Project Name: Homework 4

Unit Test Case Specification

Test Case ID: UnitTestingHomework_TC_1

Test Designed by: Tanner Huynh

Test Priority (Low/Medium/High): Low

Test Designed date: November 6, 2019

Module Name: UnitTestingHomework

Test Title: Square Root Method

Description: Test the method square root, which takes the square root of an int. Per the interface: double sqrt(double n).

Pre-conditions: N/A

Dependencies: N/A

Case	Given Input	Expected Result	Notes/Environment
1	0	0	Boundary on input; boundary on output
2	1	1	Next to boundary on the positive case side
3	-1	IllegalArgument-Exception	Next to boundary on the negative case side
4	2^{16} (65536)	256	Partition boundary 1 defined on input space. [0..65536]

5	$2^{16}+1$ (65537)	256.00	Left edge of partition boundary I defined on input space. [65537… 2,147,483,647]
6	$2^{31}-1$ (2,147,483,647)	46340.95	Right edge of range for second partition (MAX_INT).
7	4	2	Perfect square
8	37.515625	6.125	Rational root
9	2	1.414214	Irrational root. Accuracy measured to 10^{-6}
11	See table	See table	1000 numbers from each partition.

Post Conditions: N/A

Explanation:
The range of valid inputs for this function, `public double sqrt(int n),` is [`0`…`MAX_INT`]. It's a range, so we need to test on both sides of the range and inside the range. If you considered only one partition, you'd want a number smaller than zero, one larger than `MAX_INT`, and one somewhere in-between as a theoretical minimum. However, there aren't any int's bigger than `MAX_INT`, so that case is moot.

Test should run on the boundary and one past the boundary (or two or three or…). So, the test suite needs to include inputs of `0` and `-1`. It's good practice to sample around the boundary, especially on the positive case side. So the test suite would include `1`, and likely a few more, like `2`, `3`, and `4`. The test suite should include the other boundary, `MAX_INT.`

In the example, it partitioned the input space into two parts. There's no insight here other than it splits the input space into two parts. The test suite includes the boundary values on the input based on this partitioning.

After that, the next useful tests to add use domain knowledge to define test cases. Here, the example recognizes there are three special cases: perfect squares (inputs whose square roots are ints), roots that are rational, and roots that are irrational. The test suite should identify a sample of each of these. The example uses one example per type.

So the minimum total number of tests is nine. This isn't over-whelming for confidence, but it does provide direction for what a larger sampling should entail. After this, every input within a partition is equi-probable. At this point, it's basically guessing inputs and your only limits are the time and resources available to run tests. I'd add in a few hundred samples within each partition, likely creating a few "sub-partitions" as a heuristic – for example: small, medium, and large numbers.