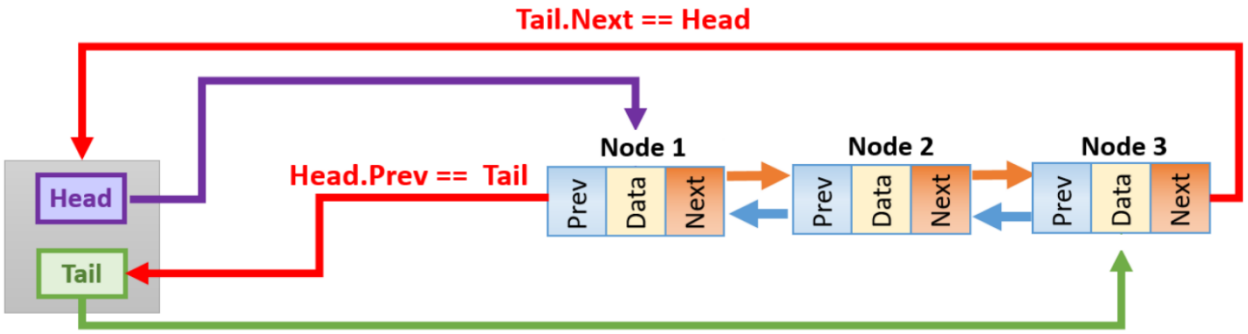
Part 1: Doubly Circular Linked List

In this data structure, similar to a circular linked list, the tail node of a doubly linked list, points back to the head node. This part consists of creating your own implementation using the generic classes in C#. **Hint**: Start by creating two generic classes **DCLinkedList<T>** and **GenericNode<T>**. Use these two classes to implement the same doubly linked list we have seen, with adding a new feature that the next of tail points to the head of the list and the previous of head points to the tail of the list.



Define the following methods in class DCLinkedList. Do not forget to write in comments the time complexity of each method (O(1), O(n), O(n2), etc). You have to call all these methods in the Main method of your program.

Method	Description		
1. **AddFirst(GenericNode<T>)**	Adds the specified new node at the start of the list.		
2. **AddLast(GenericNode<T>)**	Adds the specified new node at the end of the list.		
3. **AddAfter(GenericNode<T> node, GenericNode<T> newNode)**	Adds a new node after an existing node in the list.		
4. **AddBefore(GenericNode<T> node, GenericNode<T> newNode)**	Adds a new node before an existing node in the list.		
5. **RemoveFirst()**	Removes the node at the start of the list.		
6. **RemoveLast()**	Removes the node at the end of the list.		
7. **Remove(GenericNode<T>)**	Removes a given node from the list.		
8. **FindFirst(T)**	Finds the first node that contains the specified value.		
9. **FindLast(T)**	Finds the last node that contains the specified value.		
10. **Merge(DCLinkedList<T>)**	Merges the given doubly circular linked list at the end of the list.		
11. **Clear()**	Removes all nodes from the list.		
12. **ToString()**	Returns a string containing all node values.		
13. **ToStringReverse()**	Returns a string containing all node values (reverse mode).		
14. **RotateRight(int nPositions)**	Rotate the list to the right nPosition times. Example: For a list containing the nodes **"A", "B", "C", "D"**: - Calling RotateRight(1) should result **"D", "A", "B", "C"**. - Calling RotateRight(2) should result **"C", "D", "A", "B"**.		
15. **RotateLeft(int nPositions)**	Rotate the list to the left nPosition times. Example: For a list containing the nodes **"A", "B", "C", "D"**: - Calling RotateLeft(1) should result **"B", "C", "D", "A"**. - Calling RotateLeft(2) should result **"C", "D", "A", "B"**.		

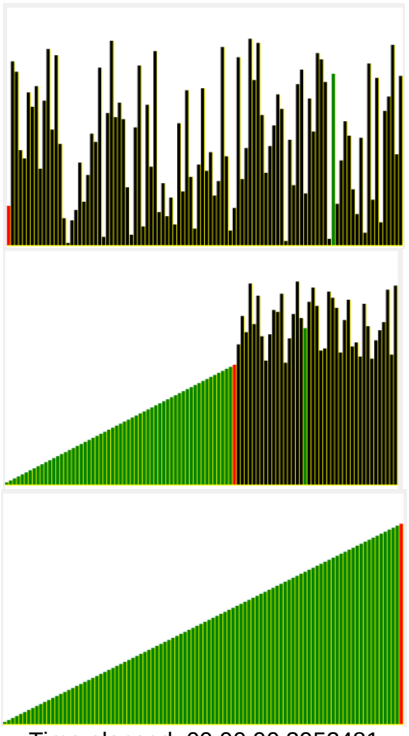
Part 2: Implementation of Sorting Algorithms

Create a Windows Forms Application that displays a histogram animation of different sorting algorithms to sort an array of 100 random numbers. The algorithms are: bubble sort, selection sort, and quick sort. The application creates an array of 100 integers, such as array[i] = i. Then, this array is randomly shuffled before applying a sorting algorithm. Finally, the application displays the running time to be able to compare the complexities of each algorithm.

Step 1 - Initialization: The size of each histogram at index i represents the value of array's element at index i. The values of elements range from 1 to 100, and they are randomly shuffled.

Step 2 - Execution: The histogram animation displays the execution of a selected sorting algorithm. The green histograms are the sorted elements and the red histogram represents the current element that is being compared in the sorting algorithm.

Step 3 - Result: The green histograms are the sorted elements. All the elements should be sorted. The execution time of the current sorting algorithm is displayed.



Time elapsed: 00:00:00.3952481