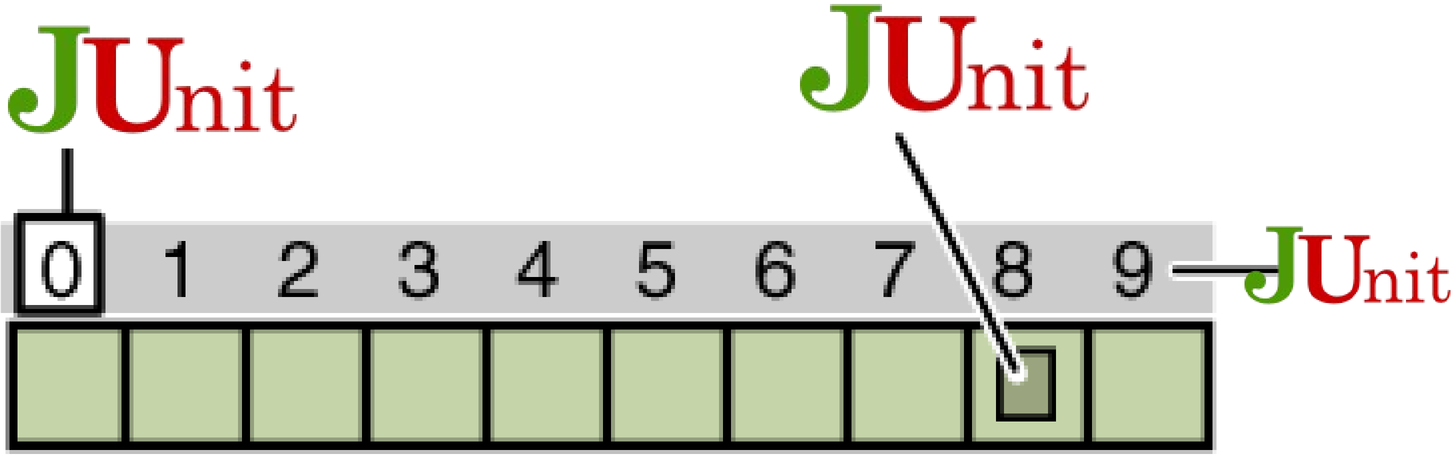
CS 1632

Deliverable 4: Property-Based Testing

Project: Testing Arrays.Sort with JUnit

Github Page: https://github.com/nkowdley/CS1632-Deliverable4



Neel Kowdley

Introduction:

For this deliverable, I had a choice between doing property based testing on Arrays.sort, or combinatorial testing on software of my choosing. I chose to do property based testing on Arrays.sort, as I found this a good chance to improve my Java skills, and I found that I was more interested in testing properties of arrays.

To start, I began by planning out my test cases. This began by deciding to randomly generate arrays and add them to an ArrayList in order to manage and maintain those arrays. I also decided to split up my tests, such that each property got its own test method, and that each run of my program would use the same arrays so that I did not have to regenerate the arrays before each test.

Once I had planned out my testing, I began by writing the code to generate random arrays. To do this, I used a random number generator, that would generate random sizes of arrays and fill those randomly sized arrays with random numbers. In order to ensure that the Arrays would not be regenerated before each test case, I used the annotation @BeforeClass before my createArrays() setup method.

Once I did this, I picked the properties I would use. I picked 4 properties to test:
1) An array must have the same length whether it is sorted or unsorted.
2) An array must have the same elements whether it is sorted or unsorted.
3) If there are two identical arrays and they are both sorted, the should have the same elements in the same order.
4) A sorted array must actually be sorted(smallest elements first, bigger elements last)

I then wrote JUnit Tests around these properties using asserts, creating a temporary version of each array in the ArrayList and then sorting the temporary array and verifying the property.

Once I had done this, I realized I did not explicitly test Edge Cases. I then added some edge cases by modifying the createArrays() setup method.

I thought of 8 edge cases to test:
1)Array of 0 objects
2)Array of 1 object
3)Array of arbitrary length filled with all negative values
4)Array of arbitrary length filled with the same number
5)Array of arbitrary length filled already sorted elements
6) Array of arbitrary length filled with Integer.MAX_VALUE
7) Array of arbitrary length filled with 0's
8) Array of arbitrary length filled elements sorted in reverse.

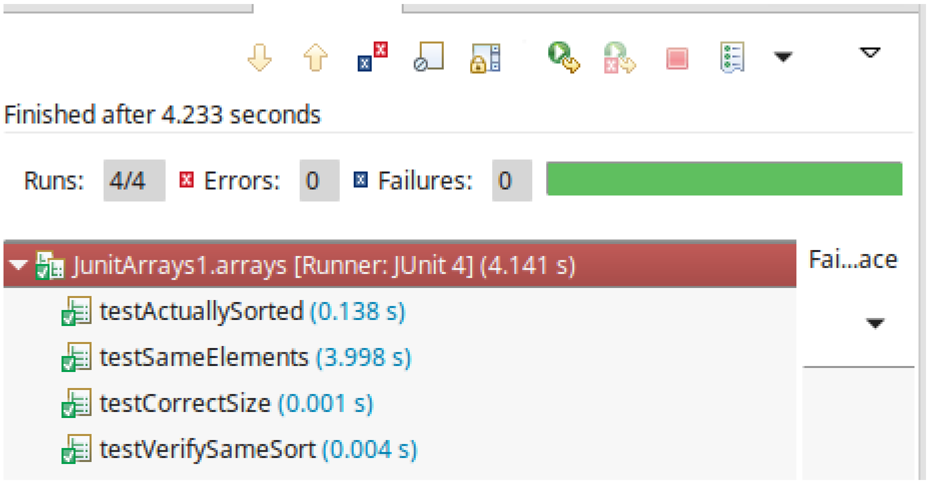
Then I had finished my test cases and they ran successfully.

Issues:

I ran into a number of issues while doing this project, but the biggest issue I faced was not being able to create large arrays. When I started this project, I had the idea that I would randomly generate arrays with sizes ranging from 0-Integer.MAX_VALUE(2^{31}-1). This turned out to be a large issue as Java cannot natively handle this amount of memory and I always got Out of Memory Exceptions. In order to combat these issue, I changed the max possible size to Integer.MAX_VALUE/100000 ((2^{31}-1)/100000).

Another issue that came up with this is that one of my test cases, testSameElements was too slow. It often crashed eclipse and I found that it was almost 20x slower than the 2nd slowest test. I tried some refactoring by counting each element in the sorted and unsorted array, but I found that I kept getting Out of Memory Exceptions, and that I was unable to create an index for negative numbers. I also found that changing the maximum size of an array had a significant impact on performance as my array sizes are huge. For this issue, I decided to keep using my slower version, as it was more reliable, and was much better for memory.

Screenshots:



Finished after 4.233 seconds

Runs: 4/4 Errors: 0 Failures: 0

JunitArrays1.arrays [Runner: JUnit 4] (4.141 s)

- testActuallySorted (0.138 s)
- testSameElements (3.998 s)
- testCorrectSize (0.001 s)
- testVerifySameSort (0.004 s)