INFO 6205 - Prof. Hillyard - Assignment 2

Name: Ashish Nevan Gade
NUID: 002889005
Github link: link to repo

A. Screenshots of Unit Tests

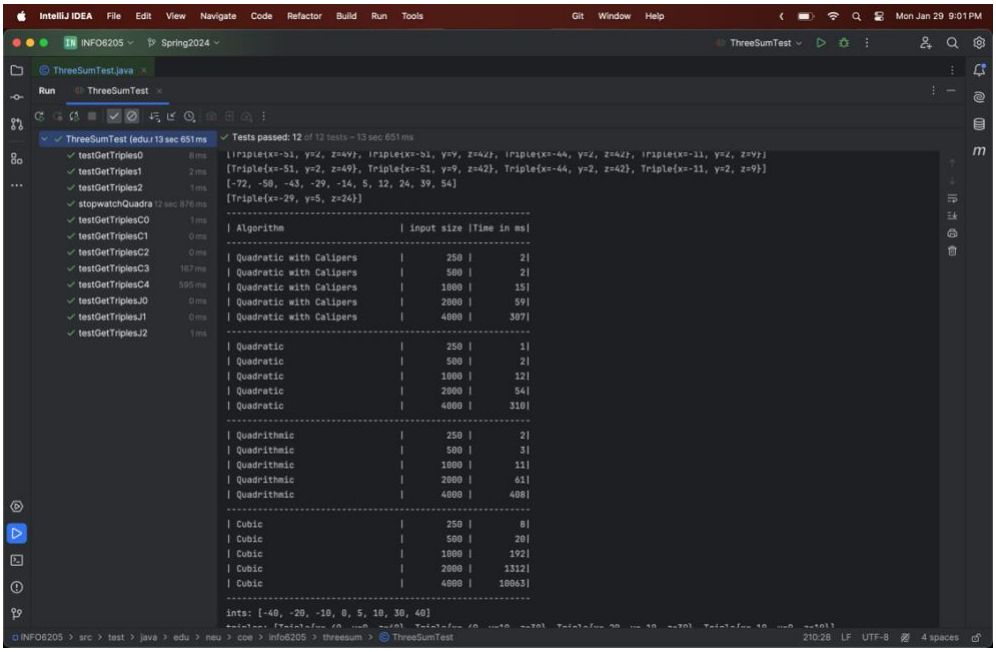

Figure 1: All unit tests pass and timings using StopWatch class

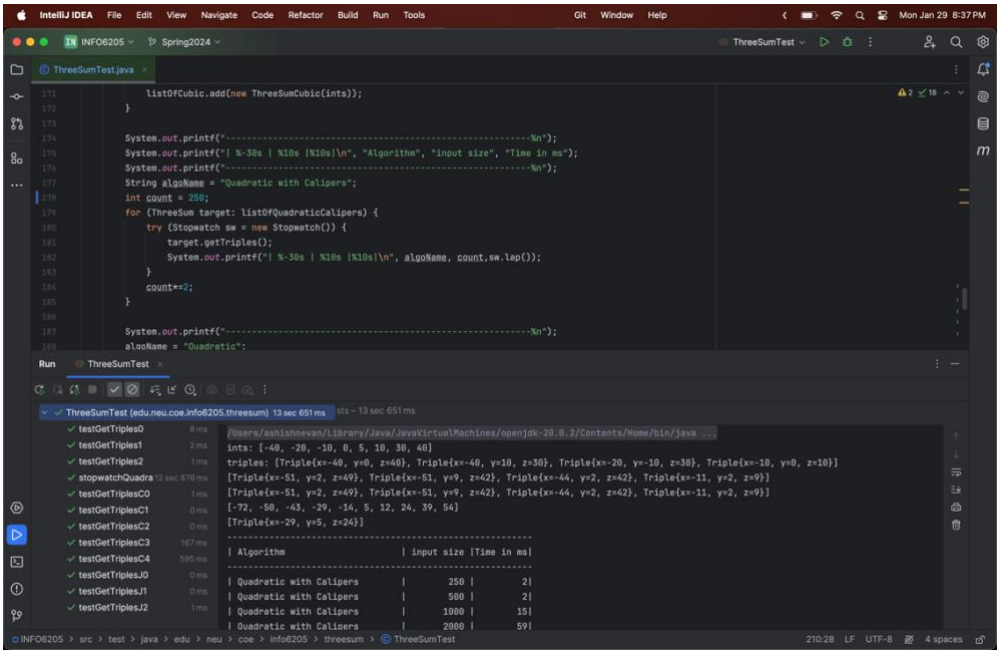

Figure 2: Code snippet from StopWatch unit test

B. Timing observations of each algorithm using doubling method

The time-to-execute was measured using the StopWatch class, creating an instance for each algorithm within a try block.

INPUT SIZE (N)	250	500	1000	2000	4000
ALGORITHM	TIME TAKEN (ms)				
Quadratic with Calipers	2	2	15	59	307
Quadratic	1	2	12	54	310
Quadrithmic	2	3	11	61	408
Cubic	8	20	192	1312	10063

Figure 3: Time-to-execute of each algorithm vs input size

C. Analysis of the quadratic methods

The three sum problem is a search problem to find 3 elements which sum up to 0. So, the brute force approach has a worst-case runtime of n^3. However, by sorting the input array we can solve it in worst case runtime n^2.

Quadratic with calipers:

We try to divide the search space into N sub-spaces by fixating one of the three elements (i), then go on to search for the remaining 2 elements by navigating the rest of the sorted array by incrementing/decrementing the 2-pointers (j = i+1, k = nums.length-1 and j<k) in worst case N. Another fact to note is that we decrease the search space for each valid triple found. The resulting algorithm has a worst-case runtime of n^2.

Quadratic:

The quadratic approach is like the "quadratic with calipers", however we fix the middle element (j) instead of first element. We then go on to search for the first and last elements (i, k) searching outward in the sorted array by incrementing/ decrementing the 2-pointers accordingly. The resulting algorithm also has a worst-case runtime of n^2.

Quadrithmic:

Unlike the quadratic and quadratic with calipers approaches, the Quadrithmic approach works by searching for a complement of each pair of elements in the array. The complexity for searching every possible pair in the array is n^2 and searching for a complement using binary search is log(n), resulting in a worst-case runtime of $n^2\log(n)$.