictionary class implements the

IDictionary<TKey,TValue> Interface

IReadOnlyCollection<KeyValuePair<TKey,TValue>> Interface

IReadOnlyDictionary<TKey,TValue> Interface

IDictionary Interface

In Dictionary, the key cannot be null, but value can be.

In Dictionary, key must be unique. Duplicate keys are not allowed if you try to use duplicate key then compiler will throw an exception.

In Dictionary, you can only store same types of elements.

The capacity of a Dictionary is the number of elements that Dictionary can hold.

[C# Dictionary with examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-dictionary-with-examples/)

● Dictionary

○ represents the items as a combination of a key and value

○ access the value based on the key

Text

Description automatically generated Diagram

Description automatically generated Diagram

Description automatically generated Diagram

Description automatically generated

● Dictionary

○ Quick initialization Graphical user interface, text, application

Description automatically generated

○ Methods

■ ContainsKey()

■ TryGetValue()

■ TryAdd()

How to create the Dictionary?

Dictionary class has 7 constructors which are used to create the Dictionary, here we only use Dictionary<TKey, TValue>() constructor and if you want to learn more about constructors then refer C# | Dictionary Class.

Dictionary<TKey, TValue>(): This constructor is used to create an instance of the Dictionary<TKey, TValue> class that is empty, has the default initial capacity, and uses the default equality comparer for the key type as follows:

Step 1: IncludeSystem.Collections.Generic namespace in your program with the help of using keyword.

Syntax:

using System.Collections.Generic;

Step 2: Create a Dictionary using Dictionary<TKey, TValue> class as shown below:

Dictionary dictionary\_name = new Dictionary();

Step 3: If you want to add elements in your Dictionary then use Add() method to add key/value pairs in your Dictionary. And you can also add key/value pair in the dictionary without using Add method. As shown in the below example.

Step 4: The key/value pair of the Dictionary is accessed using three different ways:

for loop: You can use for loop to access the key/value pairs of the Dictionary

Using Index: You can access individual key/value pair of the Dictionary by using its index value. Here, you just specify the key in the index to get the value from the given dictionary, no need to specify the index. Indexer always takes the key as a parameter, if the given key is not available in the dictionary, then it gives KeyNotFoundException.

foreach loop: You can use foreach loop to access the key/value pairs of the dictionary.As shown in the below example we access the Dictionary using a foreach loop.

[C# Dictionary with examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-dictionary-with-examples/?ref=lbp)

● HashSet

○ a collection that contains no duplicate elements, and whose

elements are in no particular order

Text

Description automatically generated

In C#, HashSet is an unordered collection of unique elements. This collection is introduced in .NET 3.5. It supports the implementation of sets and uses the hash table for storage. This collection is of the generic type collection and it is defined under System.Collections.Generic namespace. It is generally used when we want to prevent duplicate elements from being placed in the collection. The performance of the HashSet is much better in comparison to the list.

Important Points:

The HashSet class implements the ICollection, IEnumerable, IReadOnlyCollection, ISet, IEnumerable, IDeserializationCallback, and ISerializable interfaces.

In HashSet, the order of the element is not defined. You cannot sort the elements of HashSet.

In HashSet, the elements must be unique.

In HashSet, duplicate elements are not allowed.

Is provides many mathematical set operations, such as intersection, union, and difference.

The capacity of a HashSet is the number of elements it can hold.

A HashSet is a dynamic collection means the size of the HashSet is automatically increased when the new elements are added.

In HashSet, you can only store the same type of elements.

How to create a HashSet?

The HashSet class provides 7 different types of constructors which are used to create a HashSet, here we only use HashSet(), constructor. To read more about HashSet’s constructors you can refer to C# | HashSet Class.

HashSet(): It is used to create an instance of the HashSet class that is empty and uses the default equality comparer for the set type.

