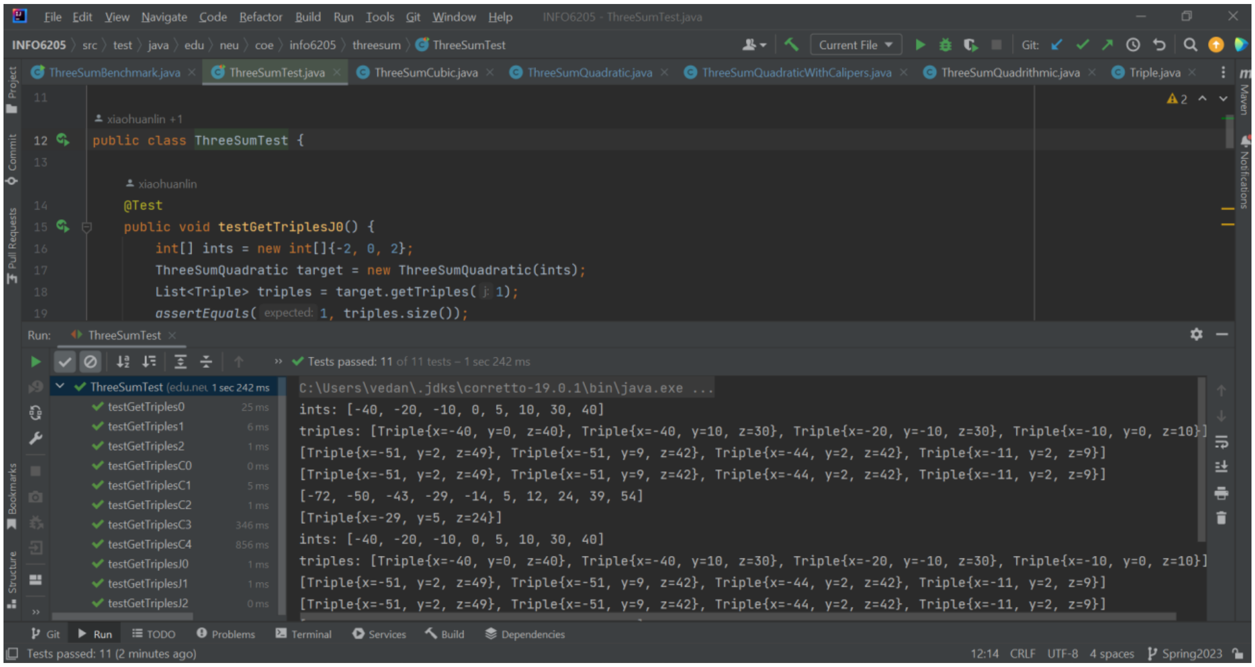
PSA – Assignment 2
3 - Sum

Name: Vedantini Gaikwad
NUID: 002998254

- **Relationship between Quadratic, Quadrithmic, QuadraticWithCalipers, and Cubic**:
  Each of these solutions has a different time complexity, with Quadratic having the lowest complexity at N^2, Quadrithmic having a complexity of N^2logN, and Cubic having the highest complexity at N^3. As the size of N increases, the Quadrithmic solution becomes less efficient compared to the Quadratic solution. Another alternative is Quadratic with callipers, which utilizes two pointers and has similar performance to the standard Quadratic solution. However, the Cubic solution is the least efficient of all the options as its complexity increases exponentially with the size of N.

- **Unit Test Evidence:**


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
INFO6205 - ThreeSumTest.java
INFO6205 〉 src 〉 test 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 threesum 〉 ThreeSumTest
Current File
Git:
ThreeSumBenchmark.java
ThreeSumTest.java
ThreeSumCubic.java
ThreeSumQuadratic.java
ThreeSumQuadraticWithCalipers.java
ThreeSumQuadrithmic.java
Triple.java
xiaohuanlin +1
xiaohuanlin
Run:
ThreeSumTest
Git
Run
TODO
Problems
Terminal
Services
Build
Dependencies
12:14 CRLF UTF-8 4 spaces Spring2023

```
public class ThreeSumTest {

    @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());
```

✓ Tests passed: 11 of 11 tests – 1 sec 242 ms

```
✓ ThreeSumTest (edu.neu 1 sec 242 ms
    ✓ testGetTriples0      25 ms
    ✓ testGetTriples1       6 ms
    ✓ testGetTriples2       1 ms
    ✓ testGetTriplesC0      0 ms
    ✓ testGetTriplesC1      5 ms
    ✓ testGetTriplesC2      1 ms
    ✓ testGetTriplesC3    346 ms
    ✓ testGetTriplesC4    856 ms
    ✓ testGetTriplesJ0      1 ms
    ✓ testGetTriplesJ1      1 ms
    ✓ testGetTriplesJ2      0 ms
```

