Program Structures and Algorithms
Spring 2023(SEC –8)

NAME: Mehul Natu
NUID: 002743870

Task:

- brief explanation of why the quadratic method(s) work.
- spreadsheet showing your timing observations for cubic, Quadarithmic, Quadratic and Quadratic with calipers.
- Test cases passing along with code

Relationship Conclusion:

- Cubic slope averages out to be : 2.977 this can be seen from figure 2 in graphical representation section in column K, highlighted cell yellow.
- Quadrathmic slope averages out to be : 2.15 this can be seen from figure 2 in graphical representation section in column L, highlighted cell yellow.
- Quadratic with calipers slope averages out to be : 2.022 this can be seen from figure 2 in graphical representation section in column I, highlighted cell yellow.
- Quadratic slope averages out to be : 1.927 this can be seen from figure 2 in graphical representation section in column H, highlighted cell yellow.

Evidence to support that conclusion:
For better understanding I have got three pointers where i <= j <= k
Also ints is the array here.
"currSum" = ints[i] + ints[j] + int[j]

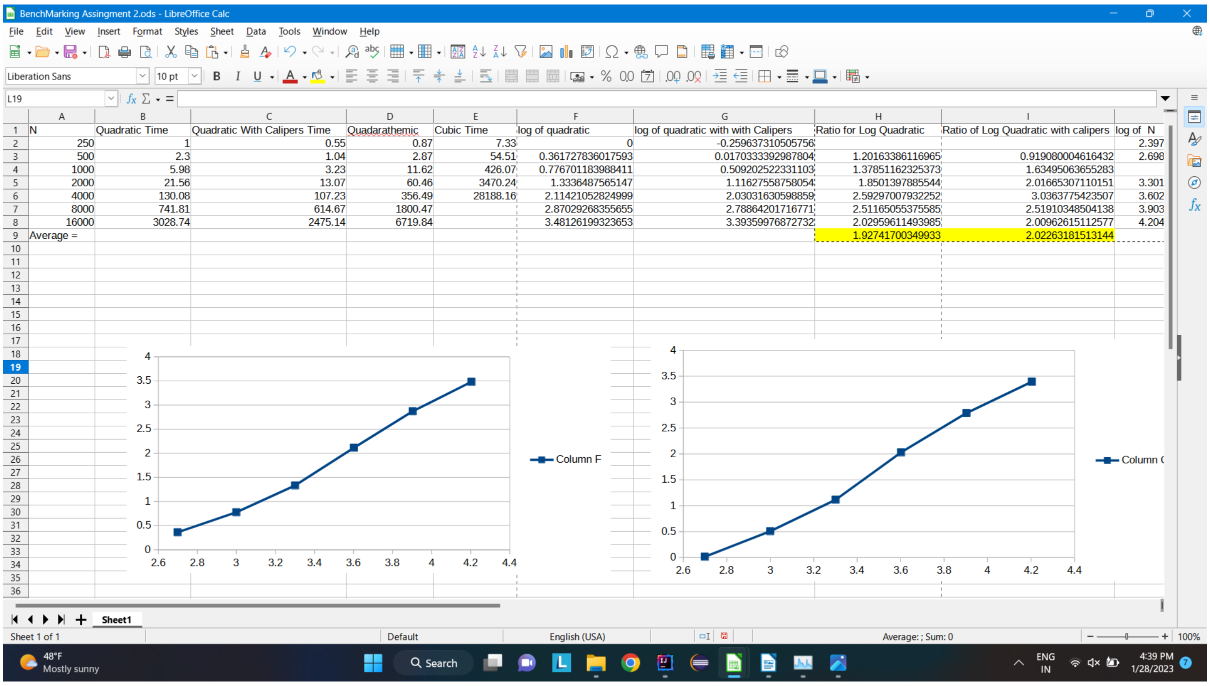
Quadratic: So let's start with the normal quadratic. This is based on the assumption that the array is sorted. What I am doing if for each value of "j" index(of the array) I am moving across the array in two directions one with "i" in the left side and one with "k" in the right side. Initial value of i = j – 1 and of k = i + 1.
"i" is always decremented when "currSum" = ints[i] + ints[j] + ints[k] > 0 so as to lower the value of currSum. And k is incremented whenever "currSum" gets below 0 so as to increase its value. And this will stop either when "i" becomes less than 0 or k becomes ints.length. So the worst case is if are iterating the whole array for each turn which will corresponds to O(n) for each value of "j" and since j will have in total n values the time complexity corresponds to $O(n^2)$. And also we can see from the graph between column F and Column J from graphical representation is it is kind of between $O(n^{1.2}) – O(n^{2.5})$. Also we can see the ration of Log(time)/Log(n) in the column **H.** Also the slope or values of column H averages out to be 1.920.

