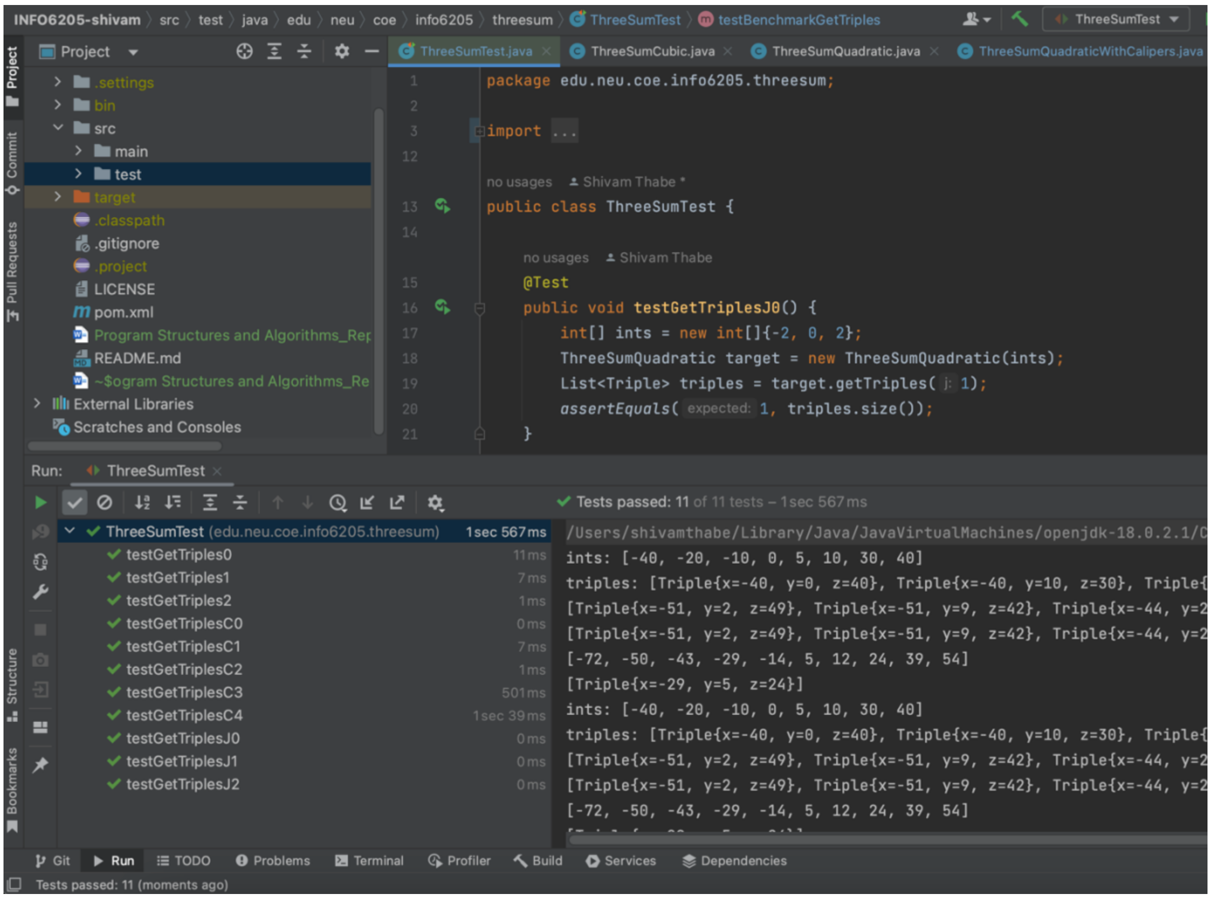
Program Structures and Algorithms
Spring 2023(SEC 03)

NAME: Shivam Thabe
NUID: 002765286
ASSIGNMENT 02: 3-SUM

Task: 3 SUM

Solve 3-SUM using the *Quadrithmic*, *Quadratic*, and (bonus point) *quadraticWithCalipers* approaches, as shown in skeleton code in the repository.

