C# Array Functions with Real-World Examples

# 1. Array.Length

Returns the number of elements in the array.

int[] temperatures = { 23, 19, 31, 27, 30, 25 };  
Console.WriteLine(temperatures.Length); // Output: 6

# 2. Array.Rank

Returns the number of dimensions. For single-dimensional arrays, it's 1.

int[] numbers = { 10, 20, 30 };  
Console.WriteLine(numbers.Rank); // Output: 1

# 3. Array.IndexOf

Finds the index of a value in an array.

string[] fruits = { "Apple", "Banana", "Cherry" };  
int index = Array.IndexOf(fruits, "Banana");  
Console.WriteLine(index); // Output: 1

**Can Array.IndexOf work on 2D arrays or jagged arrays?**

**✅ 1. Jagged Arrays (int[][]):**

Yes, **Array.IndexOf** **can** be used on a **jagged array**, **but only on the outer array** (i.e., the array of arrays), **not the elements inside the inner arrays directly**.

**❌ 2. 2D Arrays (int[,]):**

No, **Array.IndexOf does NOT work on 2D arrays directly**. It expects a **single-dimensional array**. If you try using it on a [,], it will throw a **compile-time error**.

int[][] jagged = new int[][]

{

new int[] { 1, 2, 3 },

new int[] { 4, 5, 6 },

new int[] { 7, 8, 9 }

};

// Search for the inner array { 4, 5, 6 }

int[] target = { 4, 5, 6 };

// This works only if you reference the same object

int index = Array.IndexOf(jagged, target);

Console.WriteLine("Index: " + index); // Output: -1 (different object)

# 4. Array.LastIndexOf

Finds the last index of a repeated value.

int[] nums = { 1, 2, 3, 2, 4 };  
int lastIndex = Array.LastIndexOf(nums, 2);  
Console.WriteLine(lastIndex); // Output: 3

# 5. Array.Find

Finds the first element that matches a condition.

int[] numbers = { 10, 15, 30, 25 };  
int result = Array.Find(numbers, n => n > 20);  
Console.WriteLine(result); // Output: 30

# 6. Array.FindIndex

Finds the index of the first element that matches a condition.

int[] numbers = { 5, 10, 15, 20 };  
int idx = Array.FindIndex(numbers, n => n > 10);  
Console.WriteLine(idx); // Output: 2

# 7. Array.Exists

Checks if any element matches a condition.

int[] ages = { 18, 21, 30 };  
bool exists = Array.Exists(ages, age => age > 25);  
Console.WriteLine(exists); // Output: True

# 8. Array.BinarySearch

Searches for a value in a sorted array.

int[] sortedArray = { 10, 20, 30, 40 };  
int position = Array.BinarySearch(sortedArray, 30);  
Console.WriteLine(position); // Output: 2

# 9. Array.Sort

Sorts the array.

int[] values = { 3, 1, 4, 2 };  
Array.Sort(values);  
Console.WriteLine(string.Join(", ", values)); // Output: 1, 2, 3, 4

# 10. Array.Reverse

Reverses the elements in the array.

int[] sequence = { 1, 2, 3 };  
Array.Reverse(sequence);  
Console.WriteLine(string.Join(", ", sequence)); // Output: 3, 2, 1

# 11. Array.Clear

Clears part of the array.

int[] nums = { 5, 10, 15 };  
Array.Clear(nums, 1, 2);  
Console.WriteLine(string.Join(", ", nums)); // Output: 5, 0, 0

# 12. Array.Copy

Copies elements from one array to another.

int[] source = { 1, 2, 3 };  
int[] target = new int[3];  
Array.Copy(source, target, 3);  
Console.WriteLine(string.Join(", ", target)); // Output: 1, 2, 3

# 13. Array.Resize

Changes the size of an array.

int[] arr = { 1, 2 };  
Array.Resize(ref arr, 4);  
Console.WriteLine(string.Join(", ", arr)); // Output: 1, 2, 0, 0

# 14. LINQ - Where

Filters elements.

int[] numbers = { 1, 2, 3, 4 };  
var evens = numbers.Where(n => n % 2 == 0);  
Console.WriteLine(string.Join(", ", evens)); // Output: 2, 4

# 15. LINQ - Select

Transforms elements.

int[] nums = { 1, 2, 3 };  
var squared = nums.Select(n => n \* n);  
Console.WriteLine(string.Join(", ", squared)); // Output: 1, 4, 9

# 16. LINQ - Sum, Max, Min, Average

Performs aggregate operations.

int[] values = { 2, 4, 6 };  
Console.WriteLine(values.Sum()); // 12  
Console.WriteLine(values.Max()); // 6  
Console.WriteLine(values.Min()); // 2  
Console.WriteLine(values.Average()); // 4

# 17. LINQ - All, Any

Checks conditions.

int[] scores = { 70, 80, 90 };  
Console.WriteLine(scores.All(s => s >= 60)); // True  
Console.WriteLine(scores.Any(s => s == 100)); // False

# 18. LINQ - Distinct

Removes duplicate elements.

int[] nums = { 1, 2, 2, 3 };  
var unique = nums.Distinct();  
Console.WriteLine(string.Join(", ", unique)); // Output: 1, 2, 3

# 19. LINQ - OrderBy / OrderByDescending

Sorts using LINQ.

int[] numbers = { 5, 3, 8 };  
var sorted = numbers.OrderBy(n => n);  
Console.WriteLine(string.Join(", ", sorted)); // Output: 3, 5, 8

# 20. LINQ - ToArray

Converts a LINQ result back to an array.

int[] values = { 10, 20, 30 };  
var result = values.Where(v => v > 15).ToArray();  
Console.WriteLine(string.Join(", ", result)); // Output: 20, 30