Step 1: Include System.Collections.Generic namespace in your program with the help of using keyword:

using System.Collections.Generic;

Step 2: Create a HashSet using the HashSet class as shown below:

HashSet<Type\_of\_hashset> Hashset\_name = new HashSet<Type\_of\_hashset>();

Step 3: If you want to add elements in your HashSet, then use Add() method to add elements in your HashSet. And you can also store elements in your HashSet using collection initializer.

Step 4: The elements of HashSet is accessed by using a foreach loop. As shown in the below example.

[HashSet in C# with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/hashset-in-c-sharp-with-examples/?ref=lbp)

● SortedList

○ represents a collection of key/value pairs that are sorted by key

based on the associated IComparer<T> implementation. Text

Description automatically generated

In C#, SortedList is a collection of key/value pairs which are sorted according to keys. By default, this collection sort the key/value pairs in ascending order. It is of both generic and non-generic type of collection. The generic SortedList is defined in System.Collections.Generic namespace whereas non-generic SortedList is defined under System.Collections namespace, here we will discuss non-generic type SortedList.

Important Points:

The SortedList class implements the IEnumerable, ICollection, IDictionary and ICloneable interfaces.

In SortedList, an element can be accessed by its key or by its index.

A SortedList object internally maintains two arrays to store the elements of the list, i.e, one array for the keys and another array for the associated values.

Here, a key cannot be null, but a value can be.

The capacity of a SortedList object is the number of key/value pairs it can hold.

In SortedList, duplicate keys are not allowed.

In SortedList, you can store values of the same type and of the different types due to the non-generic collection. If you use a generic SortedList in your program, then it is necessary that the type of the values should be the same.

In SortedList you cannot store keys of different data types in the same SortedList because the compiler will throw an exception. So, always add the key in your SortedList of the same type.

You can also cast key/value pair of SortedList into DictionaryEntry.

[C# SortedList with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-sortedlist-with-examples/?ref=lbp)

● Stack

○ It represents a last-in, first out

collection of object

A Stack represents a last-in, first-out collection of objects. It is used when you need last-in, first-out access to items. It is both a generic and non-generic type of collection. The generic stack is defined in System.Collections.Generic namespace whereas non-generic stack is defined under System.Collections namespace, here we will discuss non-generic type stack. A stack is used to create a dynamic collection that grows, according to the need of your program. In a stack, you can store elements of the same type or different types.

Important Points:

The Stack class implements the IEnumerable, ICollection, and ICloneable interfaces.

When you add an item in the list, it is called pushing the element.

When you remove it, it is called popping the element.

The capacity of a Stack is the number of elements the Stack can hold. As elements are added to a Stack, the capacity is automatically increased as required through reallocation.

In Stack, you are allowed to store duplicate elements.

A Stack accepts null as a valid value for reference types.

Diagram

Description automatically generated [C# Stack with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-stack-with-examples/?ref=lbp)

● Queue

○ It represents a first-in, first

out collection of object Text

Description automatically generated

A Queue is used to represent a first-in, first out(FIFO) collection of objects. It is used when you need first-in, first-out access of items. It is the non-generic type of collection which is defined in System.Collections namespace. It is used to create a dynamic collection which grows, according to the need of your program. In Queue, you can store elements of the same type and of the different types. Generally, a queue is helpful when you access that information in the same way in which they stored in the collection and it is temporary storage to store data.

Important Points:

The Queue class implements the IEnumerable, ICollection, and ICloneable interfaces.

When you add an item in the list, it is called enqueue.

when you remove an item, it is called dequeue.

Queue accepts null as a valid value for reference types.

As elements are added to a Queue, the capacity is automatically increased as required by reallocating the internal array.

In Queue, you are allowed to store duplicate elements.

The capacity of a Queue is the number of elements the Queue can hold. Diagram

Description automatically generated [C# Queue with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-queue-with-examples/?ref=lbp)

● Non-generic Collections

○ In non-generic collections, each element can represent a value of a different type. The collection

size is not fixed. Items from the collection can be added or removed at runtime

○ Using System.Collections;

○ ArrayList, HashTable, SortedList, Stack, Queue

● ArrayList

○ ArrayList class is a collection that can be used for any types or objects.

