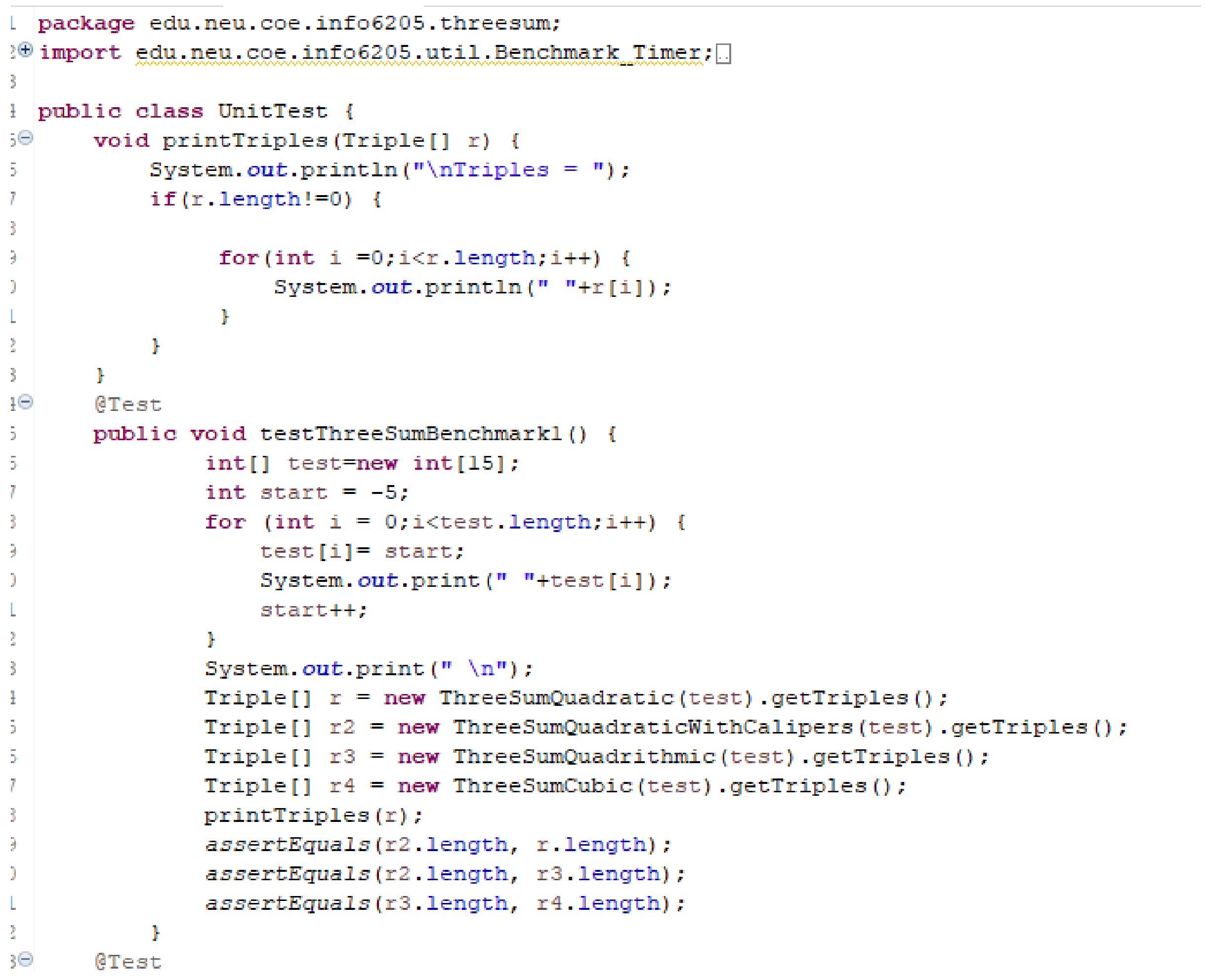
INFO6205 Assignment 2 threesum questions