Quadratic with calipers: So this approach is similar to previous one the only difference is that now we are moving the j and k pointers that to in the range where indexes are greater than "i". So here "i" is fixed for the time when we are moving "j" and "k". Initial value of j = i + 1 and of k = ints.length – 1. So for each I what we are gonna do is decrement k whenever "currSum" > 0 and then we start increasing the "j" whenever "currSum" < 0. This kind of acts as two holding end of a caliper. Always decreasing k and increasing j thus tightly holding the sub-array like a caliper. And since the worst case for a fixed "i" would be moving "j" and "k" to cover the whole array so taking in consideration of each "I". This also becomes quadratic $O(n^2)$. Because "i" would have at most n values and if every case is worst case for each "i" we will traverse the whole array thus O(n*n). Also we can see the ratio of Log(time)/Log(n) in the column I**.** Also the slope or values of column I averages out to be 2.022.

From the graphical representation and from column H and I for quadratic and quadratic with calipers we can check that the ratio or slope = $y_2 - y_1/x_2 - x_1$ => comes out to be around 2. Here y = Log(Time) and x = Log(N). N is number of elements in an array. So This gives us slope which is basically the power of N in the equation $T = N^m$. This comes out by taking log on both the sides Log(T) = Mlog(N). m which is the slope of the equation y = mx. Where y = log(T) and x =log(N). m should be $(\text{Log}(t_2) - \text{Log}(t_1))/(\text{Log}(x_2) - \log(x_1))$.

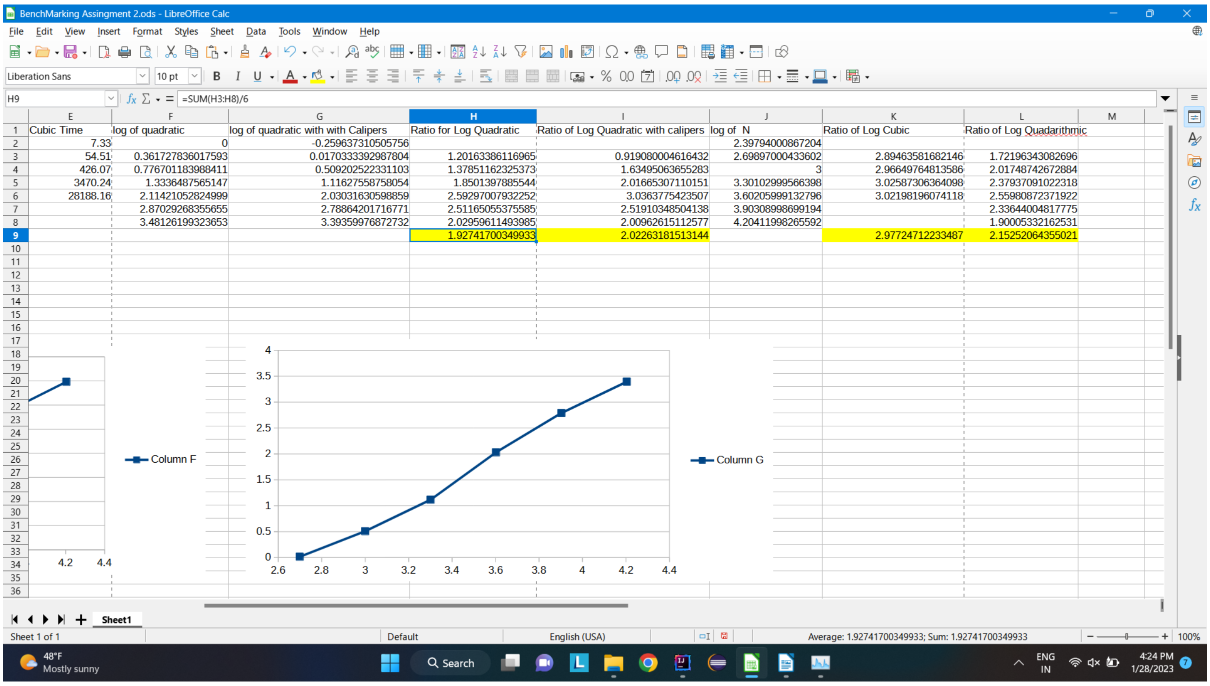
Graphical Representation:

Figure 1:

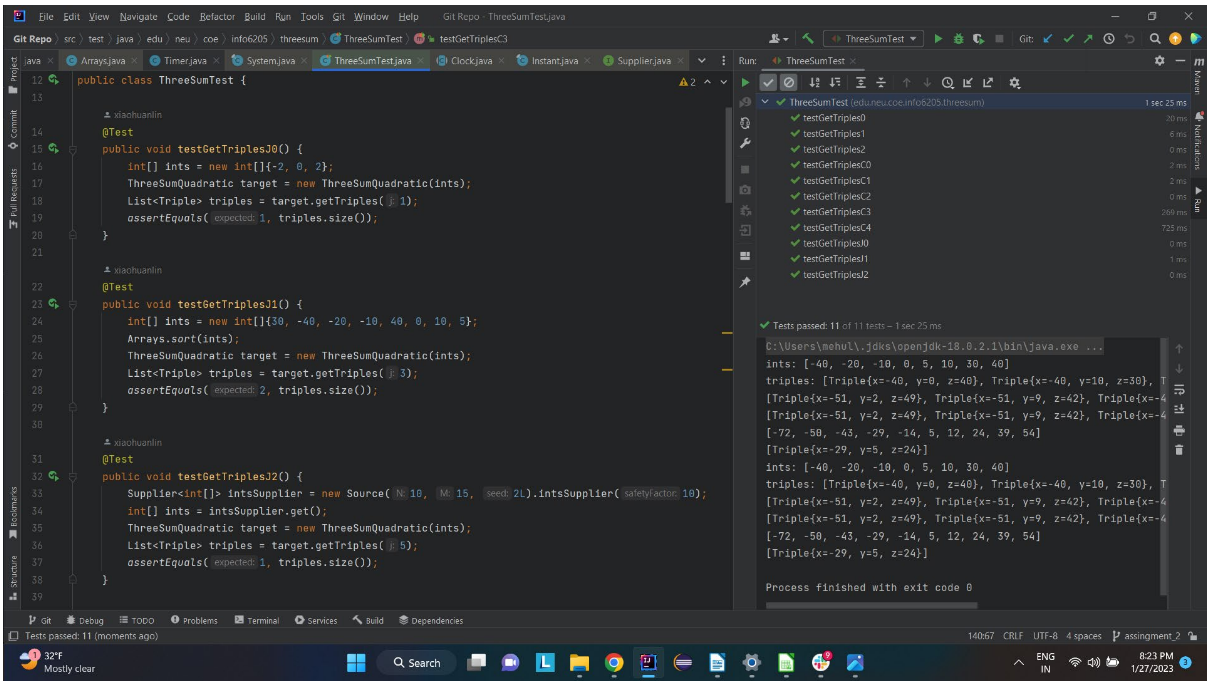


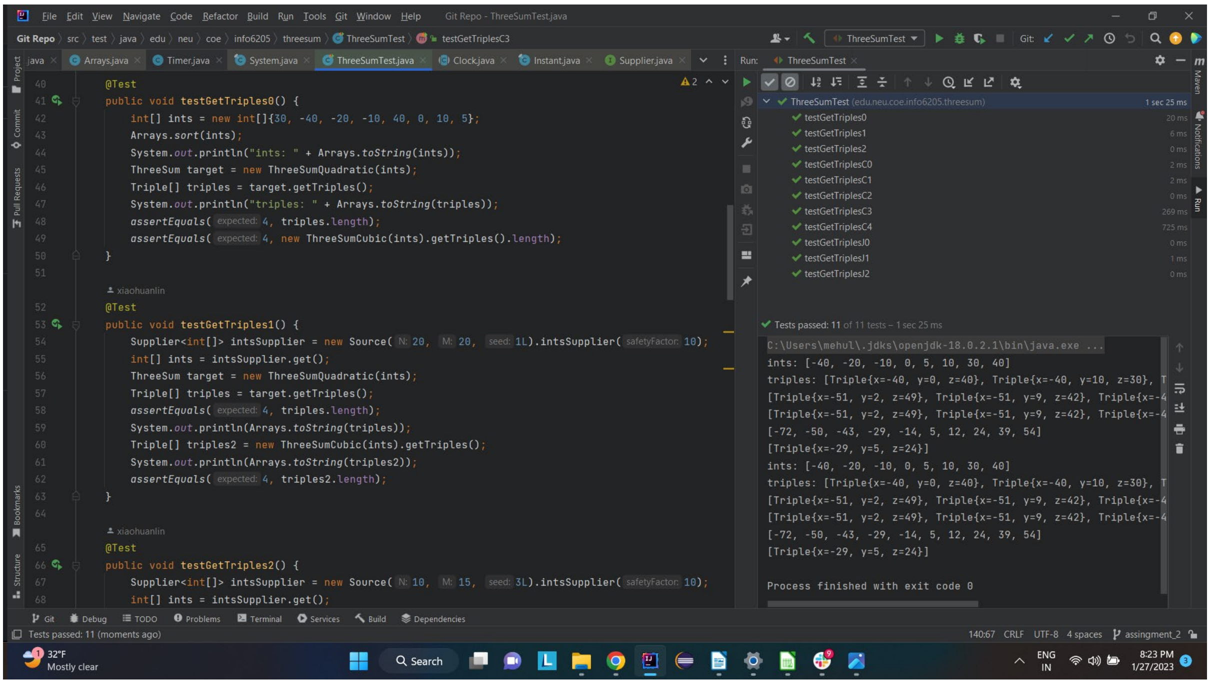
N	Quadratic Time	Quadratic With Calipers Time	Quadarathemic	Cubic Time	log of quadratic	log of quadratic with with Calipers	Ratio for Log Quadratic	Ratio of Log Quadratic with calipers	log of N
250	1	0.55	0.87	7.33	0	-0.259637310505756			2.397
500	2.3	1.04	2.87	54.51	0.361727836017593	0.0170333392987804	1.20163386116965	0.919080004616432	2.698
1000	5.98	3.23	11.62	426.07	0.776701183988411	0.509202522331103	1.37851162325373	1.63495063655283	
2000	21.56	13.07	60.46	3470.24	1.3336487565147	1.11627558758054	1.8501397885544	2.01665307110151	3.301
4000	130.08	107.23	356.49	28188.16	2.11421052824999	2.03031630598859	2.59297007932252	3.0363775423507	3.602
8000	741.81	614.67	1800.47		2.87029268355655	2.78864201716771	2.51165055375585	2.51910348504138	3.903
16000	3028.74	2475.14	6719.84		3.48126199323653	3.39359976872732	2.02959611493985	2.00962615112577	4.204
Average =							1.92741700349933	2.02263181513144	