■ Arraylist is a class that is similar to an array, but it can be used to store values of various

types.

■ An Arraylist doesn't have a specific size.

■ Any number of elements can be stored.

■ Ex. ArrayList al = new ArrayList();

● HashTable

○ represents the items as a combination of a key and value Graphical user interface, text, application, chat or text message

Description automatically generated

● SortedList

○ is a class that has the combination of arraylist and hashtable.

○ represents a collection of key/value pairs that are sorted by key and are

accessible by key and by index

Graphical user interface, text, application

Description automatically generated

● Stack

○ It represents a last-in, first out collection of object

○ It is used when you need a last-in, first-out access of items. When you add

an item in the list, it is called pushing the item and when you remove it, it is

called popping the item

● Queue

○ It represents a first-in, first out collection of object

○ It is used when you need a first-in, first-out access of items. When you add

an item in the list, it is called enqueue and when you remove it, it is called

deque

Text

Description automatically generated with medium confidence

[C# Generic & Non-generic Collections (tutorialsteacher.com)](https://www.tutorialsteacher.com/csharp/csharp-collection#:~:text=Non-generic%20Collections%20%20%20%20Non-generic%20Collections%20,style%20%28F%20...%20%202%20more%20rows%20)

1. What’s the difference between Generic Collections and Non-Generic Collection?

Generic Collections. Generic Collections work on the specific type that is specified in the program whereas non-generic collections work on the object type. Elements can be added / removed at runtime.

[What is the difference between generic and non-generic collections in C#? – Find what come to your mind (getperfectanswers.com)](https://getperfectanswers.com/what-is-the-difference-between-generic-and-non-generic-collections-in-c/#:~:text=2.%20Generic%20Collections.%20Generic%20Collections%20work%20on%20the,whereas%20non-generic%20collections%20work%20on%20the%20object%20type.)

1. What is IEnumerable? What is IList?

IEnumerable provides only minimal "iterable" functionality. You can traverse the sequence, but that's about it.

This has disadvantages; for example, it is very inefficient to count elements using IEnumerable, or to get the nth element.

But it has advantages too; for example, an IEnumerable could be an endless sequence, like the sequence of primes.

Array is a fixed-size collection with random access (i.e. you can index into it).

List is a variable-size collection (i.e. you can add and remove elements) with random access.

IList is an interface which abstracts list functionality (count, add, remove, indexer access) away from the various concrete classes such as List, BindingList, ObservableCollection, etc.

IList inherits from IEnumerable so they are essentially the same just IList has extra functionality.

It depends on what you are doing with that data once its created as to whether you should use an IEnumerable or IList.

IEnumerable is good if you only want to iterate over the collections content as its readonly.

IList allows adding and removing of content (so does ICollection) and allows direct access to elements with an index

IEnumerable is an interface that allows the iteration through a collection of items (e.g. via the foreach keyword).

An array is a .NET intrinsic. It holds items of the same type, but it is of a fixed size. Once you create an array with x elements, it cannot grow or shrink.

IList defines the interface for a list, and also implements IEnumerable.

List implements the IList interface; it is a concrete type of list.

The difference between .NET Lists and arrays is that lists can have elements added to them -- they grow to be big enough to hold all of the required items. The list stores this internally in an array and, when the array is no longer big enough to hold all of the elements, a new array is created and the items copied across.

IList & arrays both implement IEnumerable. That's how interfaces work -- classes implement the contract and behave in a similar fashion and can be treated similarly as a result (you know that the class implements IEnumerable, you don't need to know the hows or the whys). I suggest you read up on interfaces and so forth.

