Part01

Question1: What is the default value assigned to array elements in C#?

The elements of an array can be initialized to known values when the array is created. Beginning with C# 12, all of the collection types can be initialized using a [Collection expression](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/collection-expressions). Elements that aren't initialized are set to the [default value](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/default-values). The default value is the 0-bit pattern.

Question2: What is the difference between Array.Clone() and Array.Copy()?

clone() creates a **new** object with the same characteristics as the old array, i.e., same size, same type, and *same* contents , and with distinct address

A shallow copy of an [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0) copies only the elements of the [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0), whether they are reference types or value types, but it does not copy the objects that the references refer to. The references in the new [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0) point to the same objects that the references in the original [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0) point . In contrast, a deep copy of an [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0) copies the elements and everything directly or indirectly referenced by the elements. The clone is of the same [Type](https://learn.microsoft.com/en-us/dotnet/api/system.type?view=net-9.0) as the original [Array](https://learn.microsoft.com/en-us/dotnet/api/system.array?view=net-9.0).

Question3:What is the difference between GetLength() and Length for multi dimensional arrays?

GetLength(dimension): Returns the size of a specific dimension in the array , Accepts a single argument representing the dimension (0-based index). array.GetLength(0) gives the number of rows. Useful for iterating over rows or columns in loops.

Length : Returns the total number of elements in the array , Does not accept arguments; works on the entire array , array.Length gives the total count of elements (rows \* columns) , Useful to find total memory allocation or size of the array.

Question4: What is the difference between Array.Copy() and Array.ConstrainedCopy()?

Use Array.ConstrainedCopy() when dealing with arrays of mixed or complex types to ensure strict error handling, Same as Array.Copy() but includes additional type and range checks..

Array.Copy() is more commonly used for simple array copying tasks, Copies elements from a source array to a destination array.

Question5: Why is foreach preferred for read-only operations on arrays?

It iterates through each element of the array automatically, making the code cleaner and easier to understand. Its isn’t index-based

foreach handles the iteration boundaries automatically. There’s no risk of accessing invalid indices or causing runtime exceptions.

Although foreach introduces slight overhead due to the use of enumerators, modern compilers optimize this for arrays, making the performance nearly identical to a for loop for read-only operations.

**When Modifications Are Needed**: foreach doesn’t allow modifying the original array elements. Use a for loop in such cases

Question6: Why is input validation important when working with user inputs?

Input validation is critical when working with user inputs for the following reasons:

**Prevent Program Crashes**:

* + User inputs can be unpredictable. Without validation, invalid inputs (e.g., non-numeric characters for numeric input) can cause runtime exceptions.
  + Validation provides immediate feedback, guiding users to provide correct inputs without confusion.

Question7: How can you format the output of a 2D array for better readability?

**Use Fixed Spacing**:

* Align elements in columns by specifying a width, e.g., {value,4} ensures each value occupies 4 spaces, creating uniformity.

**Tab Delimiters**:

* Use \t for tab spacing between columns.
* Example: Console.Write($"{matrix[i, j]}\t");

Question8: When should you prefer a switch statement over if-else?

**Numerous Fixed Cases**:

* Use switch when handling multiple discrete values (e.g., months, days).
* It is cleaner and more readable compared to multiple if-else statements.

**Constant Comparisons**:

* switch is ideal for matching constant values (e.g., integers, enums, strings).

**Performance Optimization**:

* In some cases, switch can be optimized by the compiler to a jump table, making it faster than evaluating multiple if-else conditions sequentially.

Question9: What is the time complexity of Array.Sort()?

 **Quick Sort** for most cases.

 **Heap Sort** for fallback when recursion depth grows too large.

 **Insertion Sort** for small partition

**Best Case**: **O(n log n)**

* Occurs when the array is already partially sorted or the data is well-distributed for Quick Sort.

**Average Case**: **O(n log n)**

* Typical scenario for most datasets.

**Worst Case**: **O(n log n)**

* Introspective Sort avoids the O(n²) worst case of Quick Sort by switching to Heap Sort if the recursion depth exceeds a certain limit.

Question10: Which loop (for or foreach) is more efficient for calculating the sum of an array, and why?

When calculating the sum of an array in C#, **for** loops are generally **more efficient** than foreach loops.

**for Loop**

* **How it Works**:
  + Accesses array elements directly using an index.
  + No additional overhead because the loop operates with simple integer increments and direct array access.

**Advantages**:

* Optimal for performance as the compiler can optimize direct access to elements using the index.
* Allows full control over the iteration (e.g., skipping elements, reverse iteration).

**foreach Loop**

* **How it Works**:
  + Iterates over the collection using an enumerator under the hood.
  + The enumerator introduces slight overhead because it tracks the current element and checks boundaries.

**Advantages**:

* Cleaner and easier to read, especially for iteration-only logic.
* Reduces the risk of off-by-one errors since there is no explicit index

**Use for**:

* When performance is critical.
* When you need control over the iteration.

**Use foreach**:

* When code readability is a priority.
* When you don't need to manage indices manually.

Part02:

What happens if the user enters a value outside the range of 1 to 7?

The else statement as I handle this error when using this enum Invalid input. Please enter a number between 1 and 7