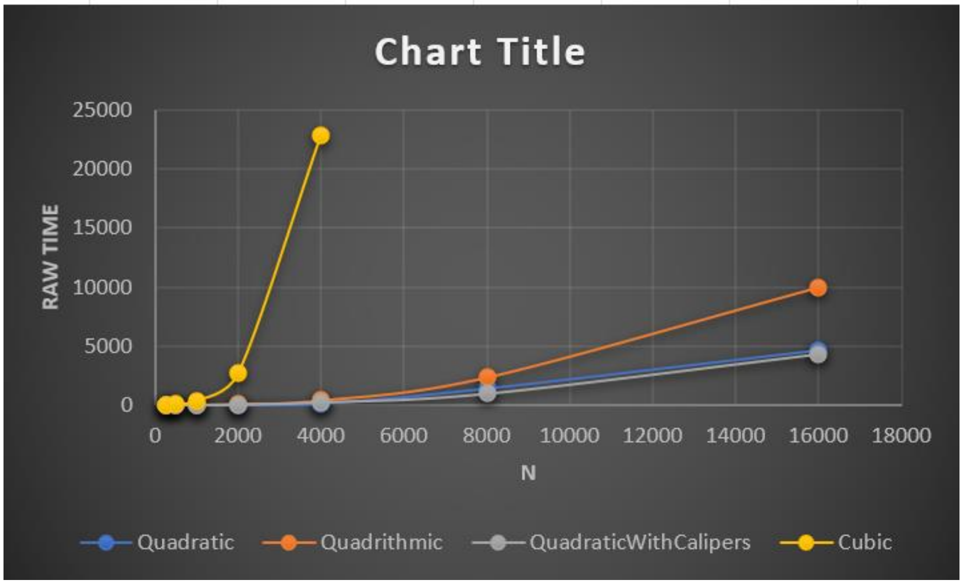
```
C:\Users\vedan\.jdks\corretto-19.0.1\bin\java.exe ...
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, 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}]
```

Tests passed: 11 (2 minutes ago)

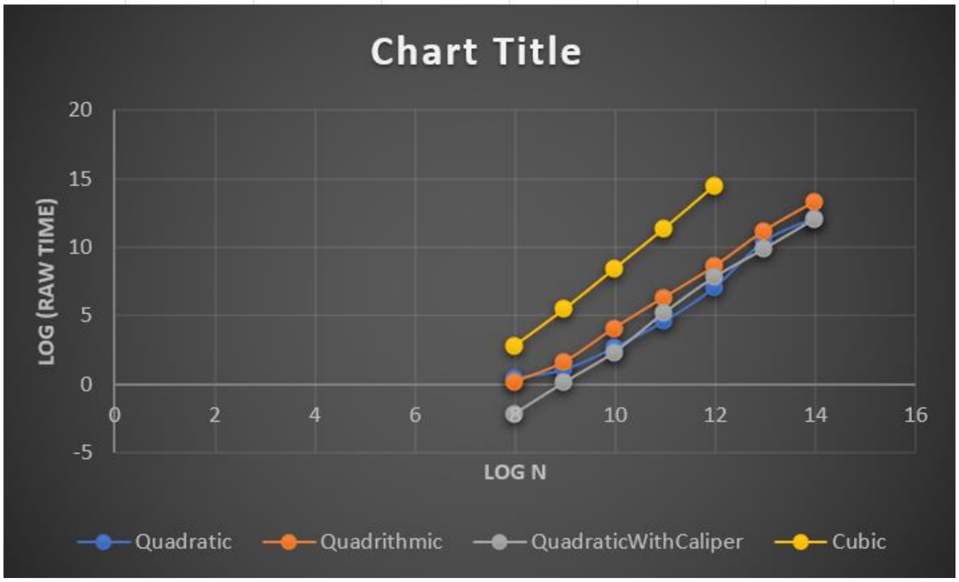
- Evidence:

		Quadratic		Quadrithmic		QuadraticWithCalipers		Cubic	
N	log N	ms	log	ms	log	ms	log	ms	log
250	7.965784	1.43	0.516015	1.13	0.176323	0.22	-2.18442	6.97	2.801159
500	8.965784	2.18	1.124328	3.14	1.650765	1.1	0.137504	44.66	5.480911
1000	9.965784	6.6	2.722466	17.2	4.104337	5.05	2.336283	344.75	8.429407
2000	10.96578	25	4.643856	81.5	6.348728	37.6	5.232661	2666.9	11.38095
4000	11.96578	136.6	7.093814	384.4	8.586465	227	7.826548	22908	14.48356
8000	12.96578	1408.67	10.46012	2328	11.18488	962.67	9.910898		
16000	13.96578	4624.5	12.17508	9949.5	13.28041	4320	12.07682		

Graph of N v/s Raw Time



Graph of Log (N) v/s Log (Raw time)



- **Why quadratic methods work:**
  The most efficient and optimal solution to the 3Sum problem is the Quadratic solution, which can resolve the issue in O(N^2) time complexity. The Quadratic solution is based on a simple yet powerful idea: iterate through the array and for each element, consider it as one of the components of the triplet (a,b,c) we are looking for. Since the problem statement is to find three numbers in the array such that their sum is equal to zero, we can translate this to a + b + c = 0. By keeping this equation in mind, we can further deduce that if we have a and c, we can find b by b = -(a + c).
  Therefore, we can treat the current element as b and look for the desired integer -b. We then use another loop to take into account the adjacent elements of the current element, i and k. The

loop iterates in the opposite direction for i and k until a solution is found. This approach is very efficient as it takes advantage of the sorted nature of the array and avoids duplicate solutions.

Example: -
Assume the given input is [-20, -15, -10, -5, 0, 5, 10, 15, 20]
j = 3
Therefore, i = 2 & k = 4
Target value = A[3] = -5
Satisfy a + b = 5

1st iteration:
Addition = A[2] + A[4] = -10 + 0 = -10
The addition is less than the target value, increment k

2nd iteration:
Addition = A[2] + A[5] = -10 + 5 = -5
The addition is less than the target value, increment k

3rd iteration:
Addition = A[2] + A[6] = -10 + 10 = 0
The addition is less than the target value, increment k

4th iteration:
Addition = A[1] + A[7] = -10 + 15 = 5
Here, addition is equal to the target value hence we found the triplet

Now, i will decrement and k will increment in order to avoid adding the same triplet to the solution set again.
We can see that only 2 loops are utilized to fully process the array, so the time complexity is N^2.