[c# - What's the difference between IEnumerable and Array, IList and List? - Stack Overflow](https://stackoverflow.com/questions/764748/whats-the-difference-between-ienumerable-and-array-ilist-and-list)

1. Explain the time complexity of List, LinkedList, and Dictionary on adding/removing/updating/deleting

Table

Description automatically generated Graphical user interface, text, application

Description automatically generated O（1）

[Doubly Linked List in C# (c-sharpcorner.com)](https://www.c-sharpcorner.com/article/doubly-linked-list-and-circular-linked-list-in-c-sharp/)

Dictionary： Add O（1）

Table

Description automatically generated

1. What are the differences between Dictionary and SortedList?

In C#, SortedList is a collection of key/value pairs which are sorted according to keys. By default, this collection sort the key/value pairs in ascending order. It is of both generic and non-generic type of collection. The generic SortedList is defined in System.Collections.Generic namespace whereas non-generic SortedList is defined under System.Collections namespace.

In C#, SortedDictionary is a generic collection which is used to store the key/value pairs in the sorted form and the sorting is done on the key. SortedDictionary is defined under System.Collection.Generic namespace. It is dynamic in nature means the size of the sorted dictionary is growing according to the need.

Table

Description automatically generated with medium confidence

[Difference between SortedList and SortedDictionary in C# - GeeksforGeeks](https://www.geeksforgeeks.org/difference-between-sortedlist-and-sorteddictionary-in-c-sharp/)

Table

Description automatically generated

Sorted dictionary should be used when-

More inserts and delete operations are required.

Data in un-ordered.

Key access is enough and index access is not required.

Memory is not a bottleneck.

On the other side, Sorted List should be used when-

More lookups and less inserts and delete operations are required.

Data is already sorted (if not all, most).

Index access is required.

Memory is an overhead.

Default Sorting

● Array.Sort(xxx) - using the System.IComparable

● xxx.Sort() – xxx is a collection – using default comparer

(System.IComparable)

○ String objects are lexicographically ordered

○ Date objects are chronologically ordered

○ Number and sub-classes are ordered numerically

[c# - What's the difference between SortedList and SortedDictionary? - Stack Overflow](https://stackoverflow.com/questions/935621/whats-the-difference-between-sortedlist-and-sorteddictionary)

1. What is the GetHashCode() and Equals() function?

What: Lets you generate Equals and GetHashCode methods.

When: Generate these overrides when you have a type that should be compared by one or more fields, instead of by object location in memory.

Why:

If you're implementing a value type, you should consider overriding the Equals method. You can gain increased performance over the default implementation of the Equals method on ValueType when you do so.

If you're implementing a reference type, you should consider overriding the Equals method if your type looks like a base type, such as Point, String, BigNumber, and so on.

Override the GetHashCode method to allow a type to work correctly in a hash table. Read more guidance on equality operators.

A hash code is a numeric value that is used to identify an object during equality testing. It can also serve as an index for an object in a collection

It is faster to use the return value of GetHashCode to determine whether two objects are equal than to call the default implementation of Equals on the object type.

This method is used to return the hash code for this instance. A hash code is a numeric value which is used to insert and identify an object in a hash-based collection. The GetHashCode method provides this hash code for algorithms that need quick checks of object equality.

Syntax:

public virtual int GetHashCode ();

Return Value: This method returns a 32-bit signed integer hash code for the current object.

Important Points:

Two objects that return different hash codes means objects are not equal but the reverse is not true. Means, equal hash codes do not imply object equality, because different (unequal) objects can have identical hash codes.

The .NET Framework does not guarantee the default implementation of the GetHashCode method, and the value this method returns may differ between .NET Framework versions and platforms, such as 32-bit and 64-bit platforms.

A hash code is not a permanent value so do not serialize, store the hash values in databases etc.

Do not test for equality of hash codes to determine whether two objects are equal.