Figure 2:

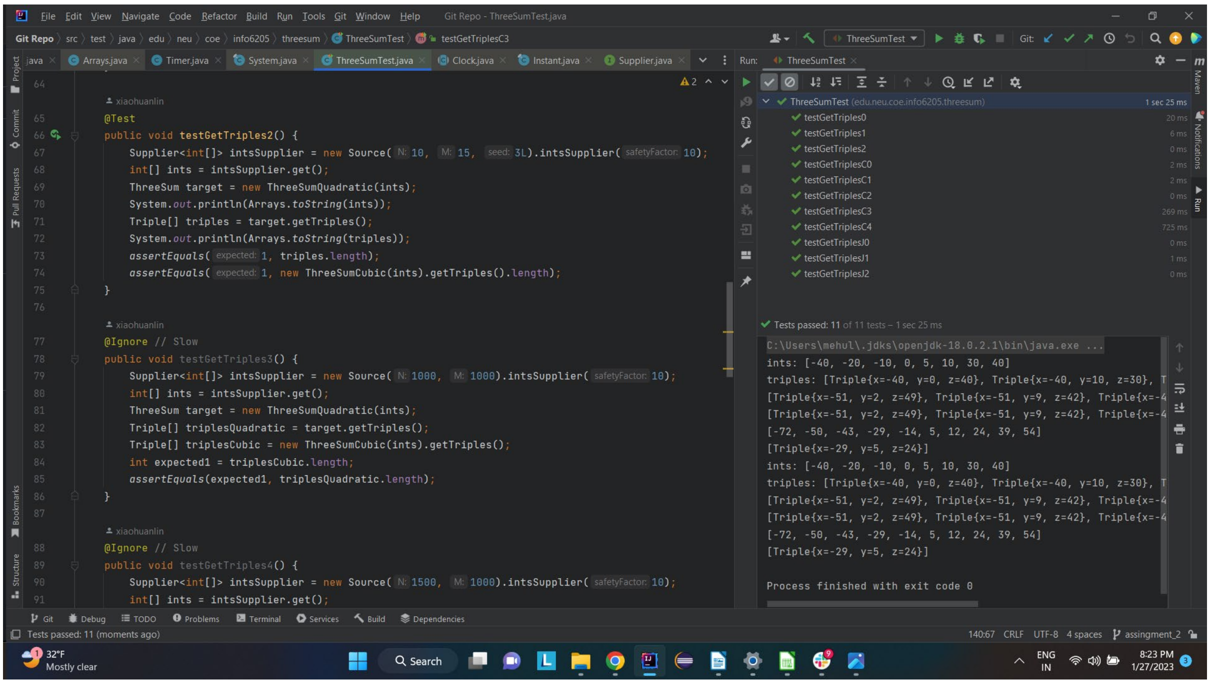


Unit Test Screenshots: 1

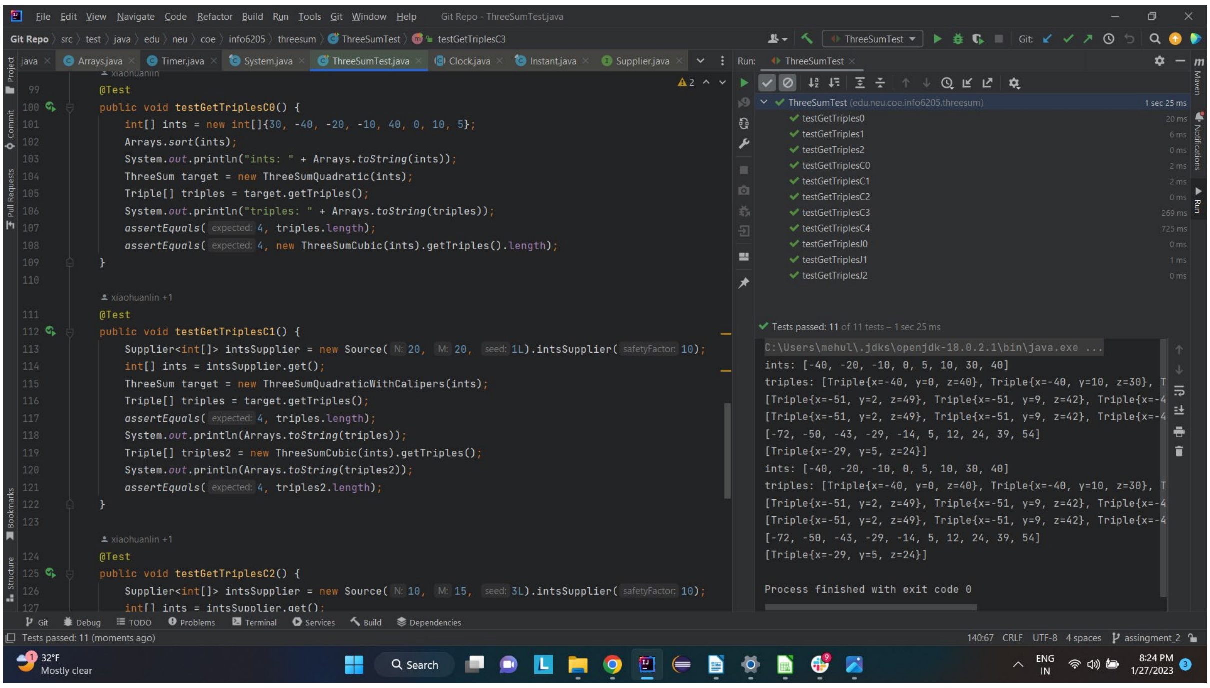



@Test
public void testGetTriples0() {
int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
Arrays.sort(ints);
System.out.println("ints: " + Arrays.toString(ints));
ThreeSum target = new ThreeSumQuadratic(ints);
Triple[] triples = target.getTriples();
System.out.println("triples: " + Arrays.toString(triples));
assertEquals(expected: 4, triples.length);
assertEquals(expected: 4, new ThreeSumCubic(ints).getTriples().length);
}
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);
}
xiaohuanlin
@Test
public void testGetTriples2() {
Supplier<int[]> intsSupplier = new Source(N: 10, M: 15, seed: 3L).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSumTest (edu.neu.coe.info6205.threesum) 1 sec 25 ms
testGetTriples0 20 ms
testGetTriples1 6 ms
testGetTriples2 0 ms
testGetTriplesC0 2 ms
testGetTriplesC1 2 ms
testGetTriplesC2 0 ms
testGetTriplesC3 269 ms
testGetTriplesC4 725 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 1 ms
testGetTriplesJ2 0 ms
Tests passed: 11 of 11 tests – 1 sec 25 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}, T
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-4
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-4
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
Process finished with exit code 0
Tests passed: 11 (moments ago)