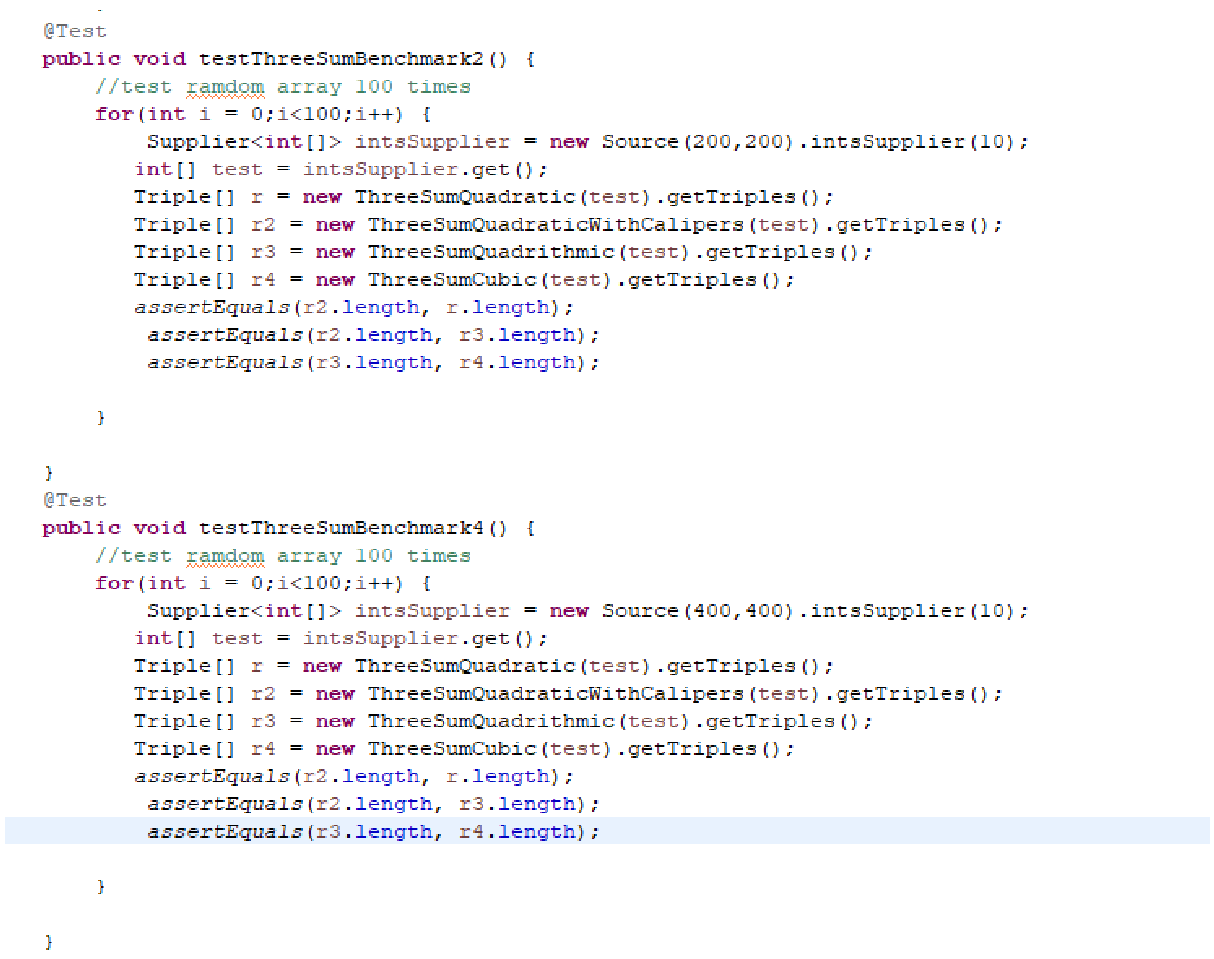
Ruizhe Zeng

1. Unit Test Result with Calipers method

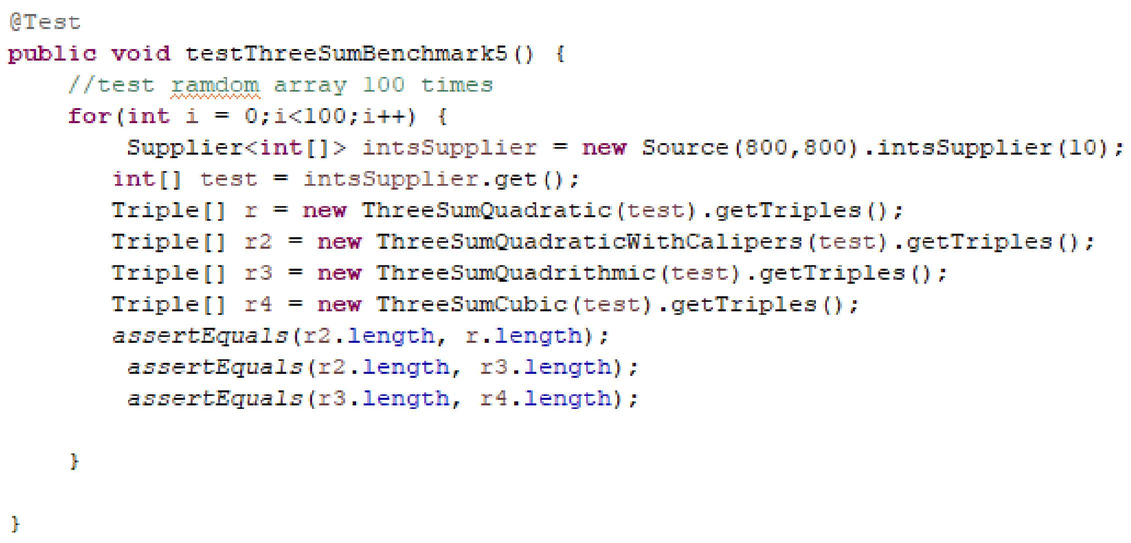
Self Test



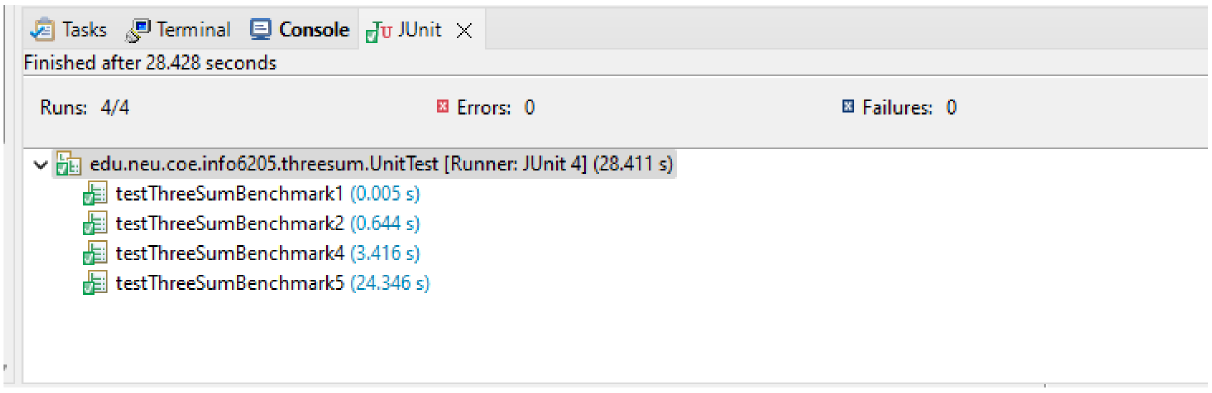
```java
package edu.neu.coe.info6205.threesum;
import edu.neu.coe.info6205.util.Benchmark_Timer;

public class UnitTest {
    void printTriples(Triple[] r) {
        System.out.println("\nTriples = ");
        if(r.length!=0) {

             for(int i =0;i<r.length;i++) {
                 System.out.println(" "+r[i]);
             }
        }
    }
    @Test
    public void testThreeSumBenchmark1() {
            int[] test=new int[15];
            int start = -5;
            for (int i = 0;i<test.length;i++) {
                test[i]= start;
                System.out.print(" "+test[i]);
                start++;
            }
            System.out.print(" \n");
            Triple[] r = new ThreeSumQuadratic(test).getTriples();
            Triple[] r2 = new ThreeSumQuadraticWithCalipers(test).getTriples();
            Triple[] r3 = new ThreeSumQuadrithmic(test).getTriples();
            Triple[] r4 = new ThreeSumCubic(test).getTriples();
            printTriples(r);
            assertEquals(r2.length, r.length);
            assertEquals(r2.length, r3.length);
            assertEquals(r3.length, r4.length);
