Program Structures and Algorithms

Spring 2023(SEC – 1)

NAME: Venkat Pavan Munaganti

NUID: 002722397

Task: Assignment 2: 3-SUM

Solve 3-SUM using the *Quadrithmic*, *Quadratic*, and (bonus point) *quadraticWithCalipers* approaches, as shown in skeleton code in the repository. There are hints at the end of Lesson 2.5 Entropy.

There are also hints in the comments of the existing code. There are a number of unit tests which you should be able to run successfully.

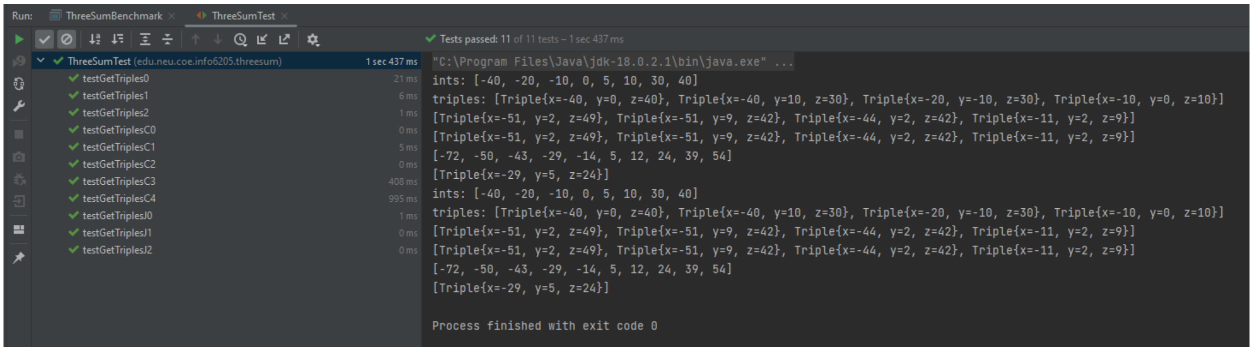
Submit (in your own repository--see instructions elsewhere--include the source code and the unit tests of course):

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

(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.

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

(a) Unit tests:

