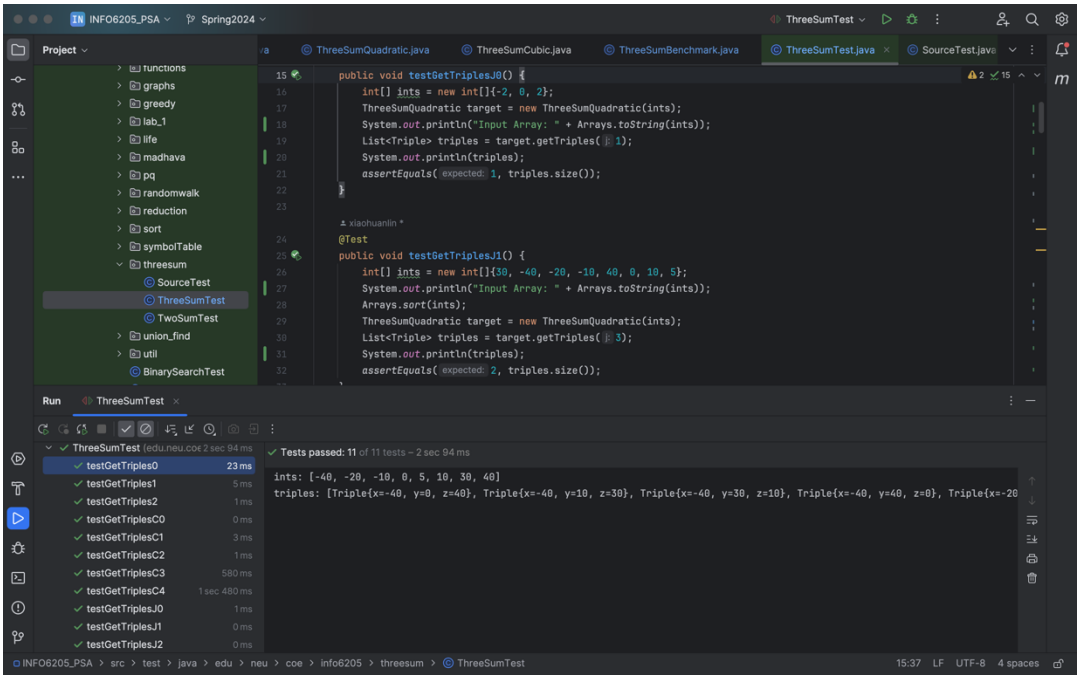
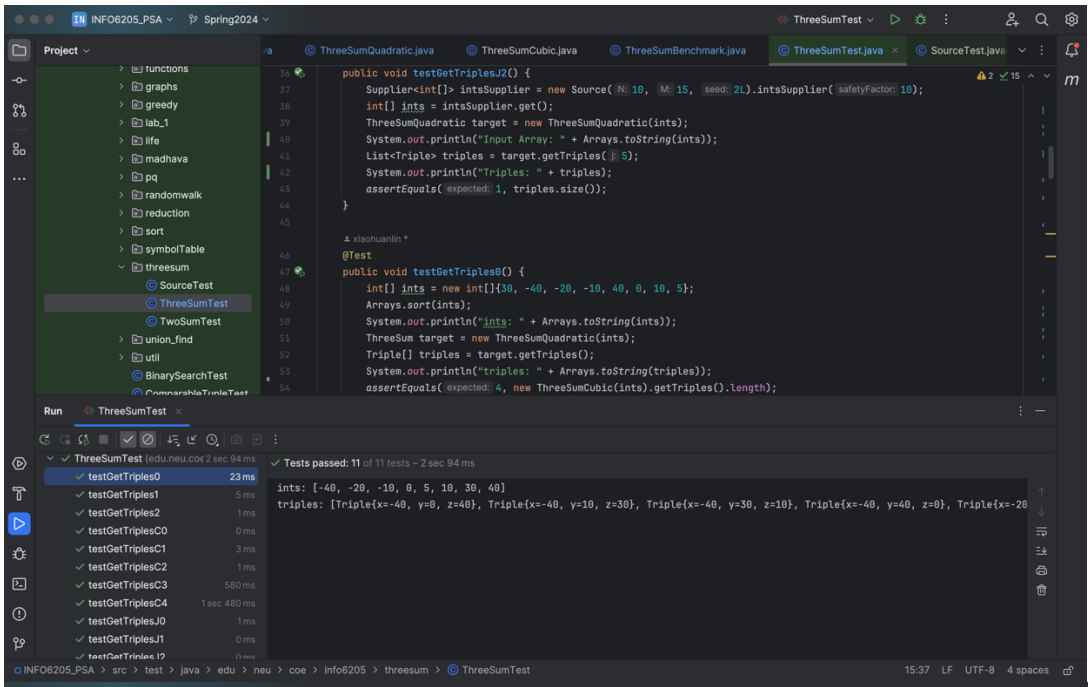
Program Structures and Algorithms Spring 2024