        }
    @Test
```



```
@Test
public void testThreeSumBenchmark2() {
    //test ramdom array 100 times
    for(int i = 0;i<100;i++) {
         Supplier<int[]> intsSupplier = new Source(200,200).intsSupplier(10);
        int[] test = intsSupplier.get();
        Triple[] r = new ThreeSumQuadratic(test).getTriples();
        Triple[] r2 = new ThreeSumQuadraticWithCalipers(test).getTriples();
        Triple[] r3 = new ThreeSumQuadrithmic(test).getTriples();
        Triple[] r4 = new ThreeSumCubic(test).getTriples();
        assertEquals(r2.length, r.length);
         assertEquals(r2.length, r3.length);
         assertEquals(r3.length, r4.length);

    }

}
@Test
public void testThreeSumBenchmark4() {
    //test ramdom array 100 times
    for(int i = 0;i<100;i++) {
         Supplier<int[]> intsSupplier = new Source(400,400).intsSupplier(10);
        int[] test = intsSupplier.get();
        Triple[] r = new ThreeSumQuadratic(test).getTriples();
        Triple[] r2 = new ThreeSumQuadraticWithCalipers(test).getTriples();
        Triple[] r3 = new ThreeSumQuadrithmic(test).getTriples();
        Triple[] r4 = new ThreeSumCubic(test).getTriples();
        assertEquals(r2.length, r.length);
         assertEquals(r2.length, r3.length);
         assertEquals(r3.length, r4.length);

    }

}
```

