UNBC Laboratory Assignment
Computer Science 200—Winter 2024

Utility Algorithms

Goals:

The goals of this laboratory exercise are three-fold:

- To practise using loop-invariants.
- To better understand the partition algorithm.

Due Date:

This assignment is due *Monday, 05 February* by the beginning of class.

Utilities

Your goal for this part of the assignment is to write a `Utilities` class that contains several small algorithms for working with a generic array. The interface is shown in Figure 1 on the following page.

With the exception of the `swap` method, each of these algorithms contains one or more loops. Write your code to contain a comment that describes the loop invariant, as shown in the sample `shuffle` algorithm.

There are four `isSorted` algorithms for determining if a section $[\ell, r)$ of an array `data` is sorted with respect to a comparator `c`. If $[\ell, r)$ is not supplied, it should be taken to be $[0, \texttt{data.length})$. If the comparator `c` is not supplied, it should be taken to be `Comparator.naturalOrder()`. Three of these algorithms should just call a different flavour of the algorithm. The fourth should document its loop invariant(s).

Finally, there is a `partition` algorithm. The intent is as follows: after executing

```
int m = partition(data, ell, arr, p);
```

the following should be true:

1. for $i \in [0, \ell)$, `data[i]` should be unchanged.
2. for $i \in [\ell, m)$, `p.test(data[i])` should be `false`.
3. for $i \in [m, r)$, `p.test(data[i])` should be `true`.
4. for $i \in [r, n)$, `data[i]` should be unchanged.

where ℓ = `ell`, r = `arr`, and n = `data.length`.

⇒ Explain why this `partition` algorithm is slightly different from the quick sort partition algorithm.

⇒ Submit your `Utilities.java` file via Moodle.

```
import java.util.function.Predicate ;
import java.util.Comparator ;

public class Utilities
{
  // a classic
  public static <E> void swap(E [] data, int i, int j)       {...}

  // tests for sortedness
  public static <E extends Comparable<? super E>>
  boolean isSorted(E [] data)                                {...}

  public static <E extends Comparable<? super E>>
  boolean isSorted(E [] data, int i, int j)                  {...}

  public static <E>
  boolean isSorted(E [] data, int i, int j, Comparator<? super E> c)
                                                             {...}

  public static <E>
  boolean isSorted(E [] data, Comparator<? super E> c)       {...}


  // a general purpose partition algorithm
  public static <E>
  int partition(E [] data, int ell, int arr, Predicate<E> p) {...}

  // a sample algorithm, with loop invariant
  public static <E> void shuffle(java.util.Random rnd, E [] data)
      {
      final int n = data.length ;
      for(int i=n;i>1;--i)
          {
          // Loop Invariant:
          //    The n-i rightmost elements are randomly selected
          //    with equal probabilities from the initial array.
          //    The range [0,i) contains the remaining elements.
          //
          //   [ ??? )[ rand )
          //   0      i      n
          swap(data,i-1,rnd.nextInt(i)) ;
          }
      }
}
```

Figure 1: `Utilities` class declaration