TASK A

One of the fastest ways to process large chunks of data in managed C# is iterating an array of structs. However, working with arrays directly can sometimes be inconvenient, especially when dynamically adding and removing items is a common operation.

Implement a custom FastList<T> type that provides convenient API like System.Collections.Generic.List<T>, but also allows to directly access the underlying array from outside to achieve maximum performance in critical regions.

Besides required core functionality, feel free to extend your result with a bigger API surface, interface implementations, documentation, whichever you see fit to match the use case in the best way possible.

SUGGESTED STEPS

In case you are unsure how to approach the task, the following steps should give you a rough outline on what you could do. Feel free to choose a different path if you like.

1. Implement a regular list collection type, similar to List<T>, with a minimal API that a developer would need to add, remove, iterate or find items.
2. Extend or adjust your collection type to enable high performance use cases.
3. Refine and polish your collection type to streamline it for the intended use case and your interpretation of it.
4. Consider best practices and pitfalls when using your list type and choose an appropriate way to communicate them to the user.

EXAMPLE USE CASES

These use cases are here for illustrative purposes only, so you can put the task into a context you may be familiar with. None of them need or should be incorporated into your implementation.

- Gathering and processing large amounts of dynamic drawcalls in a custom rendering engine.
- Realtime processing of high frequency sensor data samples.
- Working with large quantities of geometric data, such as mesh processing.
- CPU particle simulation.
- CPU physics engine allowing for large quantities of objects.