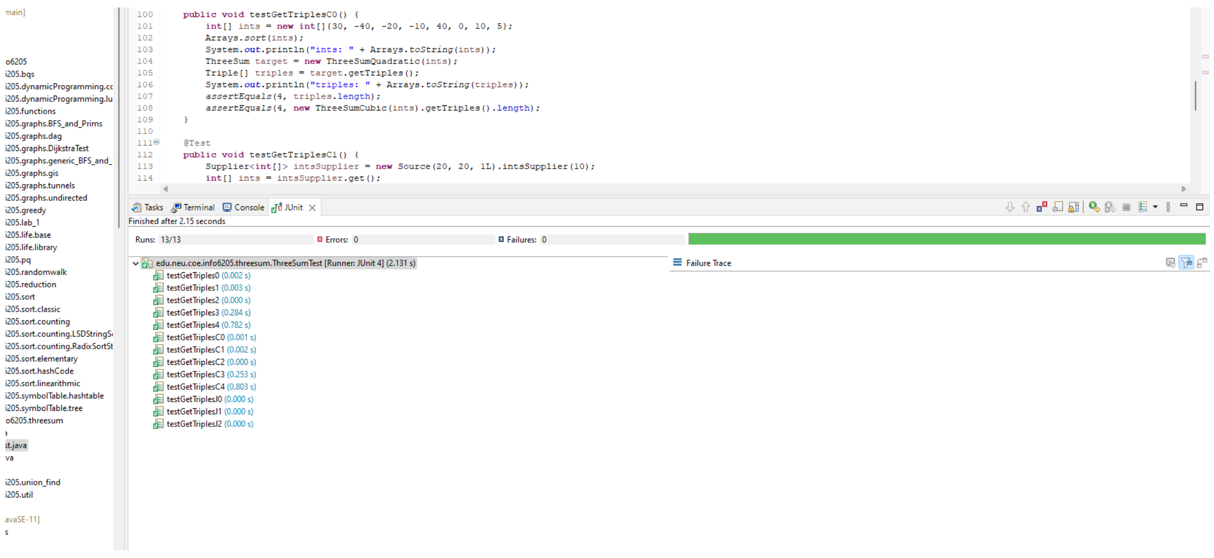


```
@Test
public void testThreeSumBenchmark5() {
    //test ramdom array 100 times
    for(int i = 0;i<100;i++) {
         Supplier<int[]> intsSupplier = new Source(800,800).intsSupplier(10);
        int[] test = intsSupplier.get();
        Triple[] r = new ThreeSumQuadratic(test).getTriples();
        Triple[] r2 = new ThreeSumQuadraticWithCalipers(test).getTriples();
        Triple[] r3 = new ThreeSumQuadrithmic(test).getTriples();
        Triple[] r4 = new ThreeSumCubic(test).getTriples();
        assertEquals(r2.length, r.length);
         assertEquals(r2.length, r3.length);
         assertEquals(r3.length, r4.length);

    }

}
```



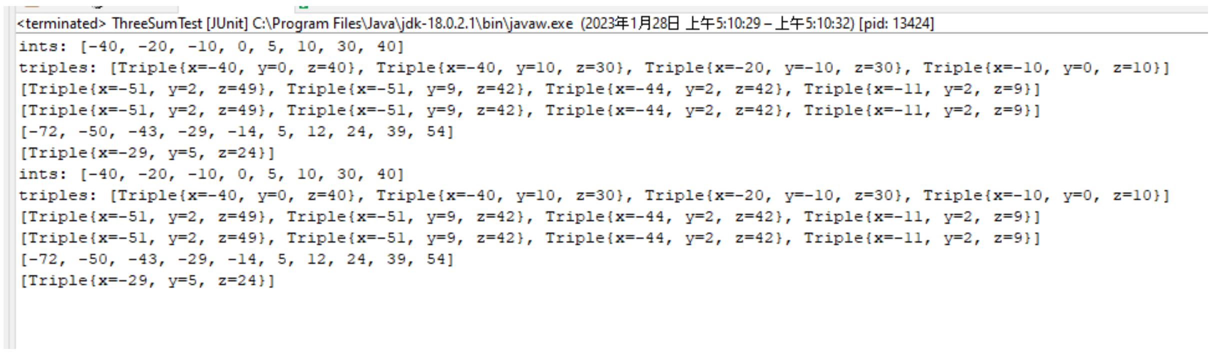
```
Tasks  Terminal  Console  JUnit
Finished after 28.428 seconds

Runs:  4/4          Errors:  0          Failures:  0

edu.neu.coe.info6205.threesum.UnitTest [Runner: JUnit 4] (28.411 s)
    testThreeSumBenchmark1 (0.005 s)
    testThreeSumBenchmark2 (0.644 s)
    testThreeSumBenchmark4 (3.416 s)
    testThreeSumBenchmark5 (24.346 s)
```

