Matthew Jeffreys

CS 1632 – DELIVERABLE 4

PROPERTY BASED TESTING

Repository: https://github.com/mdj20/cs1632\_d4

For my fourth deliverable, I chose to test the behavioral properties of the sort method provided via Java’s Arrays library. A standard method for sorting arrays of many datatypes, Arrays.sort() will utilize either a quicksort or a mergesort algorithm, depending on whether it is testing primitives or objects. The specific properties that I chose to test were the order preservation of equal elements, often called stability, sorting accuracy, element consistency.

I first wrote a custom class (TestCLass.java) required for the stability testing, which simply implemented a scaler value used at the metric for sorting, and a unique numerical ID. A separate identifier was necessary in this case to differentiate equal value elements. Since Arrays.sort() utilizes the Comparable interface, I then wrote a the required compareTo() method. However my first attempt failed to compensate for the fact that an integer will change signs when its value overflows, a mistake that was made obvious when I began to sorting very large integers.

Subtracting two large negative integers will result in a positive integer, a piece of basic computer science knowledge that I should have thought about before test writing began. Since the compareTo() method works by returning either a negative or positive integer, this error prevented Arrays.sort() from completing. It was interesting to learn however, that Java has a specific exception for such an error, giving the programmer a head start when it comes time to debug.

To test for the stability itself, I simply placed substitute TestClasses throughout the array with duplicate values, but with different identifiers. I then searched for the duplicates, and made sure that they were all adjacent, and that their identifiers remained different.

For the sorted order test, I simply compared each element with the one adjacent, and asserted that the correct order was preserved. This was simple yet important because it defines how a sorting algorithm should behave at a very basic level.

I finally tested for element consistency, or tested that no elements could disappear, or be created in there process of sorting. To accomplish this, I recorded all identifiers in a set prior to sorting, and made sure they were still present afterwards. Since a set will not allow for duplicate values, I also checked that the number of elements contained were consistent at every testable interval.

In conclusion, this assignment proved to be a simple and interesting exercise in test writing. The main obstacle that I had to overcome involved the behavior of discrete integer values while implementing a commonly used java interface, stemming from a lack of forethought. However, this was how I expected the assignment, to unfold since I was testing a library that I would assume to be thoroughly tested already.