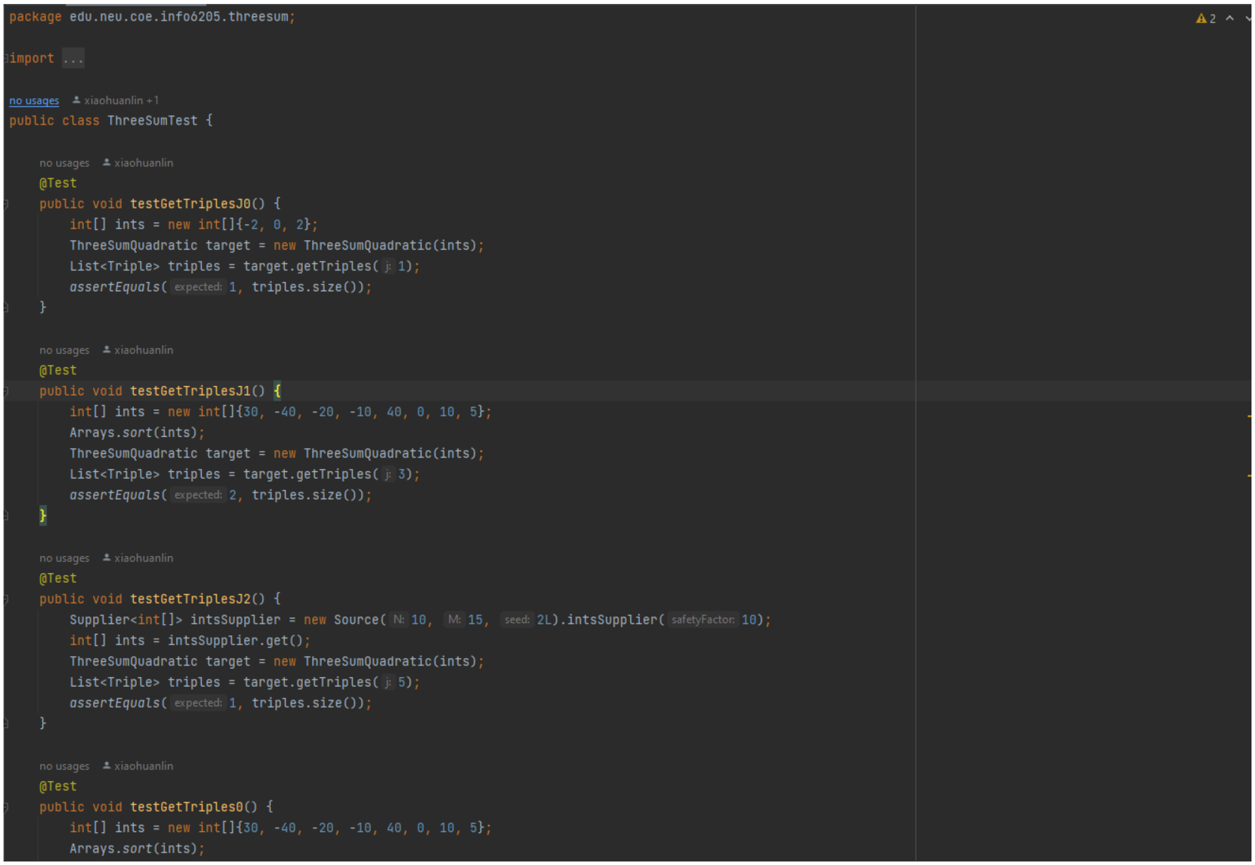


Run: ThreeSumBenchmark ThreeSumTest

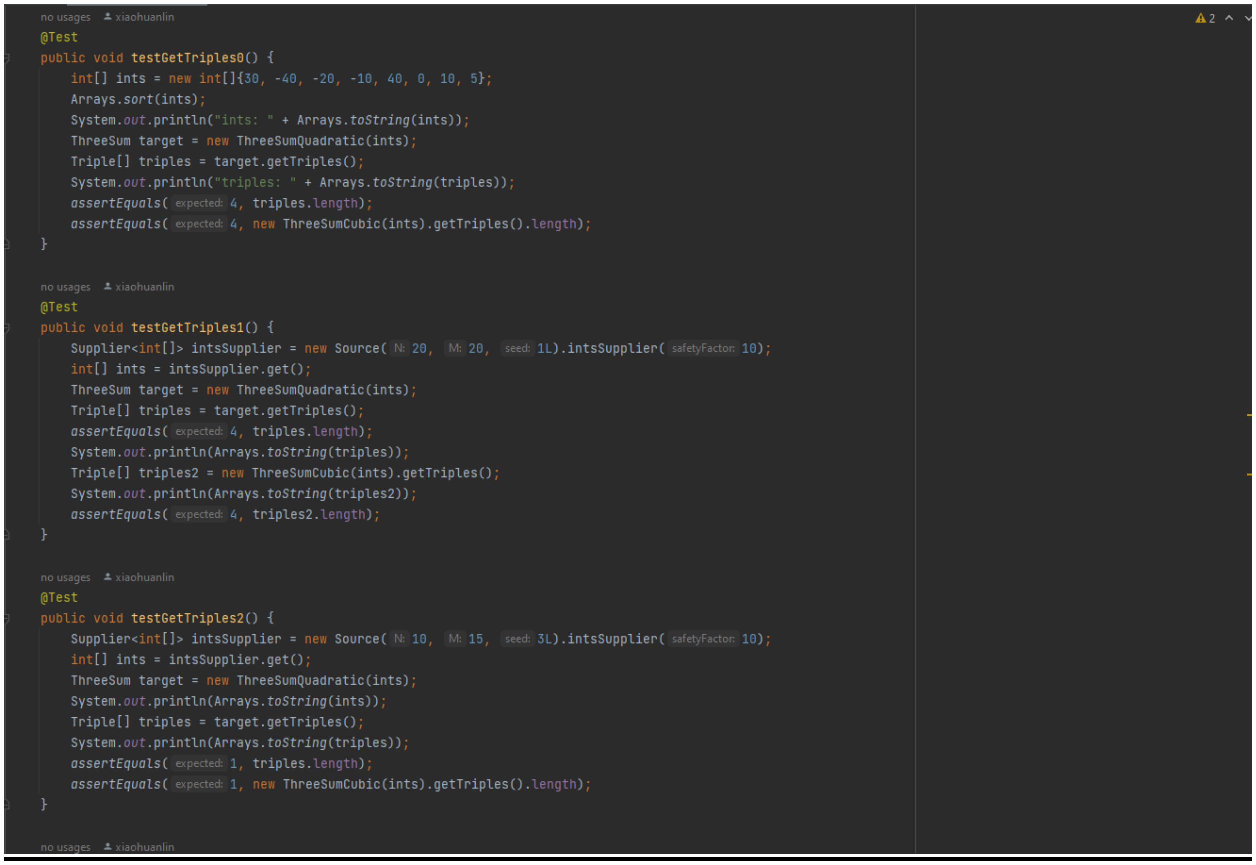
✓ Tests passed: 11 of 11 tests – 1 sec 437 ms

```
✓ ThreeSumTest (edu.neu.coe.info6205.threesum)   1 sec 437 ms
    ✓ testGetTriples0      21 ms
    ✓ testGetTriples1       6 ms
    ✓ testGetTriples2       1 ms
    ✓ testGetTriplesC0      0 ms
    ✓ testGetTriplesC1      5 ms
    ✓ testGetTriplesC2      0 ms
    ✓ testGetTriplesC3    408 ms
    ✓ testGetTriplesC4    995 ms
    ✓ testGetTriplesJ0      1 ms
    ✓ testGetTriplesJ1      0 ms
    ✓ testGetTriplesJ2      0 ms
```

