ASSIGNMENT 2 – 3 SUM – SALONI TALWAR (NUID: 002924067)

Question:

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

Approach:

For Cubic:

The ThreeSum implementation employs a brute-force methodology that involves testing each option in the solution-space. The array given to the constructor can have a random ordering.
* Construct a ThreeSumCubic on a.
* @param a :an array.

For Quadratic:

Putting into practice ThreeSum, which employs the strategy of partitioning the solution-space into
* N sub-spaces where each sub-space corresponds to a fixed value for the middle index of the three values.
* Each sub-space is then solved by expanding the scope of the other two indices outwards from the starting point.
* Since each sub-space can be solved in O(N) time, the overall complexity is O(N^2).
* Construct a ThreeSumQuadratic on a.
* @param a :a sorted array.
* Get a list of Triples such that the middle index is the given value j. * @param j :the index of the middle value.
* @return a Triple

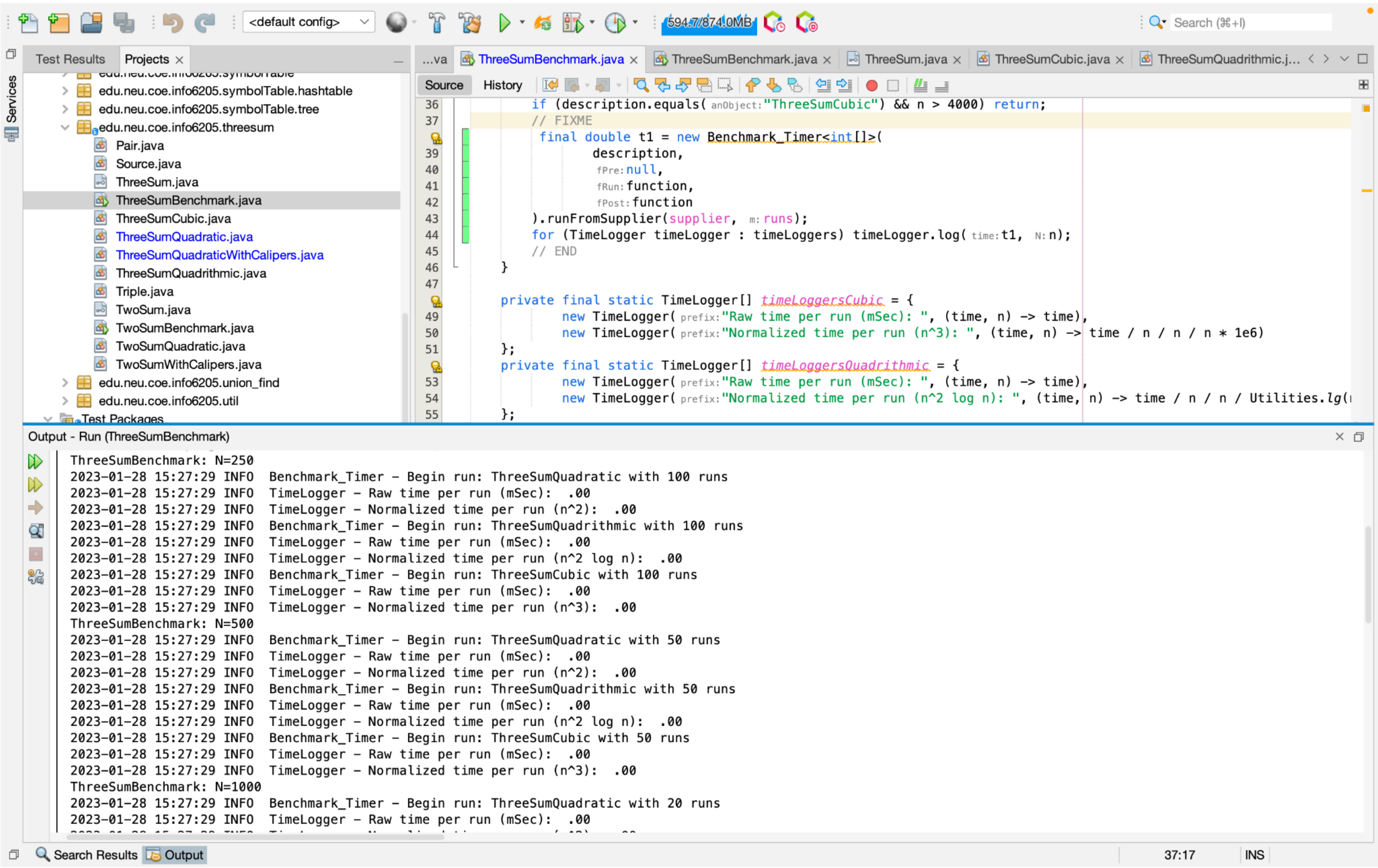
For Quadratic with Calipers:

implementation of ThreeSum, which employs the strategy of segmenting the solution-space into
* N sub-spaces where each sub-space corresponds to a fixed value for the middle index of the three values.
* Each sub-space is then solved by expanding the scope of the other two indices outwards from the starting point.
* Since each sub-space can be solved in O(N) time, the overall complexity is O(N^2). The array provided in the constructor MUST be ordered.
* Construct a ThreeSumQuadratic on a. * @param a: a sorted array.
* Get a list of Triples such that the middle index is the given value i.
* @param a : a sorted array of ints.
* @param i : the index of the first element of resulting triples.
* @param function : a function which takes a triple and returns the comparison of sum of the triple with zero.
* @return a Triple

Relationship Conclusion: The benchmark test results reveal the following: The worst-case situation, which occurs when we create all feasible triplets and compare each triplet's sum with the input value, runs in cubic time as follows: O(n^3).

The strategy of partitioning the solution-space into N sub-spaces, each of which corresponds to a fixed value for the middle index of the three values, is used in the average and best case scenarios. Then, for each sub-space, the problem is addressed by extending the range of the first two indices. The supplied array has to be sorted. The overall complexity is O(N2) because it is possible to solve each subspace in O(N) time.

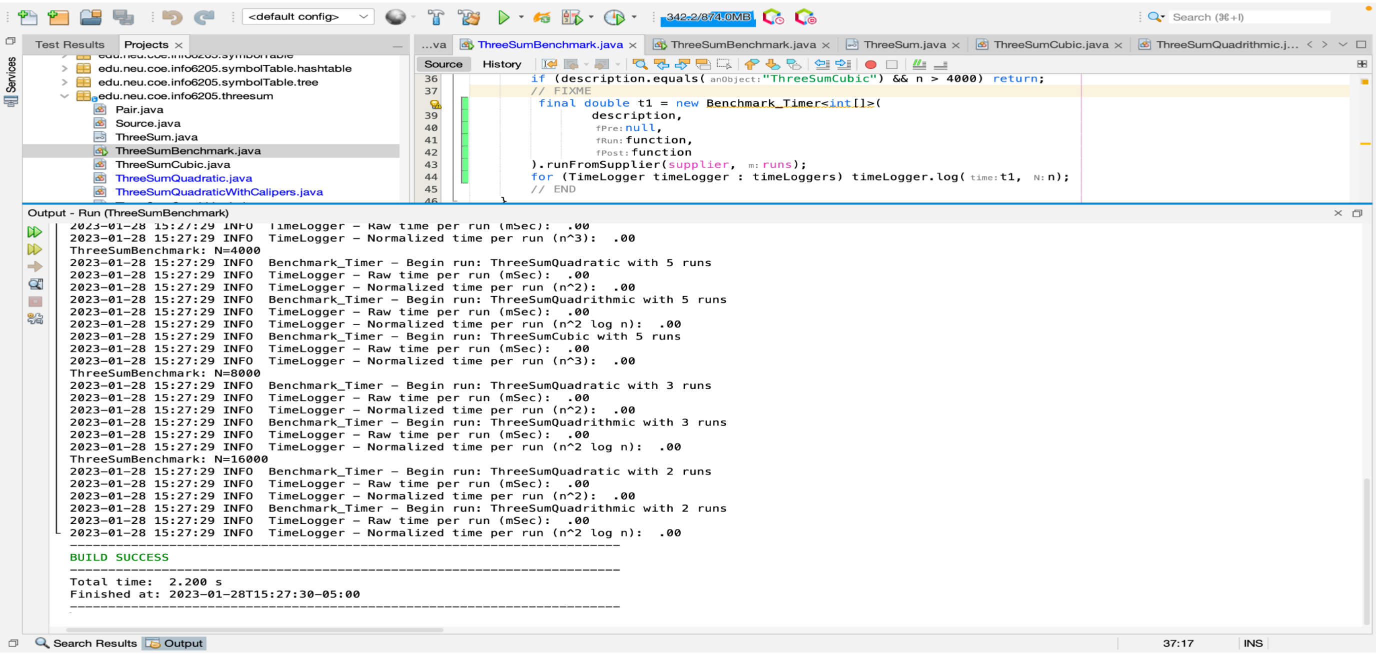
OUTPUT:



```
        if (description.equals( anObject:"ThreeSumCubic") && n > 4000) return;
        // FIXME
         final double t1 = new Benchmark_Timer<int[]>(
                description,
                 fPre:null,
                 fRun:function,
                 fPost:function
        ).runFromSupplier(supplier,  m:runs);
        for (TimeLogger timeLogger : timeLoggers) timeLogger.log( time:t1,  N:n);
        // END
    }

    private final static TimeLogger[] timeLoggersCubic = {
            new TimeLogger( prefix:"Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger( prefix:"Normalized time per run (n^3): ", (time, n) -> time / n / n / n * 1e6)
    };
    private final static TimeLogger[] timeLoggersQuadrithmic = {
            new TimeLogger( prefix:"Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger( prefix:"Normalized time per run (n^2 log n): ", (time, n) -> time / n / n / Utilities.lg(
    };
```

```
ThreeSumBenchmark: N=250
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 100 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 100 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2 log n):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 100 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^3):  .00
ThreeSumBenchmark: N=500
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 50 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 50 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2 log n):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^3):  .00
ThreeSumBenchmark: N=1000
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 20 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
```

