Oracles used in my Quadrilateral Classifier Testing Nicole Morris

1. Fuzzing
   For this oracle, I generated a series of pseudo-random quadrilaterals. I say "pseudo-random" because I created it in a manner that ensured the creation of a set number (1000) of each of the possible basic types of quadrilaterals including squares, rectangles, rhombi, parallelograms, kites, and trapezoids. These were partially random as their sizes and some shape components were adjusted based on a random value but were all created using a seed (to ensure the creation of the desired shape) which limited their randomness. A series (2000) of actually random quadrilaterals was also generated using random integers to create 4 points and verifying the result was indeed a quadrilateral. Finally, I generated a series (2000) of random points containing errors that would trigger error 1 and error 2. Ultimately, this set of tests wasn't entirely random since I controlled the amount of randomness that was allowed. However, it generated a high number of the majority of conditions that the classifier could encounter. The result of this "fuzzing" process was not very exciting but was reassuring: no errors turned up.

2. Sanitizers
   For this test, I ran my classifier in xcode using the address sanitizer and undefined behavior sanitizer options. The address sanitizer helps identify potential memory bugs in the program. Using this can help prevent the program from crashing later during run time. The undefined behavior sanitizer helps identify bugs resulting from doing something in your program that really shouldn't be done such as pointer errors, accessing an array out of bounds, and many other things you may accidentally do that may or may not cause problems later. In the case of my program, these two helped me identify an error in how I was attempting to access data stored in a vector I was using in relationship to storing data to a null pointer location.

3. Assertions
   The last item of testing I did was using assertions to determine if my equations were producing the correct results. For example, the sum of all angles in a quadrilateral should be 360°. After I generated the angles for my quadrilateral, I added an assert to check the sum of these angles. I discovered errors as sometimes the results were correct for parallelograms but incorrect for kites. I'm still working to debug these errors. Somehow, based on tests using known data, the classifier still works despite this error.