

Assignment 2: 3-Sum

- (a) evidence (screenshot) of your unit tests running (try to show the actual unit test code as well as the green strip);

The screenshot shows an IDE with the following components:

- Project Explorer:** Lists various test classes including BaseHelperTest, Benchmarks, HelperTest, InstrumentedHelperTest, OrderedArrayTest, SortTest, SymbolTable, ThreeSumTest, TwoSumTest, UnionFind, and Util.
- Editor:** Displays the source code for ThreeSumTest.java. The visible code includes two test methods:


```

      @Test
      public void testGetTriplesC3() {
          Supplier<int[]> intsSupplier = new Source(N: 1000, M: 1000).intsSupplier(safetyFactor: 10);
          int[] ints = intsSupplier.get();
          ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
          Triple[] triplesQuadratic = target.getTriples();
          Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
          assertEquals(triplesCubic.length, triplesQuadratic.length);
      }

      @Test
      public void testGetTriplesC4() {
          Supplier<int[]> intsSupplier = new Source(N: 1500, M: 1000).intsSupplier(safetyFactor: 10);
          int[] ints = intsSupplier.get();
          ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
      
```
- Run Console:** Shows the execution results of the tests. The output indicates that 12 tests passed in 5 seconds 384 ms. The tests include:
 - testGetTriples0 (67 ms)
 - testGetTriples1 (13 ms)
 - testGetTriples2 (6 ms)
 - testGetTriplesC0 (4 ms)
 - testGetTriplesC1 (11 ms)
 - testGetTriplesC2 (3 ms)
 - testGetTriplesC3 (798 ms)
 - testGetTriplesC4 (1 sec 826 ms)
 - testGetTriplesJ0 (0 ms)
 - testGetTriplesJ1 (0 ms)
 - testGetTriplesJ2 (2 ms)
 - testTimingObserv (2 sec 654 ms)

- (b) a spreadsheet showing your timing observations--using the doubling method for at least five values of N--for each of the algorithms (include cubic); Timing should be performed either with an actual stopwatch (e.g. your iPhone) or using the Stopwatch class in the repository.

N = 100 - Quadratic Time: 0 ms
 N = 100 - Cubic Time: 1 ms
 N = 100 - Quadratic with Calipers Time: 0 ms
 N = 100 - Quadrithmic Time: 4 ms
 N = 200 - Quadratic Time: 1 ms
 N = 200 - Cubic Time: 3 ms
 N = 200 - Quadratic with Calipers Time: 0 ms
 N = 200 - Quadrithmic Time: 2 ms
 N = 400 - Quadratic Time: 0 ms
 N = 400 - Cubic Time: 23 ms
 N = 400 - Quadratic with Calipers Time: 0 ms
 N = 400 - Quadrithmic Time: 7 ms
 N = 800 - Quadratic Time: 4 ms
 N = 800 - Cubic Time: 181 ms
 N = 800 - Quadratic with Calipers Time: 4 ms
 N = 800 - Quadrithmic Time: 10 ms

N = 1600 - Quadratic Time: 5 ms
N = 1600 - Cubic Time: 1447 ms
N = 1600 - Quadratic with Calipers Time: 15 ms
N = 1600 - Quadrithmic Time: 43 ms

(c) your brief explanation of why the quadratic method(s) work.

The quadratic method(s) takes advantage of the properties of a sorted array. Sorting the array allows for a systematic exploration of the solution space, facilitating the identification of triples with a sum of zero. The approach involves using two pointers that start at the extremes of the remaining array and move towards each other. The quadratic method divides the solution space into N sub-spaces, where each subspace corresponds to a fixed value for the middle index of the three values in a potential triple. By fixing the middle index, the problem is reduced to finding pairs of indices (i , k) that sum to the negation of the value at the middle index (j). The quadratic method has an overall time complexity of $O(N^2)$, where N is the size of the input array. This approach results in a time complexity that is quadratic in the size of the input array, making it a practical and efficient solution for the ThreeSum problem.