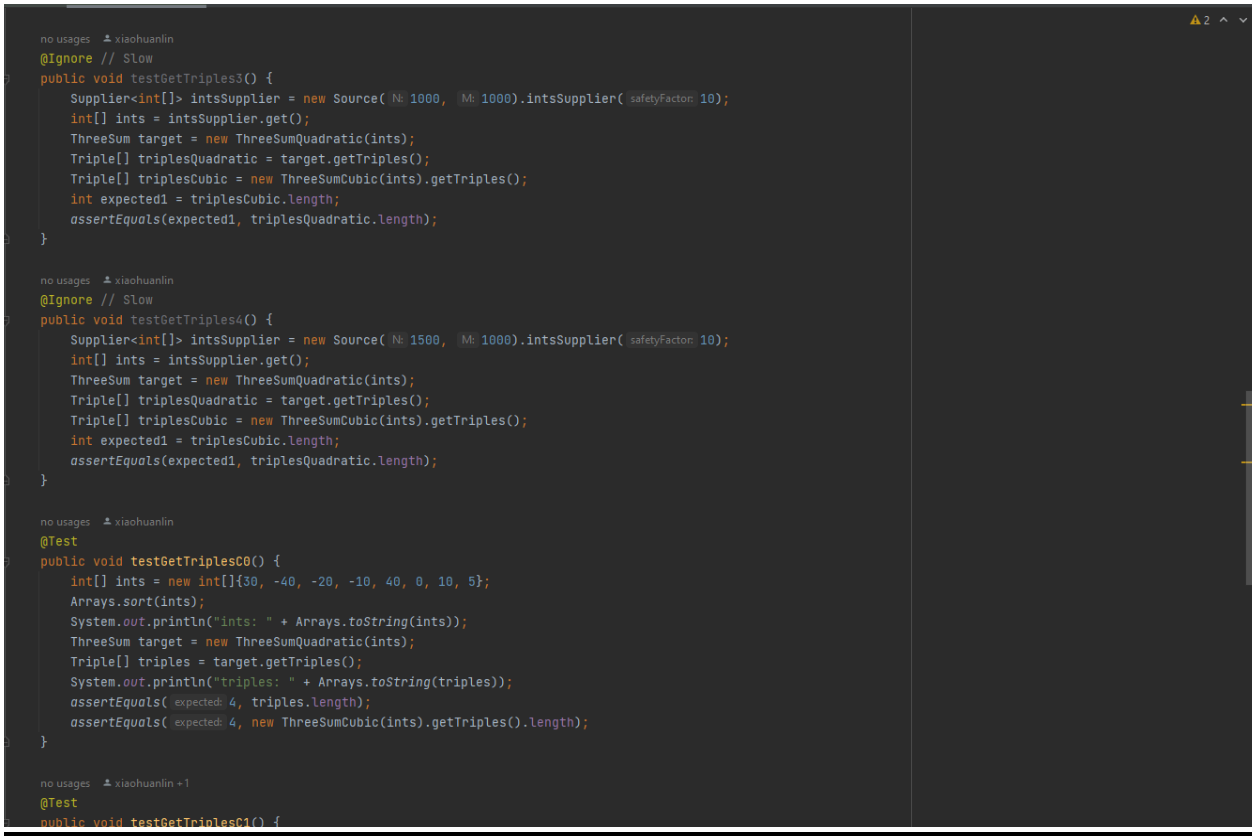
```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, 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}]

Process finished with exit code 0
```



