1. Create a `TreeSorter` class containing a static `sort` method that implements the array-sorting algorithm described in the `Treesort` handout. Your method should be generic over all arrays of objects that implement the `Comparable` interface. Test your `sort` method by having it sort some small arrays and checking that `sort` arranges their elements in ascending order.

2. Add instrumentation to the `sort` method: Arrange for it to print out the height and internal path length of the binary search tree that it builds from the array elements just before it begins to transfer them back into the array.

3. If this sorting method is applied to an array of one thousand elements that is originally in descending order, what will be the height and the internal path length of the binary search tree that it constructs? Compute your answers first using pencil and paper, recording the results, and then check them experimentally by having your instrumented `sort` procedure perform the sort.

4. Implement a generic, static method for randomly permuting the elements of an array. Add a call to this method at the beginning of the `sort` method. The random permutation should almost always produce a considerable reduction both in the internal path length and in the height of the binary search tree that `sort` constructs when the data are strongly ordered initially. Confirm this experimentally by once again sorting a thousand-element array that starts out in descending order.