[C# | Object.GetHashCode() Method with Examples - GeeksforGeeks](https://www.geeksforgeeks.org/c-sharp-object-gethashcode-method-with-examples/)

To compare if the current instance is equal to a given object or not, we make use of a function called Equals(Object) function in C#, which takes the object that is to be compared with its current instance as the parameter and returns true if the object that is be compared with its current instance as the parameter is same as the current instance and returns false if the object that is be compared with its current instance as the parameter is not same as the current instance and this function is available in .NET 5.0 version and other 35 versions.

Syntax to Declare Equals() Function:

Equals(Object);

Where an object is an object that is be compared with its current instance.

Working of Equals(Object) Function

To compare if the current instance is equal to a given object or not, we make use of a function called Equals(Object) function in C#.

The Equals(Object) function takes the object that is to be compared with its current instance as the parameter.

The Equals(Object) function returns true if the object that is be compared with its current instance as the parameter is the same as the current instance.

The Equals(Object) function returns false if the object that is be compared with its current instance as the parameter is not the same as the current instance.

The Equals(Object) function is available in .NET 5.0 version and other 35 versions.

[C# Object Equals | Working of Equals(Object) Function with Examples (educba.com)](https://www.educba.com/c-object-equals/)

1. How Is Dictionary Implemented in C#? How does its Implementation use GetHashCode and Equals Methods of Object?

The keys in a dictionary can be a reference type, i.e., objects. When an object is used as the key, the virtual methods "GetHashCode()" & "Equals()" can change how the dictionary search for the entries depending on if they are overridden, and how they are overridden

The Dictionary is implemented as a hash table.

The hash code to determine the hash slot of the dictionary entry is obtained by calling the virtual method "GetHashCode()" on the key object.

With the possibility of hash collisions, the "Equals()" method on the key object is used to identify the exact dictionary entry.

The default implementations of the two methods on the "System.Object" are equivalent to the following:

"GetHashCode()" - RuntimeHelpers.GetHashCode(). It returns identical hash codes for identical object references. Although there is no guarantee, the chance for a hash collision is very small, which makes it ideal to serve as the hash code for a dictionary entry;

"Equals()" - Object.ReferenceEquals(), which determines whether the objects are the same instance.

The GetHashCode() & Equals() methods are virtual methods that can be overridden. To check if they are overridden, we can use reflection and check the "DeclaringType" of the methods. In case they are overridden, you need to find out how they are overridden if you want to use the objects as dictionary keys. In this note, I only talked about reference types. In case of value types, the idea is the same, but the default implementations of the two methods are totally different.

[C# Dictionary & GetHashCode() & Equals() - CodeProject](https://www.codeproject.com/Articles/1279273/Csharp-Dictionary-GetHashCode-Equals#:~:text=The%20keys%20in%20a%20dictionary%20can%20be%20a,%22Dictionary%3CTKey%2CTValue%3E%22%20class%20is%20implemented%20as%20a%20hash%20table.)

1. What is IEquatable<T> Interface?

The IEquatable<T> interface is used by generic collection objects such as Dictionary<TKey,TValue>, List<T>, and LinkedList<T> when testing for equality in such methods as Contains, IndexOf, LastIndexOf, and Remove. It should be implemented for any object that might be stored in a generic collection.

Replace the type parameter of the IEquatable<T> interface with the type that is implementing this interface.

If you implement IEquatable<T>, you should also override the base class implementations of Equals(Object) and GetHashCode() so that their behavior is consistent with that of the Equals(T) method. If you do override Equals(Object), your overridden implementation is also called in calls to the static Equals(System.Object, System.Object) method on your class. In addition, you should overload the op\_Equality and op\_Inequality operators. This ensures that all tests for equality return consistent results.

[IEquatable<T> Interface (System) | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/api/system.iequatable-1?view=net-7.0)

The generic IEquatable<T> exists to solve a slightly different problem with the Equals method. The Equals method on the Object type takes the parameter of the type Object. We know that this is the only type of parameter, which is possible if we want Object.Equals to work for all the types.

