CPSC 221

Recursive Quicksort

"Simplicity is the ultimate sophistication."
--Leonardo da Vinci

Objectives

- Implement a recursive Quicksort algorithm in Java.

Tasks

Complete the `Sort` static utility class, using a recursive implementation of the Quicksort sorting algorithm.

1. Complete the `Sort` class without modifying the public `sort` methods or the method signatures of the private `quicksort` methods. No new public methods should be added to the `Sort` class.
    - One version of the algorithm will compare the elements in the lists using the class-defined `compareTo` method, which is required by the `Comparable` interface. Because your `Sort` class will be tested on lists with `Integer` objects, you will not need to create this method. It's provided by the `Integer` class.
    - The other version of the algorithm will use a `Comparator` to compare elements in the lists. To test this version of the `sort` method, you will need to provide an object that implements the `Comparator` interface. You can create your own, but the given test class has one defined internally that it uses.
    - Your implementation must work with an instance of the provided `WrappedDLL` class.

- Your quicksort implementations should run in O(n log(n)) time. Do not write or call code that would make them worse than that. Therefore, you should avoid all of the indexed methods, including the indexed `ListIterator` constructor.

2. Use the provided `SortTester` class to test your implementation. If you wish, you may add additional tests, but the given tests will be expected to pass as written.

Quicksort Algorithm

1. Choose one element from the list as the pivot (or partition) element. All other elements will be compared to this element.
   The choice of pivot is of great importance if you want to avoid a worst-case runtime for quicksort, but for this assignment, keep things simple and just use the first or last element.

2. Organize all other elements so that elements smaller than the pivot are on its left (left partition) and all other elements are on its right (right partition).
   For this application, you are advised to put smaller elements in a new left list and the rest in a new right list. Reassembling the original list with sorted elements can wait until after the left and right lists are sorted.

3. Recursively apply the quicksort algorithm to the left and right partitions.

Files

Begin your `Sort` class with the following files:

- Sort.java
- WrappedDLL.java
- IndexedUnsortedList.java

For testing your implementation, you will need the following file:

- SortTester.java

Grading

Points will be awarded according to the following breakdown:

Tasks	Points
Required functionality - sorts correctly	50
Design - uses recursion correctly	30

Required Files

You should submit your copies of the following file:

- Sort.java (updated)