IT230 Project 2
Score: _____/80 v1.2

Specifications

Create a visual sorting program by using the project in the assignment repository as a starting point. This program will visually demonstrate the process of sorting algorithms.

In the project, there's an empty directory called “Algorithms”. For each sorting algorithm you implement, add a new class in this directory. All sorting algorithms must implement the `ISortable` interface, which is already included in the project.

The `ISortable` interface includes a delegate that points to a method called `Repaint()`. This method is used to refresh the UI during sorting and should be called every time the array is modified to visually update the sorting process. For example, in a bubble sort implementation, `Repaint()` should be called after each swap to ensure the UI reflects the changes in the array. You can call it like this:

```
await Repaint(null);
```

You can replace null with an integer value, representing a millisecond delay. Passing null uses the default delay of 2ms.

Additionally, the interface contains a `CancellationToken`, which allows the user to cancel the sorting process. Every major loop or recursive call in your sorting algorithms must include the following instruction:

```
cancellationToken.ThrowIfCancellationRequested();
```

This ensures that the algorithm can be stopped promptly if the user cancels the operation. For example, in a bubble sort algorithm, you would place this instruction at the beginning of the while loop.

In the Index.razor.cs file, the `Sort()` method is triggered after the user clicks the "Go!" button on the UI. This method will start the selected sorting algorithm and pass in the array to be sorted.

In addition to implementing bubble sort, selection sort, insertion sort, quicksort, and mergesort, you are required to find four additional sorting algorithms to implement. You can refer to the Big-O Cheat Sheet website (https://www.bigocheatsheet.com) or this Wikipedia page (https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms) for ideas on additional sorting algorithms.

After sorting completes, ensure the correct best, average, and worst-case time complexities of the chosen algorithm are displayed properly on the screen.

Feel free to make UI/UX improvements to the application.

Modifying the Code to Add New Sorting Algorithms

This section explains where to make changes in the existing code when you add new algorithms.

- **`Algorithm` Enum**: There is a TODO in the `Algorithm` enum where you are required to add four additional sorting algorithms. Try to select algorithms with varied time complexities.
- **Sorting Algorithm Instances**: Another TODO is located in the `SortingAlgorithmInstances` array. Here, you need to instantiate each new algorithm you add to the `Algorithm` enum. For each algorithm, specify the best, average, and worst-case complexities. An example has been provided for bubble sort, and you should follow a similar structure for the additional algorithms you implement. Be sure to modify this array to include all the algorithms you have implemented in the Algorithms directory.
- **Algorithm Dropdown**: There is a TODO in the `<select>` element within the Index.razor file. You are required to add options for your new sorting algorithms here. Each new algorithm you add to the `Algorithm` enum and `SortingAlgorithmInstances` array should also be selectable by the user from this dropdown.
- **`ChangeAlgorithmToRun()` Method**: There is also a TODO in the `ChangeAlgorithmToRun()` method, where you must add your new sorting algorithms to the switch statement. This will allow the user to select them from the UI.

Constraints

- Include a small delay between any swaps or changes in the array to allow users to see the gradual progression of sorting in the UI. You can use the `Repaint()` method with a delay to achieve this.
- All sorting algorithms must implement `ISortable`.
- For each sorting algorithm you implement, leave comments in your code that explain the algorithm and demonstrate that you understand how it works.
- Do not implement sorting algorithms that would not be easy to visualize. For example, do not use Counting Sort for this project.

Submission

The submission for this project is the commit ID you want to be graded. You will submit the commit ID via the Canvas assignment. If you need help finding the commit ID, ask for help.

See the grading rubric on Canvas for grading criteria.

Remember to write efficient code. Optimize your code wherever possible. Points will be deducted for unoptimized code.

Daily commits and pushes to your assignment GitHub repository are <u>mandatory</u>.

Tips

- Ask your instructor for any Blazor-specific help.
- I recommend creating a separate C# project to test new sorting algorithms.

Academic Honesty Policy

All code submissions must be original and authored by the student. Any code sourced from another student, falsely presented as one's own, or derived from third-party websites or generated by AI, will constitute a breach of the school's academic honesty policy.

Daily commits and pushes to your assignment GitHub repository are <u>mandatory</u> while working on your project. Failure to comply will result in deductions from your final grade at minimum, and could lead to academic honesty violations.