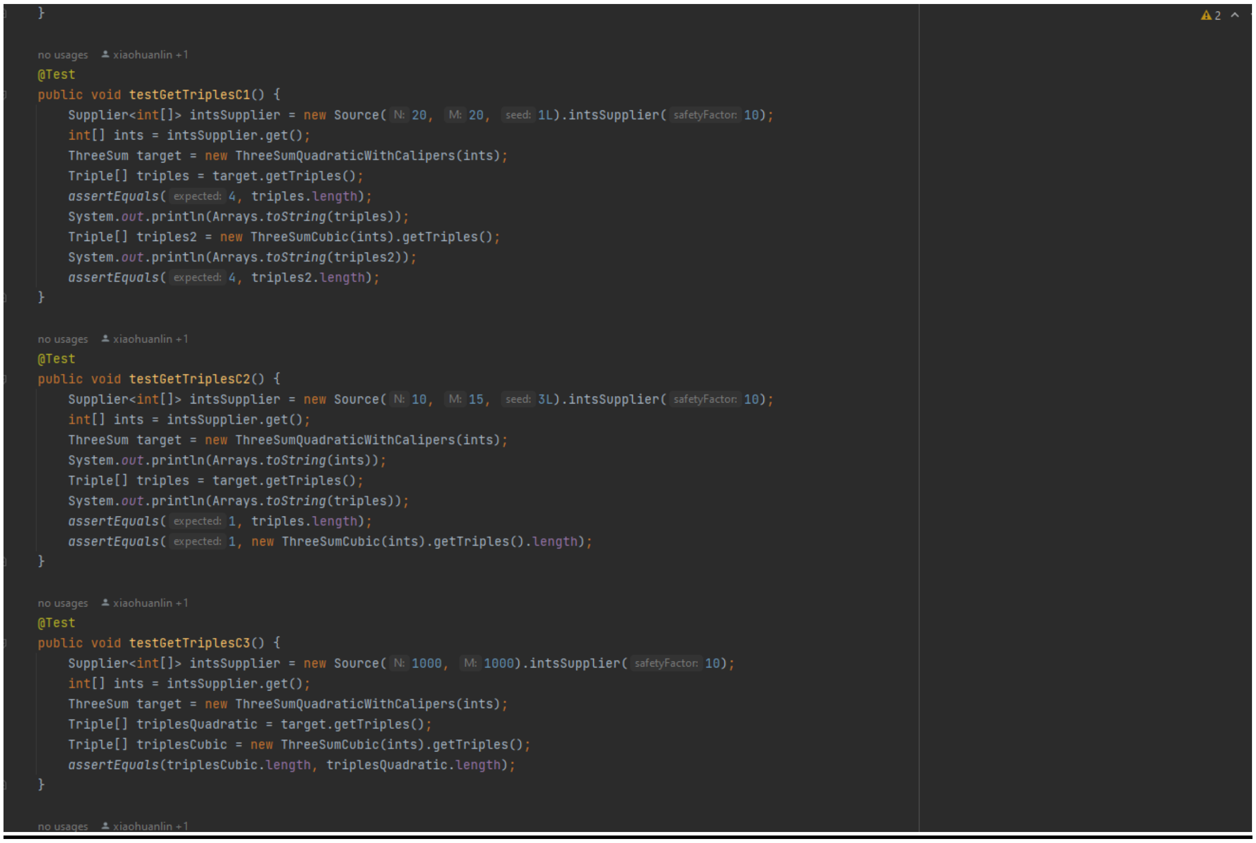
```java
package edu.neu.coe.info6205.threesum;

import ...

no usages  xiaohuanlin +1
public class ThreeSumTest {

    no usages  xiaohuanlin
    @Test
    public void testGetTriplesJ0() {
        int[] ints = new int[]{-2, 0, 2};
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples( j: 1);
        assertEquals( expected: 1, triples.size());
    }

    no usages  xiaohuanlin
    @Test
    public void testGetTriplesJ1() {
        int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
        Arrays.sort(ints);
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples( j: 3);
        assertEquals( expected: 2, triples.size());
    }

    no usages  xiaohuanlin
    @Test
    public void testGetTriplesJ2() {
        Supplier<int[]> intsSupplier = new Source( N: 10,  M: 15,  seed: 2L).intsSupplier( safetyFactor: 10);
        int[] ints = intsSupplier.get();
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples( j: 5);
        assertEquals( expected: 1, triples.size());
    }

    no usages  xiaohuanlin
    @Test
    public void testGetTriples0() {
        int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
        Arrays.sort(ints);
```