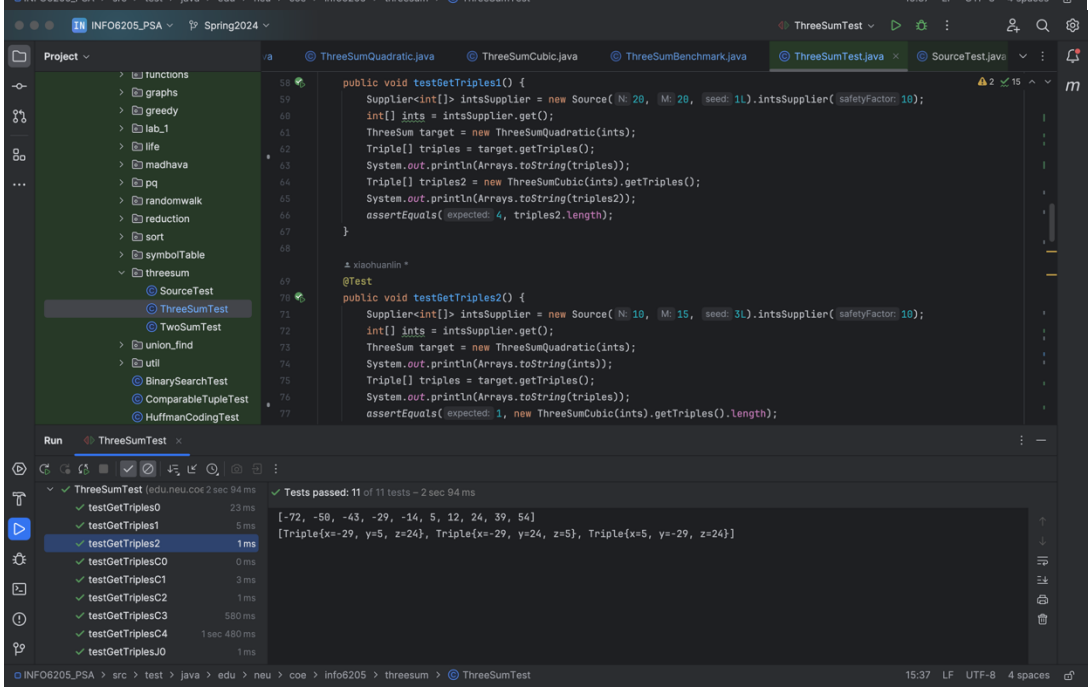
NAME: PRANAV ARUN KAPSE
NUID: 002871241
GITHUB LINK: https://github.com/kapsep/INFO6205_PSA