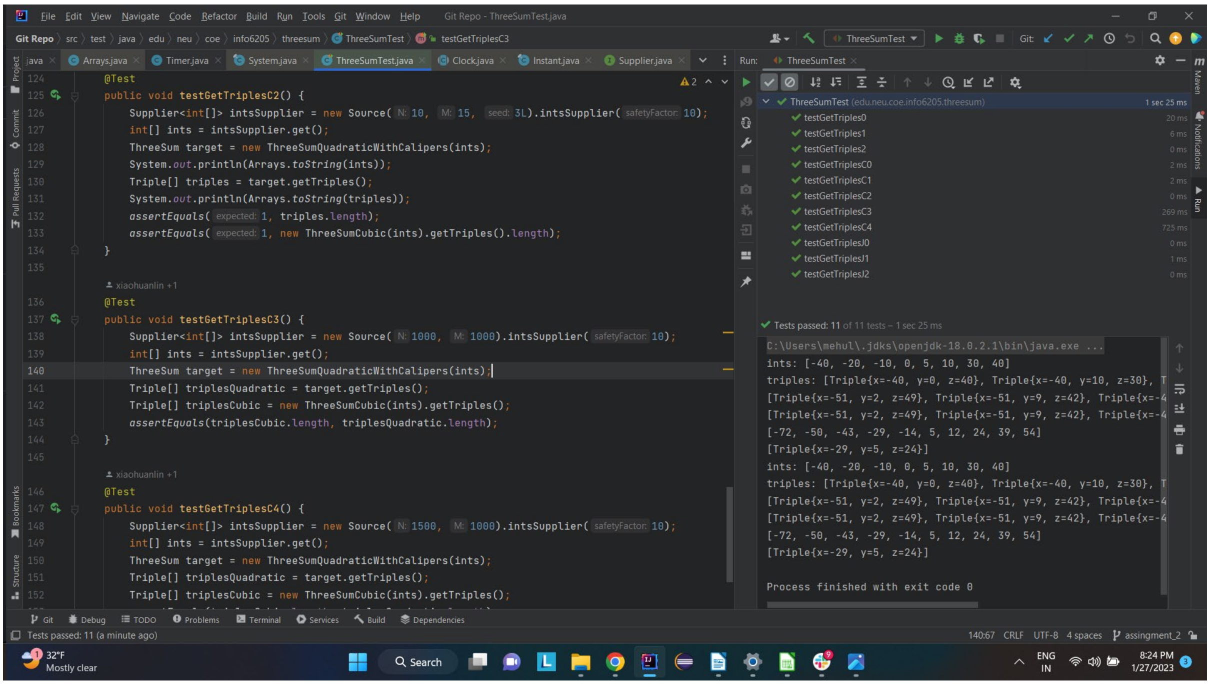
3


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);
}
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);
}
xiaohuanlin
@Ignore // Slow
public void testGetTriples4() {
Supplier<int[]> intsSupplier = new Source(N: 1500, M: 1000).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
Tests passed: 11 of 11 tests – 1 sec 25 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}, T
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
Process finished with exit code 0
Tests passed: 11 (moments ago)

