Array and String Methods in C# – Deep Concepts with Simple Examples

This document provides an in-depth understanding of commonly used array and string methods in C#. Each section includes when to use the method, when to avoid it, and practical examples with clear explanations.

Table of Contents

---

1. Arrays in C# – Basics

An array is a fixed-length collection of elements of the same type.

string[] tasks = new string[5];

You must define the size while creating it.

Key Points

- Arrays start at index 0.
- Arrays are reference types.
- Size cannot change after creation.

---

2. Array.IndexOf()

Finds the index of the **first exact match** in an array.

int index = Array.IndexOf(tasks, "Fix bike");

When to Use

- Searching for a known, exact value.
- Small fixed-size arrays.

When to Avoid

- Case-insensitive or partial match needed.
- Large datasets (performance).

Example

string[] names = { "Ram", "Sam", "Pam" };

int pos = Array.IndexOf(names, "Pam"); // pos = 2

---

3. Array.Copy()

Copies values from one array to another.

Array.Copy(sourceArray, destinationArray, length);

When to Use

- To work on a copy without affecting original.
- To extract part of an array.

When to Avoid

- If you need conditional filtering.
- If you don’t know array length.

Example

string[] original = { "A", "B", "C" };

string[] backup = new string[3];

Array.Copy(original, backup, 3);

---

4. Array.Sort()

Sorts the array in ascending order.

Array.Sort(array);

When to Use

- Simple alphabetic/numeric sorting.

When to Avoid

- You want descending order (requires Reverse).
- Nulls in array may cause issues.
- Sorting complex objects (needs IComparer or custom logic).

Example

string[] tasks = { "Wash", "Clean", "Cook" };

Array.Sort(tasks); // Cook, Clean, Wash

---

5. Array.Reverse()

Reverses the elements of an array.

Array.Reverse(array);

When to Use

- Descending sort (after Sort).
- Reversing strings.

When to Avoid

- You only want to reverse part of an array (you'll need to copy that part first).

Example

int[] nums = { 1, 2, 3 };

Array.Reverse(nums); // 3, 2, 1

---

6. Array.Clear()

Resets elements in an array to default.

Array.Clear(array, 0, array.Length);

When to Use

- You want to wipe all values.

When to Avoid

- You expect it to resize the array (it doesn't).

Example

string[] data = { "A", "B", "C" };

Array.Clear(data, 0, data.Length); // All elements become null

---

7. Array.Find() and Array.Exists()

Array.Find()

Returns the **first matching element** that satisfies a condition.

string result = Array.Find(tasks, t => t.Contains("bike"));

Array.Exists()

Checks if any element matches a condition.

bool exists = Array.Exists(tasks, t => t.Contains("bike"));

Use helper methods if you are not using lambda:

bool match(string t) => t.Contains("test");

Array.Find(tasks, match);

---

8. Shallow vs Deep Copy (Complex Objects)

Shallow Copy

Array.Copy(original, copy, length);

Only references are copied — both arrays point to same objects.

Deep Copy

Create new object instances:

deep[i] = new TaskItem { Title = original[i].Title };

---

9. String Methods: ToCharArray(), Split(), Join()

ToCharArray()

Splits a string into characters.

char[] chars = "abc".ToCharArray();

Split()

Splits a string based on a delimiter.

string[] words = "Fix the bike".Split(' ');

Join()

Combines array elements into a single string.

string result = String.Join("-", words);

---

10. Common Mistakes & Tips

Mistake	Explanation
Using ToCharArray() instead of Split() for words	ToCharArray() gives characters, not words
Expecting Clear() to resize	It resets values, not length
Thinking Copy() creates deep copy	It doesn't for objects
Sorting array with nulls	Nulls go to front, may break logic

This document can be added to your GitHub repo alongside your Task Manager code to explain your learning and implementation of C# array and string methods in real-world style.