Part01

Question01

Elements are automatically initialized to default values.
For int → 0, bool → false, reference types → null.

Question02

Clone() creates a new array with the same elements and returns it.
Copy() copies elements from one array to another existing array.

Question03

GetLength(dimension) returns size of one dimension.
Length returns total number of elements in all dimensions.

Question04

Copy() performs normal copy and may leave partial results on failure.
ConstrainedCopy() guarantees rollback if copy fails, ensuring data integrity.

Question05

It is safer, cleaner, and prevents accidental modification or index errors.

Question06

Prevents crashes, avoids invalid data, and ensures program reliability and security.

Question07

Use nested loops and alignment formatting like Console.Write($"{value,4}") to print matrix form.

Question08

When checking many fixed discrete values.
Switch is faster, cleaner, and more readable.

Question09

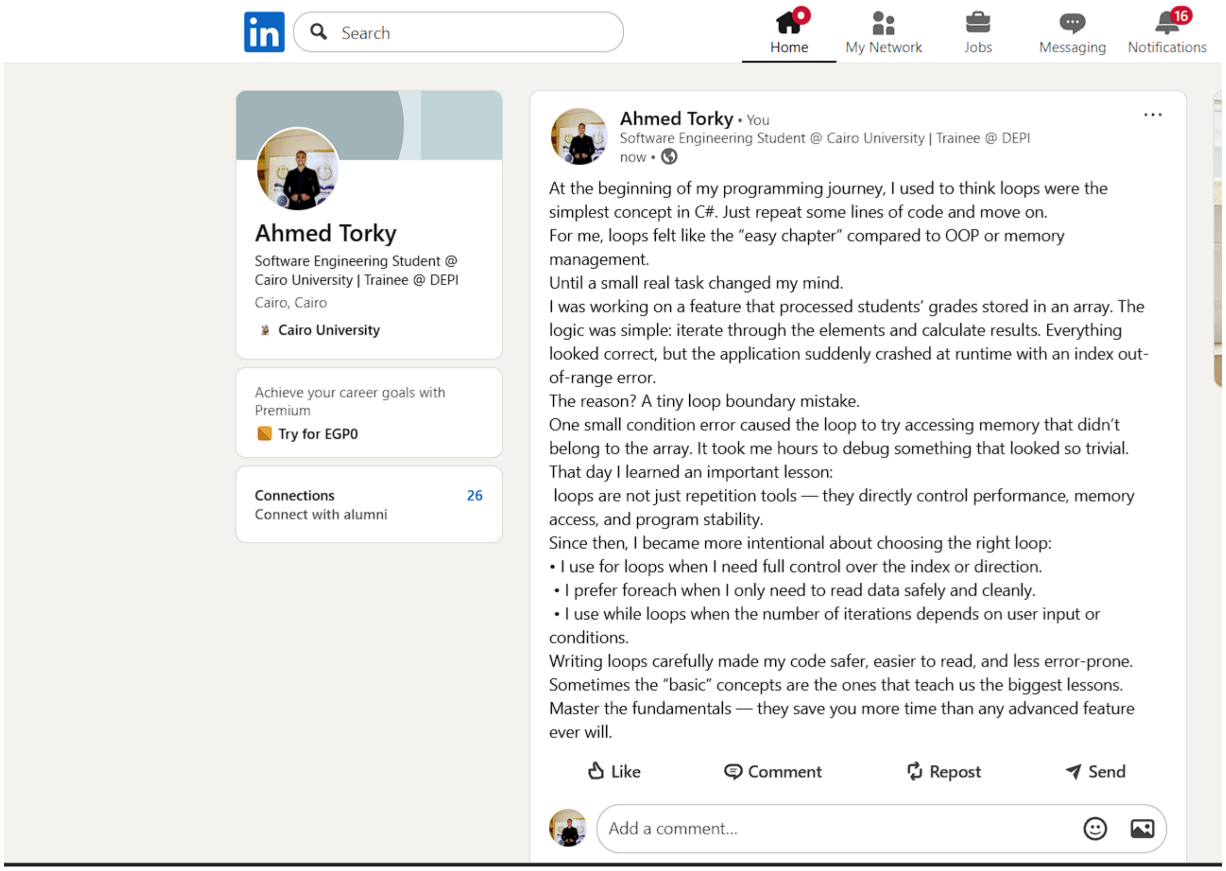
Average complexity: O(n log n)

Question10

Both are similar, but foreach is slightly safer and cleaner. Performance difference is negligible.

Part02

LinkedIn article:



At the beginning of my programming journey, I used to think loops were the simplest concept in C#. Just repeat some lines of code and move on.
For me, loops felt like the "easy chapter" compared to OOP or memory management.
Until a small real task changed my mind.
I was working on a feature that processed students' grades stored in an array. The logic was simple: iterate through the elements and calculate results. Everything looked correct, but the application suddenly crashed at runtime with an index out-of-range error.
The reason? A tiny loop boundary mistake.
One small condition error caused the loop to try accessing memory that didn't belong to the array. It took me hours to debug something that looked so trivial.
That day I learned an important lesson:
loops are not just repetition tools — they directly control performance, memory access, and program stability.
Since then, I became more intentional about choosing the right loop:
• I use for loops when I need full control over the index or direction.
• I prefer foreach when I only need to read data safely and cleanly.
• I use while loops when the number of iterations depends on user input or conditions.
Writing loops carefully made my code safer, easier to read, and less error-prone.
Sometimes the "basic" concepts are the ones that teach us the biggest lessons.
Master the fundamentals — they save you more time than any advanced feature ever will.

Question01

Enum.Parse throws an exception or results in invalid enum value. Validation is required.

Part03-Bonus

Default stack and heap size:

Stack: usually 1MB per thread(4 MB for 64-bit processes in some configurations)
Heap: grows dynamically
Stack is faster but small; heap is larger but slower.

What is time complexity:

Time complexity measures how algorithm execution time grows relative to input size using Big-O notation.