Professor ’s Unit Test



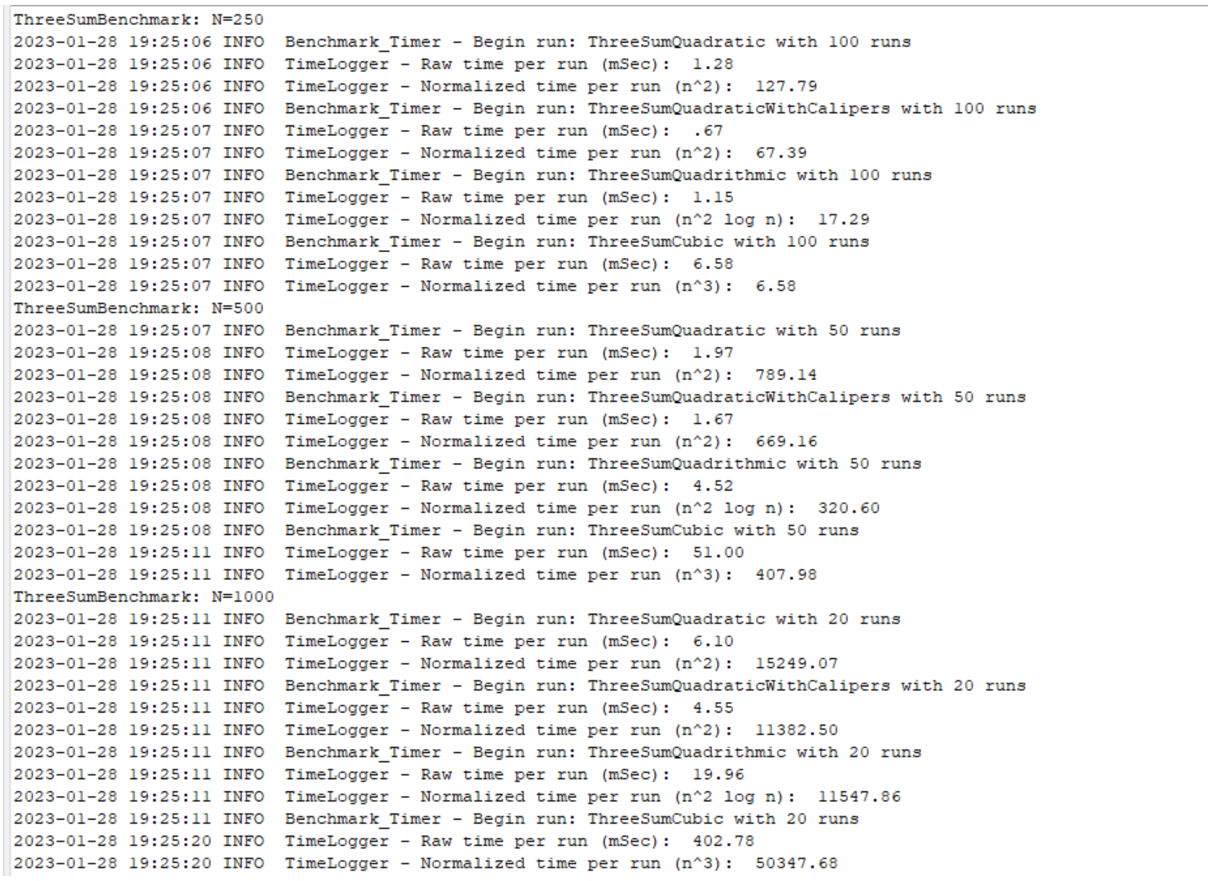
```java
    public void testGetTriplesC0() {
        int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
        Arrays.sort(ints);
        System.out.println("ints: " + Arrays.toString(ints));
        ThreeSum target = new ThreeSumQuadratic(ints);
        Triple[] triples = target.getTriples();
        System.out.println("triples: " + Arrays.toString(triples));
        assertEquals(4, triples.length);
        assertEquals(4, new ThreeSumCubic(ints).getTriples().length);
    }

    @Test
    public void testGetTriplesC1() {
        Supplier<int[]> intsSupplier = new Source(20, 20, 1L).intsSupplier(10);
        int[] ints = intsSupplier.get();
```

```
Tasks  Terminal  Console  JUnit
Finished after 2.15 seconds

Runs:  13/13          Errors:  0          Failures:  0

edu.neu.coe.info6205.threesum.ThreeSumTest [Runner: JUnit 4] (2.131 s)        Failure Trace
    testGetTriples0 (0.002 s)
    testGetTriples1 (0.003 s)
    testGetTriples2 (0.000 s)
    testGetTriples3 (0.284 s)
    testGetTriples4 (0.782 s)
    testGetTriplesC0 (0.001 s)
    testGetTriplesC1 (0.002 s)
    testGetTriplesC2 (0.000 s)
    testGetTriplesC3 (0.253 s)
    testGetTriplesC4 (0.803 s)
    testGetTriplesJ0 (0.000 s)
    testGetTriplesJ1 (0.000 s)
    testGetTriplesJ2 (0.000 s)
```