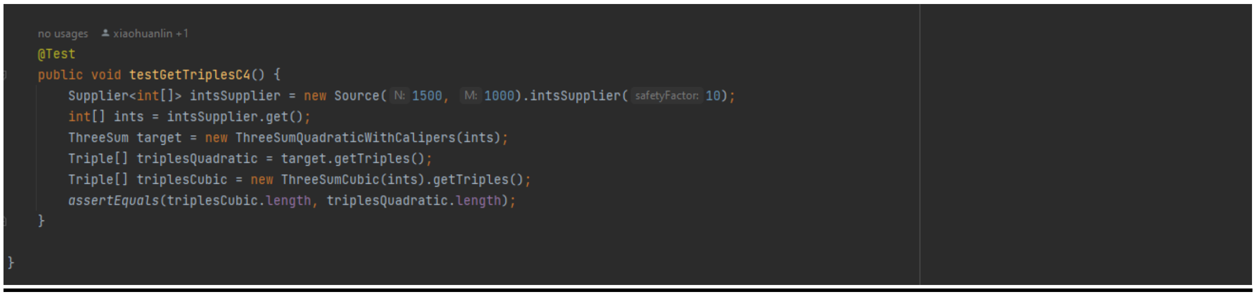
```
no usages  xiaohuanlin
@Test
public void testGetTriples0() {
    int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
    Arrays.sort(ints);
    System.out.println("ints: " + Arrays.toString(ints));
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triples = target.getTriples();
    System.out.println("triples: " + Arrays.toString(triples));
    assertEquals( expected: 4, triples.length);
    assertEquals( expected: 4, new ThreeSumCubic(ints).getTriples().length);
}

no usages  xiaohuanlin
@Test
public void testGetTriples1() {
    Supplier<int[]> intsSupplier = new Source( N: 20,  M: 20,  seed: 1L).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triples = target.getTriples();
    assertEquals( expected: 4, triples.length);
    System.out.println(Arrays.toString(triples));
    Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
    System.out.println(Arrays.toString(triples2));
    assertEquals( expected: 4, triples2.length);
}

no usages  xiaohuanlin
@Test
public void testGetTriples2() {
    Supplier<int[]> intsSupplier = new Source( N: 10,  M: 15,  seed: 3L).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadratic(ints);
    System.out.println(Arrays.toString(ints));
    Triple[] triples = target.getTriples();
    System.out.println(Arrays.toString(triples));
    assertEquals( expected: 1, triples.length);
    assertEquals( expected: 1, new ThreeSumCubic(ints).getTriples().length);
}

no usages  xiaohuanlin
```



```
no usages  xiaohuanlin
@Ignore // Slow
public void testGetTriples3() {
    Supplier<int[]> intsSupplier = new Source( N: 1000,  M: 1000).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triplesQuadratic = target.getTriples();
    Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
    int expected1 = triplesCubic.length;
    assertEquals(expected1, triplesQuadratic.length);
}

no usages  xiaohuanlin
@Ignore // Slow
public void testGetTriples4() {
    Supplier<int[]> intsSupplier = new Source( N: 1500,  M: 1000).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triplesQuadratic = target.getTriples();
    Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
    int expected1 = triplesCubic.length;
    assertEquals(expected1, triplesQuadratic.length);
}

no usages  xiaohuanlin
@Test
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( expected: 4, triples.length);
    assertEquals( expected: 4, new ThreeSumCubic(ints).getTriples().length);
}

no usages  xiaohuanlin +1
@Test
public void testGetTriplesC1() {
```



```
    }

    no usages   xiaohuanlin +1
    @Test
    public void testGetTriplesC1() {
        Supplier<int[]> intsSupplier = new Source( N: 20,  M: 20,  seed: 1L).intsSupplier( safetyFactor: 10);
        int[] ints = intsSupplier.get();
        ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
        Triple[] triples = target.getTriples();
        assertEquals( expected: 4, triples.length);
        System.out.println(Arrays.toString(triples));
        Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
        System.out.println(Arrays.toString(triples2));
        assertEquals( expected: 4, triples2.length);
    }

    no usages   xiaohuanlin +1
    @Test
    public void testGetTriplesC2() {
        Supplier<int[]> intsSupplier = new Source( N: 10,  M: 15,  seed: 3L).intsSupplier( safetyFactor: 10);
        int[] ints = intsSupplier.get();
        ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
        System.out.println(Arrays.toString(ints));
        Triple[] triples = target.getTriples();
        System.out.println(Arrays.toString(triples));
        assertEquals( expected: 1, triples.length);
        assertEquals( expected: 1, new ThreeSumCubic(ints).getTriples().length);
    }

    no usages   xiaohuanlin +1
    @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);
    }

    no usages   xiaohuanlin +1
```