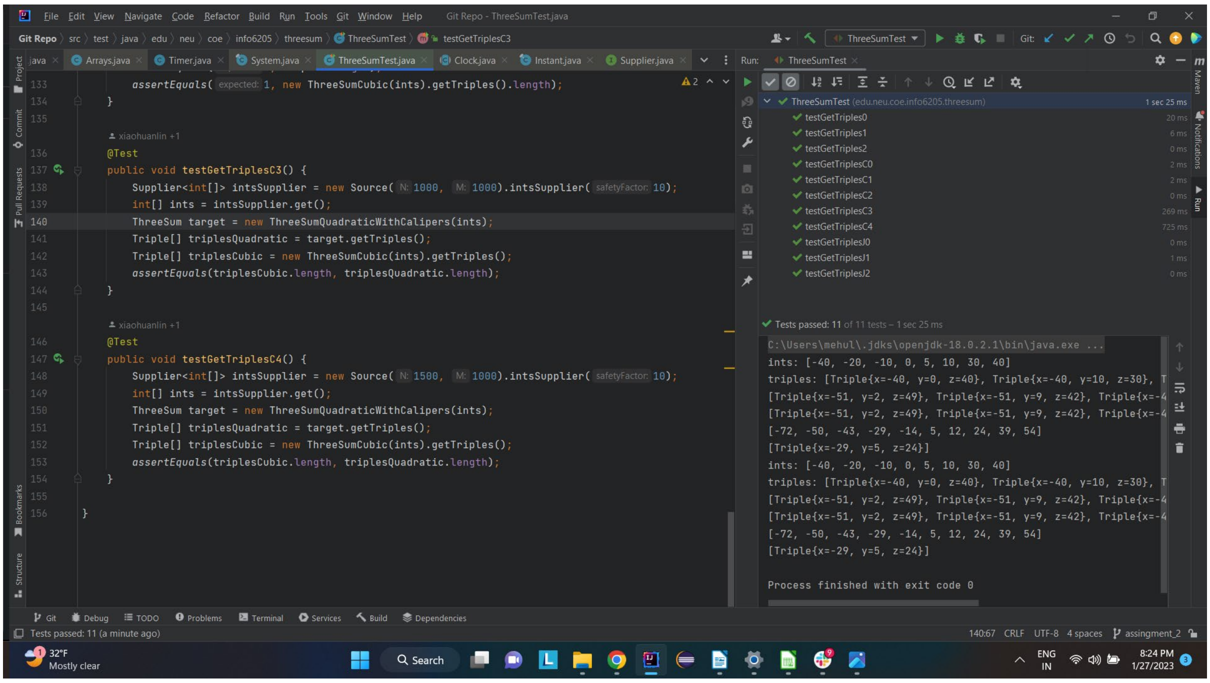
4


File Edit View Navigate Code Refactor Build Run Tools Git Window Help Git Repo - ThreeSumTest.java
Git Repo src test java edu neu coe info6205 threesum ThreeSumTest testGetTriplesC3
@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);
}
@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);
}
@Test
public void testGetTriplesC2() {
Supplier<int[]> intsSupplier = new Source(N: 10, M: 15, seed: 3L).intsSupplier(safetyFactor: 10);
ThreeSumTest (edu.neu.coe.info6205.threesum) 1 sec 25 ms
testGetTriples0 20 ms
testGetTriples1 6 ms
testGetTriples2 0 ms
testGetTriplesC0 2 ms
testGetTriplesC1 2 ms
testGetTriplesC2 0 ms
testGetTriplesC3 269 ms
testGetTriplesC4 725 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 1 ms
testGetTriplesJ2 0 ms
Tests passed: 11 of 11 tests – 1 sec 25 ms
C:\Users\mehul\.jdks\openjdk-18.0.2.1\bin\java.exe ...
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
Process finished with exit code 0
Tests passed: 11 (moments ago)
8:24 PM 1/27/2023

5


File Edit View Navigate Code Refactor Build Run Tools Git Window Help Git Repo - ThreeSumTest.java
@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);
}
@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);
}
@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();
Tests passed: 11 of 11 tests – 1 sec 25 ms
Process finished with exit code 0
Tests passed: 11 (a minute ago)
8:24 PM 1/27/2023

6



Git Repo - ThreeSumTest.java
Git Repo › src › test › java › edu › neu › coe › info6205 › threesum › ThreeSumTest › testGetTriplesC3
assertEquals(expected: 1, new ThreeSumCubic(ints).getTriples().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);
}
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);
}
}
Run: ThreeSumTest
ThreeSumTest (edu.neu.coe.info6205.threesum) 1 sec 25 ms
testGetTriples0 20 ms
testGetTriples1 6 ms
testGetTriples2 0 ms
testGetTriplesC0 2 ms
testGetTriplesC1 2 ms
testGetTriplesC2 0 ms
testGetTriplesC3 269 ms
testGetTriplesC4 725 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 1 ms
testGetTriplesJ2 0 ms
Tests passed: 11 of 11 tests – 1 sec 25 ms
C:\Users\mehul\.jdks\openjdk-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=-51, y=2, z=49}, Triple{x=-51, y=9, z=42},
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42},
[-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=-51, y=2, z=49}, Triple{x=-51, y=9, z=42},
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42},
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
Process finished with exit code 0
Tests passed: 11 (a minute ago)
140:67 CRLF UTF-8 4 spaces assingment_2
32°F Mostly clear
8:24 PM 1/27/2023