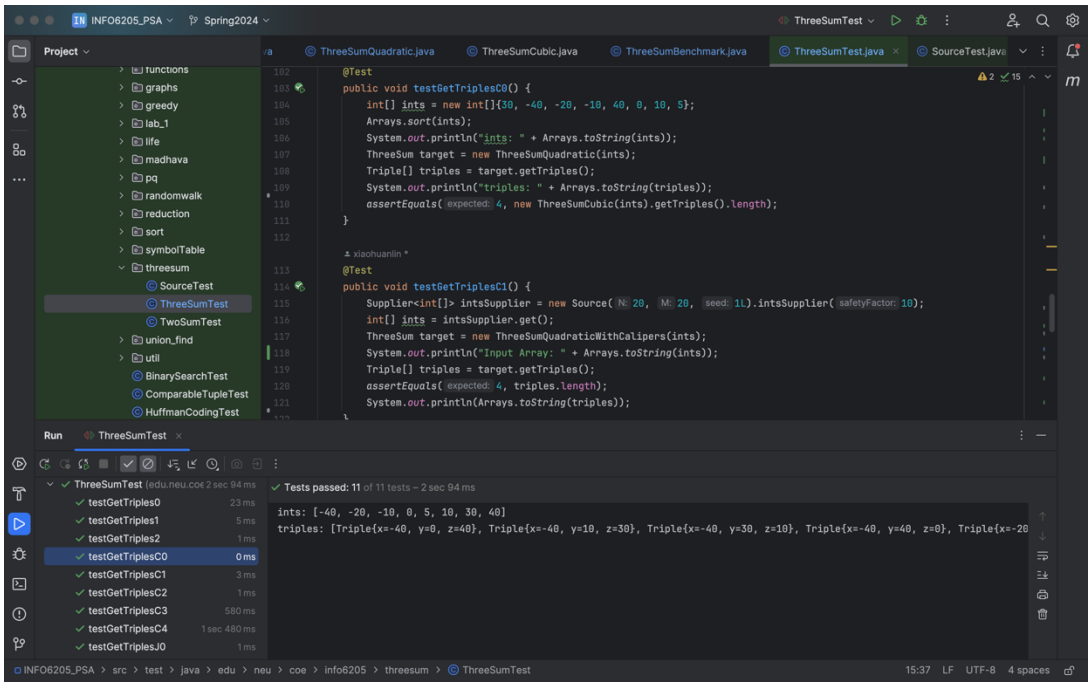
TASK: Assignment 2 (3 - SUM)

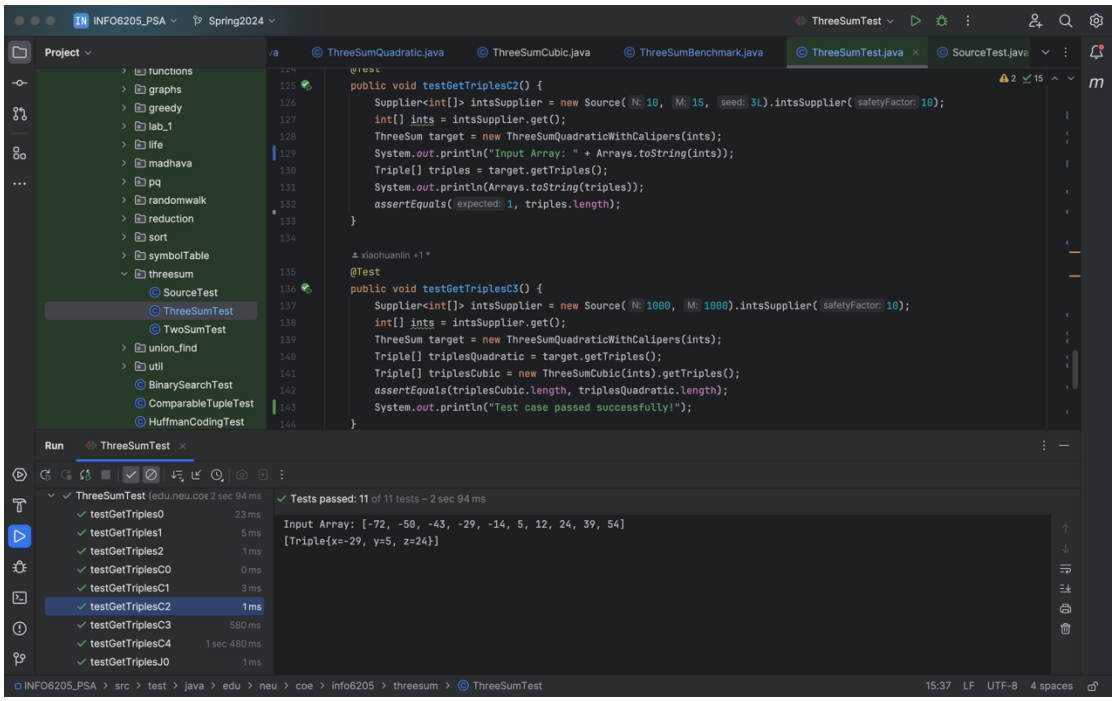
A) All 11 Unit tests are running and following are screenshots for all test cases:

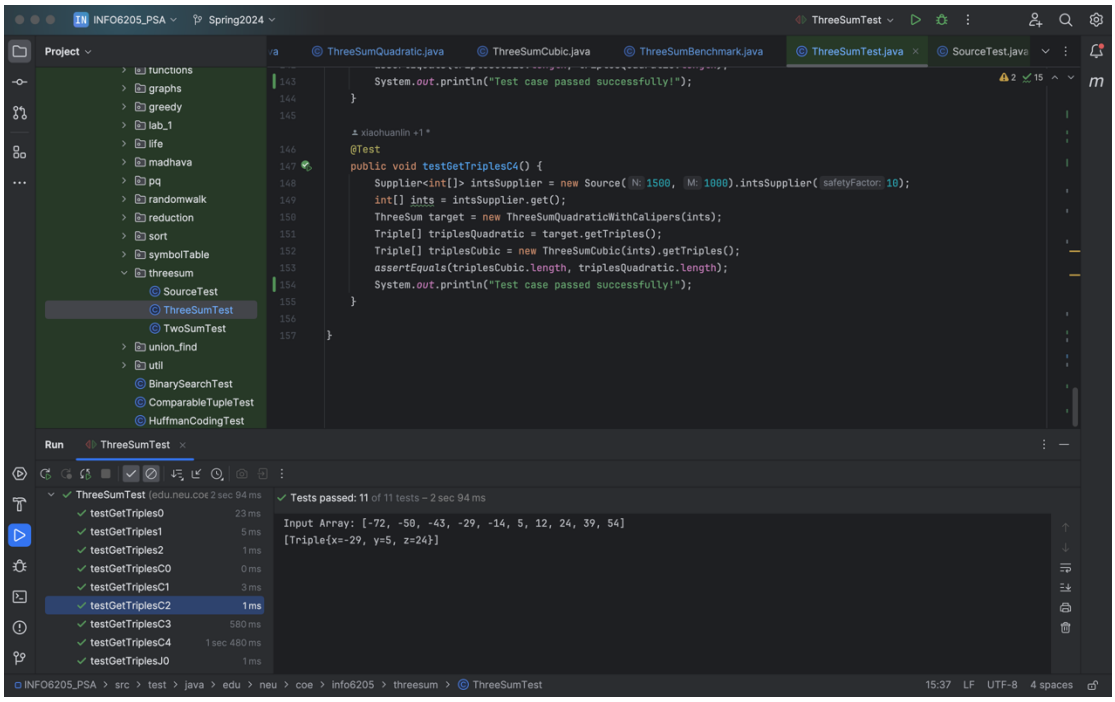



INFO6205_PSA
Spring2024
ThreeSumTest
Project
ThreeSumQuadratic.java
ThreeSumCubic.java
ThreeSumBenchmark.java
ThreeSumTest.java
SourceTest.java
functions
graphs
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTest
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);
System.out.println("Input Array: " + Arrays.toString(ints));
List<Triple> triples = target.getTriples(j: 5);
System.out.println("Triples: " + triples);
assertEquals(expected: 1, triples.size());
}
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, new ThreeSumCubic(ints).getTriples().length);
Run
ThreeSumTest
Tests passed: 11 of 11 tests – 2 sec 94 ms
ThreeSumTest (edu.neu.coe 2 sec 94 ms
testGetTriples0 23 ms
testGetTriples1 5 ms
testGetTriples2 1 ms
testGetTriplesC0 0 ms
testGetTriplesC1 3 ms
testGetTriplesC2 1 ms
testGetTriplesC3 580 ms
testGetTriplesC4 1 sec 480 ms
testGetTriplesJ0 1 ms
testGetTriplesJ1 0 ms
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=-40, y=30, z=10}, Triple{x=-40, y=40, z=0}, Triple{x=-20
INFO6205_PSA > src > test > java > edu > neu > coe > info6205 > threesum > ThreeSumTest
15:37 LF UTF-8 4 spaces


