To confirm that the actual running time of the algorithms matched with the efficiency class of the algorithms we ran our code with varying sizes of input. We used our data generation functions to create random input and circle input for varying values of n, and we used the same data for both the brute force and the quick hull algorithms. We looped through the quick hull and brute force algorithms enough times to see a clear difference between the two algorithms.

Testing the speed of the algorithm proved to be a challenge due to the extreme inefficiency of the brute force approach with large amounts of data. I attempted to run the brute force algorithm with 10,000 elements and waited 40 minutes before giving up. This led us to test with sizes 10, 100, and 1000. The table below shows the data we collected when running the tests.

Algorithm	Data Type	Number of Elements	Number of times run	Total Time taken(milliseconds)	Average Time taken(milliseconds)
Brute Force	Circular	10	2000	25	0
Brute Force	Random	10	2000	10	0
Quick Hull	Circular	10	2000	33	0
Quick Hull	Random	10	2000	8	0
Brute Force	Circular	100	200	262	1
Brute Force	Random	100	200	194	0
Quick Hull	Circular	100	200	42	0
Quick Hull	Random	100	200	5	0
Brute Force	Circular	1000	20	147371	7368
Brute Force	Random	1000	20	7368	64
Quick Hull	Circular	1000	20	5	0
Quick Hull	Random	1000	20	4	0
Quick Hull	Circular	10000	1	17	17
Quick Hull	Random	10000	1	4	4
Quick Hull	Circular	100000	1	79	79
Quick Hull	Random	100000	1	23	23

Looking at quick hull and comparing it with brute force with 10 elements shows very similar running speeds, but as soon as 100 elements are tested the difference becomes apparent. The rate of growth difference seems to be massive. With 100 elements of circular data brute force took 6 times longer to run than quick hull, and at 1000 elements of circular data brute force took 29,474 times longer to run than quick hull. While brute force took longer than 40 minutes to run before we quit the process on 10,000 elements quick hull took only 17 milliseconds.

We expected the data generated in circular form to perform in n squared time for quick hull because it is the worst case, and in n cubed time for brute force, so we assumed that the difference in running time would be a factor of n. With 100 elements the factor was only 6, and with 1,000 elements the factor was 29,474, and these factors are very different than the factors of 100 and 1,000 that we expected. This showed us that the performance of an algorithm with a specific efficiency class can vary drastically due to implementation details, and although constant factors are ignored in efficiency classes they make a big difference in practice.