In C#, a `SortedList` is a collection type that stores key-value pairs in sorted order based on the keys. It's part of the System.Collections namespace and provides a sorted dictionary-like data structure. Here's a full description of the `SortedList` class:

1. Key-Value Pairs:

- A `SortedList<TKey, TValue>` is a collection that stores key-value pairs. Keys are used to index and access values. The keys must be unique, and they are stored in sorted order.

2. Sorted Order:

- The primary characteristic of a `SortedList` is that it maintains its elements in sorted order based on the keys. The sorting order is determined by the natural ordering of the key type (using the `IComparable` interface) or a custom `IComparer` provided during initialization.

3. Performance:

- A `SortedList` provides efficient look-up and retrieval operations. Because elements are sorted, it can perform binary searches to find keys and values, resulting in logarithmic time complexity for these operations.

4. Key Type:

- The key type in a `SortedList` must be sortable. This means that it should implement the `IComparable` interface or be used with a custom `IComparer` to define the sorting order.

5. Dynamic Sizing:

- A `SortedList` dynamically resizes itself as items are added, ensuring that there's always enough capacity to hold new elements efficiently.

6. No Duplicate Keys:

- A `SortedList` does not allow duplicate keys. If you attempt to add a key that already exists, the new value will replace the existing value associated with that key.

7. Indexing:

- Elements can be accessed by their keys using the indexing operator (`[]`). This allows you to quickly retrieve values associated with specific keys.

8. Enumerable:

- A `SortedList` can be easily enumerated using `foreach` loops, making it straightforward to iterate over its key-value pairs.

9. Memory Overhead:

- `SortedList` generally has a slightly higher memory overhead compared to other dictionary-like collections due to its need to maintain the sorted order of elements.

10. Initialization:

- You can create a `SortedList` by specifying the key and value types it will hold. For example, `SortedList<string, int>` will hold a collection of string keys and integer values.

11. Sorting Mechanism:

- You can choose the sorting mechanism of the `SortedList` by providing a custom `IComparer` during initialization or relying on the natural sorting order of the key type. The sorting is performed based on the key values, not the values themselves.

12. Use Cases:

- `SortedList` is useful when you need to maintain a collection of key-value pairs that are always sorted by their keys. Common use cases include maintaining a dictionary of words and their frequency counts, managing a list of appointments sorted by date, or organizing a catalog of products by SKU or name.

In summary, a `SortedList` is a collection in C# that offers efficient operations for maintaining and accessing key-value pairs in sorted order. It's suitable for scenarios where you require both key uniqueness and elements sorted based on keys, making it a valuable tool for various applications and data organization needs.