INFO6205_PSA
Spring2024
ThreeSumTest
Project
ThreeSumQuadratic.java
ThreeSumCubic.java
ThreeSumBenchmark.java
ThreeSumTest.java
SourceTest.java
functions
graphs
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTest
HuffmanCodingTest
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();
System.out.println(Arrays.toString(triples));
Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
System.out.println(Arrays.toString(triples2));
assertEquals(expected: 4, triples2.length);
}
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, new ThreeSumCubic(ints).getTriples().length);
Run
ThreeSumTest
Tests passed: 11 of 11 tests – 2 sec 94 ms
ThreeSumTest (edu.neu.coe 2 sec 94 ms
testGetTriples0 23 ms
testGetTriples1 5 ms
testGetTriples2 1 ms
testGetTriplesC0 0 ms
testGetTriplesC1 3 ms
testGetTriplesC2 1 ms
testGetTriplesC3 580 ms
testGetTriplesC4 1 sec 480 ms
testGetTriplesJ0 1 ms
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}, Triple{x=-29, y=24, z=5}, Triple{x=5, y=-29, z=24}]
INFO6205_PSA > src > test > java > edu > neu > coe > info6205 > threesum > ThreeSumTest
15:37 LF UTF-8 4 spaces


INFO6205_PSA
Spring2024
ThreeSumTest
Project
ThreeSumQuadratic.java
ThreeSumCubic.java
ThreeSumBenchmark.java
ThreeSumTest.java
SourceTest.java
graphs
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTest
HuffmanCodingTest
@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, new ThreeSumCubic(ints).getTriples().length);
}
xiaohuanlin *
@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);
System.out.println("Input Array: " + Arrays.toString(ints));
Triple[] triples = target.getTriples();
assertEquals(expected: 4, triples.length);
System.out.println(Arrays.toString(triples));
Run
ThreeSumTest
Tests passed: 11 of 11 tests – 2 sec 94 ms
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=-40, y=30, z=10}, Triple{x=-40, y=40, z=0}, Triple{x=-20
testGetTriples0 23 ms
testGetTriples1 5 ms
testGetTriples2 1 ms
testGetTriplesC0 0 ms
testGetTriplesC1 3 ms
testGetTriplesC2 1 ms
testGetTriplesC3 580 ms
testGetTriplesC4 1 sec 480 ms
testGetTriplesJ0 1 ms
INFO6205_PSA › src › test › java › edu › neu › coe › info6205 › threesum › ThreeSumTest
15:37 LF UTF-8 4 spaces


INFO6205_PSA
Spring2024
ThreeSumTest
Project
ThreeSumQuadratic.java
ThreeSumCubic.java
ThreeSumBenchmark.java
ThreeSumTest.java
SourceTest.java
graphs
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTest
HuffmanCodingTest
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("Input Array: " + Arrays.toString(ints));
Triple[] triples = target.getTriples();
System.out.println(Arrays.toString(triples));
assertEquals(expected: 1, triples.length);
}
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);
System.out.println("Test case passed successfully!");
}
Run
ThreeSumTest
Tests passed: 11 of 11 tests – 2 sec 94 ms
Input Array: [-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
testGetTriples0 23 ms
testGetTriples1 5 ms
testGetTriples2 1 ms
testGetTriplesC0 0 ms
testGetTriplesC1 3 ms
testGetTriplesC2 1 ms
testGetTriplesC3 580 ms
testGetTriplesC4 1 sec 480 ms
testGetTriplesJ0 1 ms
INFO6205_PSA › src › test › java › edu › neu › coe › info6205 › threesum › ThreeSumTest
15:37 LF UTF-8 4 spaces


INFO6205_PSA
Spring2024
ThreeSumTest
Project
functions
graphs
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTest
HuffmanCodingTest
ThreeSumQuadratic.java
ThreeSumCubic.java
ThreeSumBenchmark.java
ThreeSumTest.java
SourceTest.java
System.out.println("Test case passed successfully!");
}
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);
System.out.println("Test case passed successfully!");
}
}
Run
ThreeSumTest
ThreeSumTest (edu.neu.coe 2 sec 94 ms
testGetTriples0 23 ms
testGetTriples1 5 ms
testGetTriples2 1 ms
testGetTriplesC0 0 ms
testGetTriplesC1 3 ms
testGetTriplesC2 1 ms
testGetTriplesC3 580 ms
testGetTriplesC4 1 sec 480 ms
testGetTriplesJ0 1 ms
Tests passed: 11 of 11 tests – 2 sec 94 ms
Input Array: [-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
INFO6205_PSA > src > test > java > edu > neu > coe > info6205 > threesum > ThreeSumTest
15:37 LF UTF-8 4 spaces