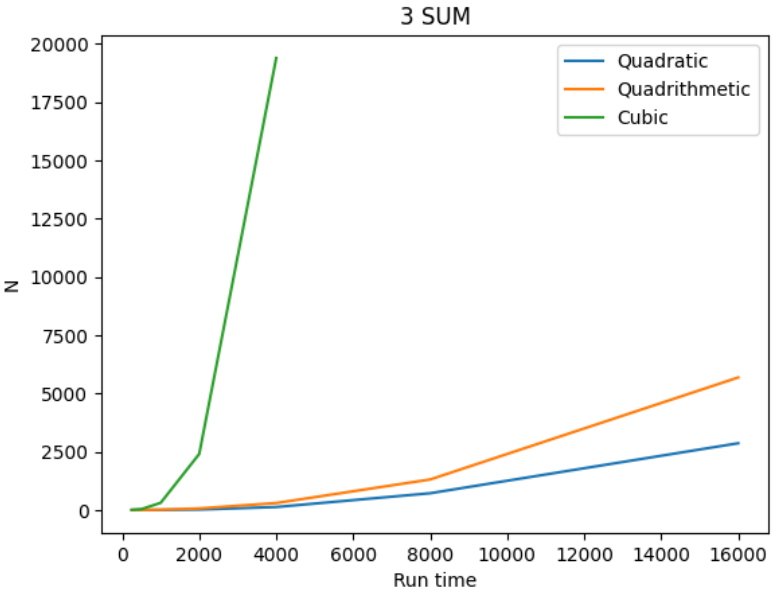
```
    no usages   xiaohuanlin +1
    @Test
    public void testGetTriplesC4() {
        Supplier<int[]> intsSupplier = new Source( N: 1500,  M: 1000).intsSupplier( safetyFactor: 10);
        int[] ints = intsSupplier.get();
        ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
        Triple[] triplesQuadratic = target.getTriples();
        Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
        assertEquals(triplesCubic.length, triplesQuadratic.length);
    }

}
```