Screenshot of Unit Tests:



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

import ...

no usages  Shivam Thabe *
public class ThreeSumTest {

    no usages  Shivam Thabe
    @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 – 1sec 567ms

✓ ThreeSumTest (edu.neu.coe.info6205.threesum)   1sec 567ms
    ✓ testGetTriples0                                 11ms
    ✓ testGetTriples1                                  7ms
    ✓ testGetTriples2                                  1ms
    ✓ testGetTriplesC0                                 0ms
    ✓ testGetTriplesC1                                 7ms
    ✓ testGetTriplesC2                                 1ms
    ✓ testGetTriplesC3                               501ms
    ✓ testGetTriplesC4                          1sec 39ms
    ✓ testGetTriplesJ0                                 0ms
    ✓ testGetTriplesJ1                                 0ms
    ✓ testGetTriplesJ2                                 0ms
```

```
/Users/shivamthabe/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/C
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{
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2
[-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{
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
```

Tests passed: 11 (moments ago)

Timing Observations:

No.	N	Quadratic-Calipers (ms)	Quadratic (ms)	Quadritmic (ms)	Cubic (ms)
1	500	6	15	10	42
2	1000	4	10	14	276
3	2000	25	32	92	2070
4	4000	46	115	394	18298
5	8000	445	679	2039	144218
6	16000	2881	4391	9108	1113920

Explanation:

1) **3Sum Quadratic:**
   a. For this approach, it is an assumption that the input data will be distinct integers which are in order.
   b. To get a combination of three integers summing up to zero, a linear iteration is necessary to sweep through all the numbers.
      i. This is implemented by the outer \`for\` loop which results in a complexity of O(N).
      ii. Every element accessed in the for loop is treated as a middle number (num[j]). Since, the input data is in order, all elements to the left of the middle number (num[j]) are smaller than the element num[j] and those on the right are greater than the element.
      iii. E.g.: num[left] < num[j] < num[right]
   c. With the understanding established above, a \`while\` loop is needed which checks for the valid values of num[left] and num[right], such that the elements sum up to zero. e.g.: num[left] + num[j] + num[right] = 0
      i. If the sum of three numbers is greater than zero, decrement the left pointer.
      ii. If the sum of the three numbers is less than zero, increment the right pointer.
      iii. Since, for every element num[j], the input data needs to be iterated again to find the left and right index, this loop adds a time complexity of O(N)
   d. As there are two nested loops which iterate through the input data to find the solution, the time complexity of the approach is O (N^2)

2) **3Sum Quadratic with Callipers:**
   a. For this approach, it is an assumption that the input data will be distinct integers which are in order.
   b. To get a combination of three integers summing up to zero, a linear iteration is necessary to sweep through all the numbers.
      i. This is implemented by the outer \`for\` loop which results in a complexity of O(N).
      ii. Every element accessed in the \`for\` loop is treated as the first number of the possible solution i.e., num[i].
   c. With the understanding established above, a \`while\` loop is then run to find the other two elements such that the sum of all the three elements is zero.
      i. Two pointers are placed on the element on the immediate right side of num[i] and last element of the input data.
      ii. Now, since the input data is in order, the relationship between the three elements is num[i] < num[left] < num[right]
      iii. Then the sum of the three numbers is calculated. If the sum of three numbers is greater than zero, decrement the right pointer.
      iv. If the sum of the three numbers is less than zero, increment the left pointer.
   d. As there are two nested loops which iterate through the input data to find the solution, the time complexity of the approach is O(N^2).
   e. Although the time complexity of this approach is O(N^2), it is actually faster than the quadratic approach. This is because, as the element accessed from first \`for\` loop proceeds further in the input array, the elements to be checked for the solution reduces. i.e., The size of the array for iterating over by the \`while\` loop reduces.