B) 7 examples of Timing observations as follows:

Sr.No	Algorithm	Input Size(N)	Raw Time per Run (mSec)	Normalized Time per Run
1	ThreeSumQuadratic	250	2.53	40.48
	ThreeSumQuadrathmic	250	1.48	2.97
	ThreeSumQuadraticWithCalipers	250	0.64	10.24
	ThreeSumCubic	250	5.73	0.37
2	ThreeSumQuadratic	500	3.12	12.48
	ThreeSumQuadrathmic	500	3.08	1.37
	ThreeSumQuadraticWithCalipers	500	0.9	3.6
	ThreeSumCubic	500	36.92	0.3
3	ThreeSumQuadratic	1000	13.25	13.25
	ThreeSumQuadrathmic	1000	18.05	1.81
	ThreeSumQuadraticWithCalipers	1000	3.6	3.6
	ThreeSumCubic	1000	286.4	0.29
4	ThreeSumQuadratic	2000	72.2	18.05
	ThreeSumQuadrathmic	2000	92.1	2.1
	ThreeSumQuadraticWithCalipers	2000	27.6	6.9
	ThreeSumCubic	2000	2261	0.28
5	ThreeSumQuadratic	4000	396.2	24.76
	ThreeSumQuadrathmic	4000	442.4	2.31
	ThreeSumQuadraticWithCalipers	4000	145.8	9.11
	ThreeSumCubic	4000	18018.6	0.28
6	ThreeSumQuadratic	8000	1850	28.91
	ThreeSumQuadrathmic	8000	2096	2.53
	ThreeSumQuadraticWithCalipers	8000	577	9.02
	ThreeSumCubic	8000	143292.33	0.28
7	ThreeSumQuadratic	16000	10356	40.45
	ThreeSumQuadrathmic	16000	9139	2.56
	ThreeSumQuadraticWithCalipers	16000	2696	10.53
	ThreeSumCubic	16000	1706772.5	0.42

C) The 3SUM issue is aimed to be solved more quickly using the quadratic methods - Quadratic, QuadraticWithCalipers, and Quadrithmic than it is with the cubic method.

Determining every triple in an array of integers that adds up to zero is the primary objective of the 3SUM problem. The quadratic methods take advantage of the idea that we can optimize the search process if we fix one element (let's say the middle element) and look for two other elements whose total with the fixed element equals zero.

Here's a brief explanation of why the quadratic methods work:

1. Quadratic Method:
    - It is expected that the array is sorted. The method uses two pointers (left and right) to scan the array from both ends for each entry at position i.
    - The pointers proceed in the direction of one another when the sum of the elements at indices i, left, and right is equivalent to zero.
    - The left pointer incremented for considering a larger element if the sum is less than zero.
    - The right pointer is decremented to take into consideration a lesser element if the sum is larger than zero.
    - A triple is found if the sum equals zero, and both pointers are then changed to explore further possible triples.
2. QuadraticWithCaliper Method:
    - It is like the quadratic approach but qualifies triples and explores the array using a caliper function.
    - When given a triple, the caliper function returns a value that matches with zero. This function helps to verify the validity of the existing triple solution.
    - The calipers function is used to filter and navigate through potential triples efficiently.
3. Quadrathmic Method:
    - By using binary search, this method improves upon the quadratic approach.

- It is assumed that the array is sorted. The algorithm executes a binary search for each pair of elements at indices i and j to locate the third element such that the sum is zero.
- By using a binary search, the quadratic method's O(N) time complexity is reduced to O(log N).
- This approach finds the 3rd element for a given pair quickly by utilizing the array's sorted structure.
-

In conclusion, by using effective search techniques, these quadratic methods take advantage of the array's sorted nature to lower the 3SUM problem's time complexity from cubic (O(N^3)) to quadratic (O(N^2)) or even Quadrithmic (O(N^2 log N)).