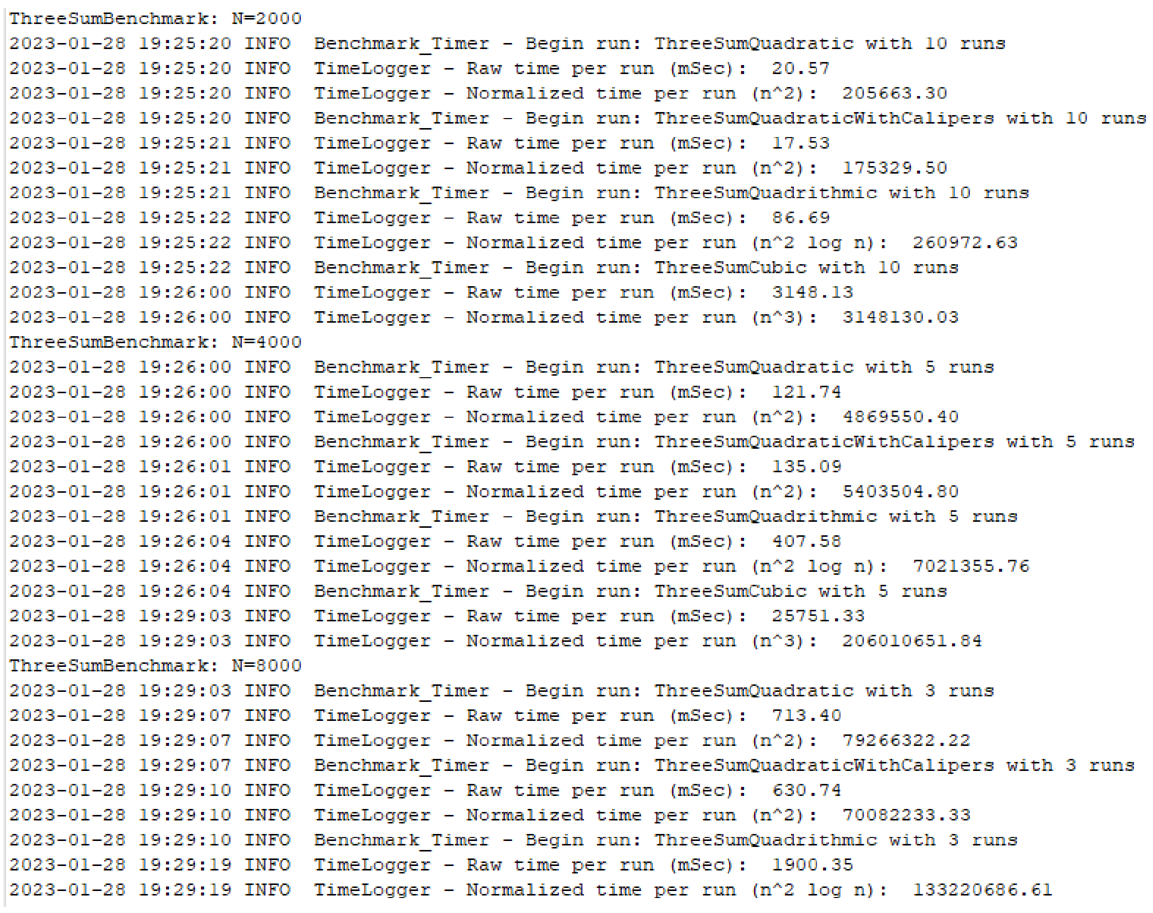


```
<terminated> ThreeSumTest [JUnit] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe  (2023年1月28日 上午5:10:29 – 上午5:10:32) [pid: 13424]
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
```

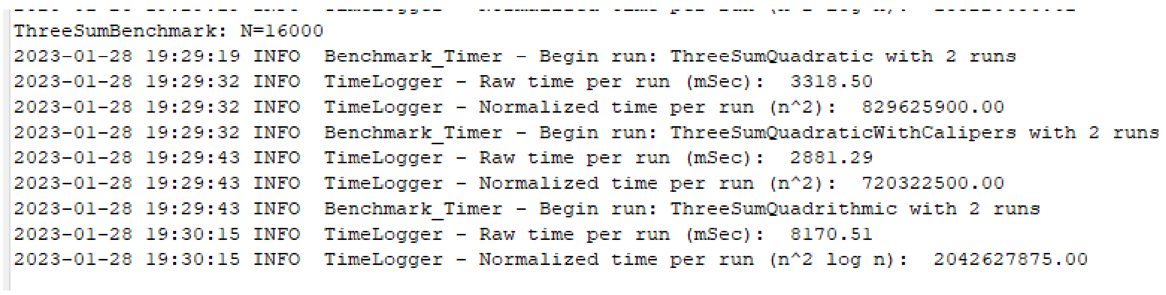
2. **Data observation**
    a. **Data from program output**



```
ThreeSumBenchmark: N=250
2023-01-28 19:25:06 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 100 runs
2023-01-28 19:25:06 INFO  TimeLogger - Raw time per run (mSec):  1.28
2023-01-28 19:25:06 INFO  TimeLogger - Normalized time per run (n^2):  127.79
2023-01-28 19:25:06 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 100 runs
2023-01-28 19:25:07 INFO  TimeLogger - Raw time per run (mSec):  .67
2023-01-28 19:25:07 INFO  TimeLogger - Normalized time per run (n^2):  67.39
2023-01-28 19:25:07 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 100 runs
2023-01-28 19:25:07 INFO  TimeLogger - Raw time per run (mSec):  1.15
2023-01-28 19:25:07 INFO  TimeLogger - Normalized time per run (n^2 log n):  17.29
2023-01-28 19:25:07 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 100 runs
2023-01-28 19:25:07 INFO  TimeLogger - Raw time per run (mSec):  6.58
2023-01-28 19:25:07 INFO  TimeLogger - Normalized time per run (n^3):  6.58
ThreeSumBenchmark: N=500
2023-01-28 19:25:07 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 50 runs
2023-01-28 19:25:08 INFO  TimeLogger - Raw time per run (mSec):  1.97
2023-01-28 19:25:08 INFO  TimeLogger - Normalized time per run (n^2):  789.14
2023-01-28 19:25:08 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 50 runs
2023-01-28 19:25:08 INFO  TimeLogger - Raw time per run (mSec):  1.67
2023-01-28 19:25:08 INFO  TimeLogger - Normalized time per run (n^2):  669.16
2023-01-28 19:25:08 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 50 runs
2023-01-28 19:25:08 INFO  TimeLogger - Raw time per run (mSec):  4.52
2023-01-28 19:25:08 INFO  TimeLogger - Normalized time per run (n^2 log n):  320.60
2023-01-28 19:25:08 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 19:25:11 INFO  TimeLogger - Raw time per run (mSec):  51.00
2023-01-28 19:25:11 INFO  TimeLogger - Normalized time per run (n^3):  407.98
ThreeSumBenchmark: N=1000
2023-01-28 19:25:11 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 20 runs
2023-01-28 19:25:11 INFO  TimeLogger - Raw time per run (mSec):  6.10
2023-01-28 19:25:11 INFO  TimeLogger - Normalized time per run (n^2):  15249.07
2023-01-28 19:25:11 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 20 runs
2023-01-28 19:25:11 INFO  TimeLogger - Raw time per run (mSec):  4.55
2023-01-28 19:25:11 INFO  TimeLogger - Normalized time per run (n^2):  11382.50
2023-01-28 19:25:11 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 20 runs
2023-01-28 19:25:11 INFO  TimeLogger - Raw time per run (mSec):  19.96
2023-01-28 19:25:11 INFO  TimeLogger - Normalized time per run (n^2 log n):  11547.86
2023-01-28 19:25:11 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 20 runs
2023-01-28 19:25:20 INFO  TimeLogger - Raw time per run (mSec):  402.78
2023-01-28 19:25:20 INFO  TimeLogger - Normalized time per run (n^3):  50347.68
```

