C# SortedDictionary Cheat Sheet

Namespace: using System.Collections.Generic;

1. What is it?

A SortedDictionary works exactly like a normal Dictionary, but with one superpower: **The Keys are always sorted.**

- **Insertion Speed:** $O(\log n)$ (Slightly slower than normal Dictionary).
- **Lookup Speed:** $O(\log n)$.
- **Sorting:** Automatic upon insertion.

2. Creating a SortedDictionary

Default (Ascending Order)

Sorts numbers $0 \to 9$ or Strings $A \to Z$ automatically.

```
SortedDictionary<string, int> phoneBook = new SortedDictionary<string, int>();
```

Custom (Descending/Reverse Order)

Common trick for "High Score" lists. You pass a Comparer to the constructor.

```
// Sorts Keys Z -> A or 9 -> 0
var reverseDict = new SortedDictionary<int, string>(
    Comparer<int>.Create((x, y) => y.CompareTo(x))
);
```

3. Adding & Modifying

- **.Add(key, value)**: Adds a pair. Throws an error if the Key already exists.
- **dict[key] = value**: The "Safe" add. If key exists, it updates value. If not, it adds it.

```
phoneBook.Add("Alice", 123);
phoneBook["Bob"] = 456;
phoneBook["Alice"] = 789; // Updates Alice to 789
```

4. Accessing & Checking

- **.Count**: Returns the number of items in the dictionary.
- **.ContainsKey(key)**: Returns true if key exists. $O(\log n)$.
- **.ContainsValue(value)**: Returns true if the value exists. ⚠️ **Warning: This is** $O(n)$ **(Slow)!** It has to search the entire list one by one.
- **.TryGetValue(key, out val)**: Tries to get value, returns false if missing (Best for

performance).

- **.Keys / .Values**: properties that return a collection of just the keys or just the values (Sorted!).

```
if (phoneBook.ContainsKey("Alice")) {
    Console.WriteLine(phoneBook["Alice"]);
}

if (phoneBook.Count == 0) {
    Console.WriteLine("Phonebook is empty!");
}
```

5. Iteration (The Superpower)

When you use foreach, the items come out **already sorted by Key**. You do not need to call .Sort().

```
phoneBook.Add("Charlie", 3);
phoneBook.Add("Alice", 1);
phoneBook.Add("Bob", 2);

foreach (var item in phoneBook)
{
    // Output will be: Alice, then Bob, then Charlie
    Console.WriteLine($"{item.Key}: {item.Value}");
}
```

6. Removing

- **.Remove(key)**: Removes the item with that key. Returns true if successful.
- **.Clear()**: Wipes the entire dictionary.

💡 Comparison: When to use what?

Collection	Sorting	Speed (Insert/Get)	Use When...
Dictionary	Random / None	$O(1)$ (Fastest)	You just need lookups and don't care about order.
SortedDictionary	Auto-Sorted by Key	$O(\log n)$	You need to print/process data in order (e.g., "Print words alphabetically").
SortedList	Auto-Sorted by Key	$O(n)$ (Slow Insert)	You need to access items by Index (e.g.,

			list.Keys[0]) and use less memory.

⚠️ Important Warning

SortedDictionary only sorts by **KEY**, not by Value.

- If you need to sort by **Value** (e.g., "Word Frequency: Most frequent first"), you cannot use SortedDictionary alone. You must use a normal Dictionary and then LINQ:
    - dict.OrderBy(x => x.Value)