(b) Time observations:

N	Quadratic		Quadrithmic		Cubic	
	(Milli secs)	Log ratio	(Milli secs)	Log ratio	(Milli secs)	Log ratio
250	0.39		0.35		5.16	
	6.24		0.7		0.33	
500	1.6	2.036525876	2.14	2.612183969	38.38	2.894911741
	6.4		0.95		0.31	
1000	4.55	1.50779464	11.95	2.481327916	309.05	3.009413648
	4.55		1.2		0.31	
2000	19.2	2.077167861	66.4	2.474172623	2404.4	2.959764755
	4.8		1.51		0.3	
4000	128	2.736965594	298.2	2.167025111	19386	3.011266326
	8		1.56		0.3	

8000	719	2.48984796	1309	2.134112935		
	11.23		1.58			
16000	2863	1.993463995	5685	2.118695252		
	11.18		1.59			

Below is graph plotted for the benchmarking data between the quadratic methods and the cubic method



(c) Why quadratic method(s) works?

The quadratic methods solve the 3-SUM problem by iterating through all the elements in the input array and for each element, using another nested loop to check if the sum of the current element and any two other elements in the array equals to 0. This method has a time complexity of $\mathbf{0}(\boldsymbol{n}^{\mathbf{2}})$, where $\boldsymbol{n}$ is the number of elements in the input array. This is because for each element in the array, the nested loop will also iterate through all n elements.

When the array is sorted, we can use two pointers to solve the problem in quadratic time. In normal quadratic solution we are dividing the solution space into two parts such that the middle number i.e., ***j,*** is fixed and we use two pointers starting from $\boldsymbol{j} - \mathbf{1}$ and $\boldsymbol{j} + \mathbf{1}$ moving outward to find a pair of numbers $(\boldsymbol{i}, \boldsymbol{k})$ such that $\boldsymbol{a}[\boldsymbol{i}] + \boldsymbol{a}[\boldsymbol{j}] + \boldsymbol{a}[\boldsymbol{k}] = \mathbf{0}$. We are using two loops. One loop keeps the $\boldsymbol{j}$ fixed, i.e., diving the solution space into two sub problems. This loop runs $\boldsymbol{n}$ times diving the solution space into two parts for each value of $\boldsymbol{n}$. Another loop uses two pointers, left pointer searches for a value in left and right pointer searches the value in right sub problem such that sum of left $(\boldsymbol{a}[\boldsymbol{i}])$, fixed middle $(\boldsymbol{a}[\boldsymbol{j}])$ and right $(\boldsymbol{a}[\boldsymbol{k}])$ is 0. These two pointer loops run n times for each value of n, hence the run time is $\mathbf{0}(\boldsymbol{n}^{\mathbf{2}})$

Quadraticwithcalipers uses similar technique to above quadratic solution, except in this version the initial value is fixed i.e., $\boldsymbol{i}$. We use two pointers, one starting from $\boldsymbol{i} + \mathbf{1}$, and one starting from the end of the array. These two pointers move towards the center search for a pair $(\boldsymbol{j}, \boldsymbol{k})$ such that $\boldsymbol{a}[\boldsymbol{i}] + \boldsymbol{a}[\boldsymbol{j}] + \boldsymbol{a}[\boldsymbol{k}] = \mathbf{0}$**.** Since the array is sorted, we can use the ordering to increment and decrement the value of $\boldsymbol{i}$ and $\boldsymbol{j}$ respectively.