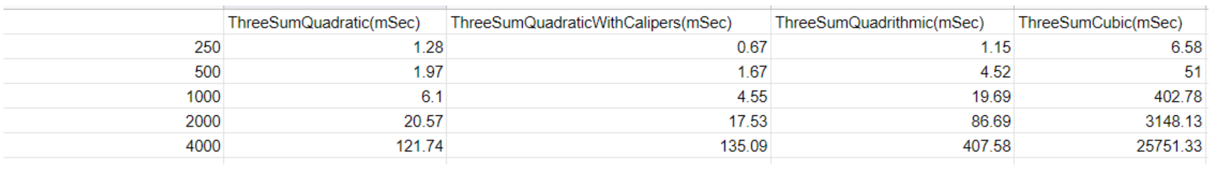


```
ThreeSumBenchmark: N=2000
2023-01-28 19:25:20 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 10 runs
2023-01-28 19:25:20 INFO  TimeLogger - Raw time per run (mSec):  20.57
2023-01-28 19:25:20 INFO  TimeLogger - Normalized time per run (n^2):  205663.30
2023-01-28 19:25:20 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 10 runs
2023-01-28 19:25:21 INFO  TimeLogger - Raw time per run (mSec):  17.53
2023-01-28 19:25:21 INFO  TimeLogger - Normalized time per run (n^2):  175329.50
2023-01-28 19:25:21 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 10 runs
2023-01-28 19:25:22 INFO  TimeLogger - Raw time per run (mSec):  86.69
2023-01-28 19:25:22 INFO  TimeLogger - Normalized time per run (n^2 log n):  260972.63
2023-01-28 19:25:22 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 10 runs
2023-01-28 19:26:00 INFO  TimeLogger - Raw time per run (mSec):  3148.13
2023-01-28 19:26:00 INFO  TimeLogger - Normalized time per run (n^3):  3148130.03
ThreeSumBenchmark: N=4000
2023-01-28 19:26:00 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 5 runs
2023-01-28 19:26:00 INFO  TimeLogger - Raw time per run (mSec):  121.74
2023-01-28 19:26:00 INFO  TimeLogger - Normalized time per run (n^2):  4869550.40
2023-01-28 19:26:00 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 5 runs
2023-01-28 19:26:01 INFO  TimeLogger - Raw time per run (mSec):  135.09
2023-01-28 19:26:01 INFO  TimeLogger - Normalized time per run (n^2):  5403504.80
2023-01-28 19:26:01 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 5 runs
2023-01-28 19:26:04 INFO  TimeLogger - Raw time per run (mSec):  407.58
2023-01-28 19:26:04 INFO  TimeLogger - Normalized time per run (n^2 log n):  7021355.76
2023-01-28 19:26:04 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 5 runs
2023-01-28 19:29:03 INFO  TimeLogger - Raw time per run (mSec):  25751.33
2023-01-28 19:29:03 INFO  TimeLogger - Normalized time per run (n^3):  206010651.84
ThreeSumBenchmark: N=8000
2023-01-28 19:29:03 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 3 runs
2023-01-28 19:29:07 INFO  TimeLogger - Raw time per run (mSec):  713.40
2023-01-28 19:29:07 INFO  TimeLogger - Normalized time per run (n^2):  79266322.22
2023-01-28 19:29:07 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 3 runs
2023-01-28 19:29:10 INFO  TimeLogger - Raw time per run (mSec):  630.74
2023-01-28 19:29:10 INFO  TimeLogger - Normalized time per run (n^2):  70082233.33
2023-01-28 19:29:10 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 3 runs
2023-01-28 19:29:19 INFO  TimeLogger - Raw time per run (mSec):  1900.35
2023-01-28 19:29:19 INFO  TimeLogger - Normalized time per run (n^2 log n):  133220686.61
```