Object is a reference type, which means that, if you want to place a value type as an argument, the value type would be boxed, which will be a performance hit, which is bad. Typically, when we define the value type instead of the reference type it is because we are concerned with the performance. Thus, we always want to avoid this performance overhead of boxing and unboxing.

[Introduction to IEquatable<T> interface in C# (c-sharpcorner.com)](https://www.c-sharpcorner.com/article/story-of-equality-in-net-part-three/)

Comparison (check for equality)

● If type T implements the IEquatable<T> generic interface, then the

equality comparer is the Equals method of that interface.

● If type T does not implement IEquatable<T>, Object.Equals is used.

1. What is IComparable and IComparer interface? What are the differences between them and how to use them?

IComparable<T> interface

Compares the receiving object with the specified object

● Return value must be:

○ <0, if this precedes obj

○ ==0, if this has the same order as obj

○ >0, if this follows obj

Text

Description automatically generated

IComparer<T> Interface

Compares its two arguments

● Return value must be

○ <0, if x precedes y

○ ==0, if x has the same ordering as y

○ >0, if x follows y

Graphical user interface, text, application

Description automatically generated

## **IComparable Interface in C#**

Use the IComparable Interface in C# to sort elements. It is also used to compare the current instance with another object of same type.

It provides you with a method of comparing two objects of a particular type. Remember, while implementing the IComparable interface, CompareTo() method should also be implemented.

Let us see an example −

int IComparable.CompareTo(object ob) {

   Vehicle v=(Vehicle)ob;

   return String.Compare(this.make,v.make);

}

## **IComparer interface in C#**

The IComparer interface is used to sort elements that compare two objects and provides additional comparison method.

## **Example**

private class sortYearAscendingHelper : IComparer {

   int IComparer.Compare(object ob1, object ob2) {

      Vehicle v1=(Vehicle)ob1;

      Vehicle v2=(Vehicle)ob2;

      if (v1.year > v2.year)

      return 1;

      if (v1.year < v2.year)

      return -1;

      else

      return 0;

   }

}

[Difference between IComparable and IComparer Interface in C# (tutorialspoint.com)](https://www.tutorialspoint.com/difference-between-icomparable-and-icomparer-interface-in-chash)

# Coding Questions:

## Write code in c# to solve the following problems. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow best practices when you write the code so that it would be easily readable, maintainable, and efficient. Clearly state your assumptions if you have any. You may discuss with others on the questions, but please write your own code.

1. Write a C# program to iterate a List in reverse order.
2. Write a C# program to iterate a LinkedList in reverse order.
3. Write a C# program to get the ﬁrst and last occurrence of the speciﬁed elements in a List
4. Write a C# program to get the ﬁrst and last occurrence of the speciﬁed elements in a LinkedList
5. Write a C# program to compare two sets and retain elements that exist on both sets.
6. Given a string, ﬁnd the ﬁrst non-repeating character in it and return it's index. If it doesn't exist, return -1.

Input: s = "leetcode", return 0. Input s = "loveleetcode", return 2.

1. Sort a List in descending order. (hint: customize comparer) Eg. given [1,7,3,2,5,9,7], return [9, 7, 7, 5, 3, 2, 1]
2. A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a palindrome, or* false *otherwise*. Test case 1:

Input: s = "race a car"

Output: false

Explanation: "raceacar" is not a palindrome.

Test Case 2:

Input: s = "A man, a plan, a canal: Panama" Output: true

Explanation: "amanaplanacanalpanama" is a palindrome.

1. A DNA string is made up of four symbols: A, T, C, and G. Each symbol has a complement:1) A and T complement each other. 2) C and G complement each other

Determine the reverse complement of a DNA string by reversing the symbols in the string and replacing each symbol in the reversed string by its complement

Example.

Input string s = GTCAG

The method should ﬁrst reverse the string: GTCAG -> GACTG Then replace each symbol with its complement: GACTG -> CTGAC Then return CTGAC

Write a c# program/method which takes a string as input, and returns the reverse complement of the given DNA string