UNBC Laboratory Assignment
Computer Science 200—Fall 2018

Sorting and Timing I

Purpose:

To gain experience with coding sorting algorithms and testing their behaviour. verify their O() timing characteristics.

Due Date:

This assignment is due ***14 November 2018***.

Assignment:

⇒ Implement insertion sort, and Shell sort.

Implement each of these sorting algorithms in a manner suitable for inclusion in a library of routines to be supplied to another user. In particular:

- Make sure that your algorithm works when the array supplied by the user is very small, even 0 or 1 elements.
- Code your algorithms with generic public static function interfaces. Minimally, provide a version that takes an array of arbitrary elements and a `Comparator` on those elements.
- Ensure that your implementation of insertion sort is stable.
- Write a version of insertion sort that uses sentinel elements.
- Write two versions of each algorithm: one that has a signature like

```
public static <E extends Comparable<E>> void sort(E [] data) ...
```

and one that has a signature like

```
public static <E> void sort(E [] data, Comparator<E> pred) ...
```

Here are some details to watch relating to particular sorts.

Shell sort Write a generic Shell sort that takes an increment sequence as an argument so that you can compare various different increment sequences. Test at least the increment sequences proposed by Shell (like 2^n), Hibbard (like $2^n - 1$), and Gonnet (where $k_{i+1} = \lfloor k_i / 2.2 \rfloor$).

Testing and Measuring

⇒ Test your sorting algorithms for *correctness*. That is, write code that verifies that your sorting algorithm produces (a) a sorted array, and (b) an array that is a permutation of the original. You may want to code some test routines to generate data other than permutations. For instance, an array randomly filled with 1's and 2's provides a good test for how your algorithms handle equal elements.

⇒ Test insertion sort sort for stability.

⇒ Time your sorting procedures for various different sized data collections.

All of your tests should be automated. That is, they should produce timing data files that are easy to use with your plotting software.

You must hand-in plots of running times versus size of data collection!

⇒ Show that your actual running times are consistent with the $\Theta(n)$-behavior for insertion sort and Shell sort.