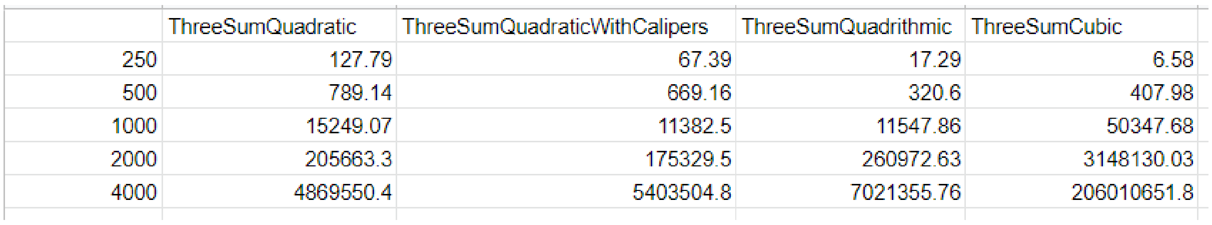
```
ThreeSumBenchmark: N=16000
2023-01-28 19:29:19 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 2 runs
2023-01-28 19:29:32 INFO  TimeLogger - Raw time per run (mSec):  3318.50
2023-01-28 19:29:32 INFO  TimeLogger - Normalized time per run (n^2):  829625900.00
2023-01-28 19:29:32 INFO  Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 2 runs
2023-01-28 19:29:43 INFO  TimeLogger - Raw time per run (mSec):  2881.29
2023-01-28 19:29:43 INFO  TimeLogger - Normalized time per run (n^2):  720322500.00
2023-01-28 19:29:43 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 2 runs
2023-01-28 19:30:15 INFO  TimeLogger - Raw time per run (mSec):  8170.51
2023-01-28 19:30:15 INFO  TimeLogger - Normalized time per run (n^2 log n):  2042627875.00
```

b. Spreadsheet for data



	ThreeSumQuadratic(mSec)	ThreeSumQuadraticWithCalipers(mSec)	ThreeSumQuadrithmic(mSec)	ThreeSumCubic(mSec)
250	1.28	0.67	1.15	6.58
500	1.97	1.67	4.52	51
1000	6.1	4.55	19.69	402.78
2000	20.57	17.53	86.69	3148.13
4000	121.74	135.09	407.58	25751.33

Raw time per run (mSec)



	ThreeSumQuadratic	ThreeSumQuadraticWithCalipers	ThreeSumQuadrithmic	ThreeSumCubic
250	127.79	67.39	17.29	6.58
500	789.14	669.16	320.6	407.98
1000	15249.07	11382.5	11547.86	50347.68
2000	205663.3	175329.5	260972.63	3148130.03
4000	4869550.4	5403504.8	7021355.76	206010651.8

Normalized time per run

3. threesum question explanation

a. ThreeSumQuadratic

Imagine in an **SORTED** array, each time we pick one of the elements from the array and call it **Mid**. To get all the elements picked it requires O(n). Now we choose and element at its left side and call it **Left**, and then pick a element at Mid's right side and call it **Right.** Notice Left,Mid,Right should be in range of array all the time. After that we calculate the **Sum** = Mid+Left +Left+Right. If Sum equals 0. We find what we want, and move Left to the left for one index. If Sum < 0, then we know we cannot find any Sum = 0 if we continually move Left to left, it will only get smaller. So we only need to move Right to the right in one index. And check for the Sum again. If Sum > 0, we know we cannot get any result if we move Right to the right, because it will only get larger. So we only need to move Left to the left for one index and check for Sum. Notice it may have more than one solution when we check for the Mid. That is why we need to move Left to the left in one index,when we find a Sum = 0. It will stop when Left or Right is no longer in the range of array. Therefore we ignore many invalid candidates, to do this for each Mid we need up to O(n) times. So it requires O(n^2) time.

b. **ThreeSumQuadraticWithCalipers**

This method is very similar to the previous one. This time we going to pick **Left** instead of Mid from array each time, so it also requires O(n) time to pick all the candidates. Then we are going to pick the element at Left's right side and call it **Mid**. After that we are going to pick the **Last** element in the array and call it **Right**. We want to make sure Left < Mid < Right and they are in the range of the array all the time. If Sum = Left + Mid + Right = 0 , we find the result and move Left to its right for one index. If Sum > 0, we know since Left is the smallest number we can get, we need to move Right to the left .If Sum < 0, since the Right is the largest number we can get, we need to move Mid to the right. This process several time until the statement Left< Mid<Right is no longer true or any of them is not in the range of array. To check all the Mid and Right it takes up to O(n) for each Mid. So it is also O(n^2), same as the previous method. However for each Mid it will only check at most (N - Mid) elements, but the previous method is going to check N elements if it cannot find any solution, so this method is more unlikely to get to the worst case, so technically it will be a little faster than the previous method.