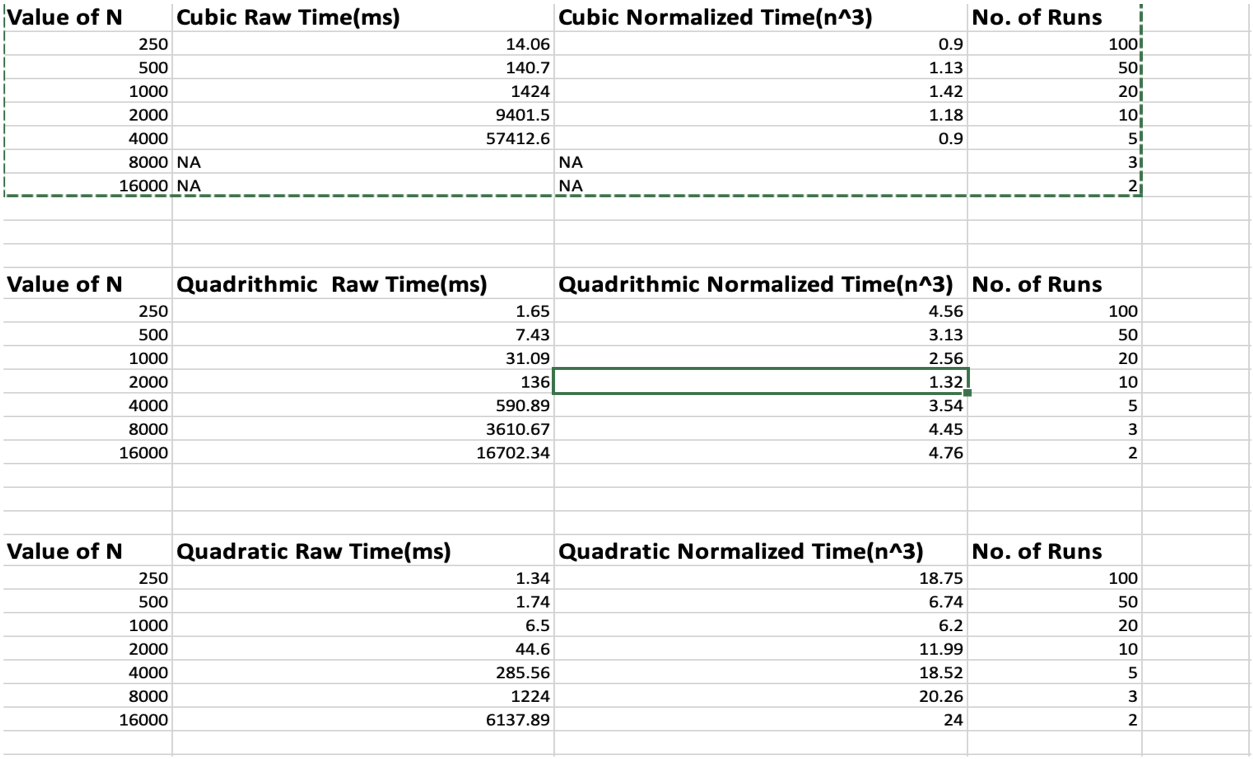


```
        if (description.equals( anObject: "ThreeSumCubic") && n > 4000) return;
        // FIXME
         final double t1 = new Benchmark_Timer<int[]>(
                description,
                fPre: null,
                fRun: function,
                fPost: function
        ).runFromSupplier(supplier,  m: runs);
        for (TimeLogger timeLogger : timeLoggers) timeLogger.log( time: t1,  N: n);
        // END
```

```
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^3):  .00
ThreeSumBenchmark: N=4000
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 5 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 5 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2 log n):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumCubic with 5 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^3):  .00
ThreeSumBenchmark: N=8000
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 3 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 3 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2 log n):  .00
ThreeSumBenchmark: N=16000
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadratic with 2 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2):  .00
2023-01-28 15:27:29 INFO  Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 2 runs
2023-01-28 15:27:29 INFO  TimeLogger - Raw time per run (mSec):  .00
2023-01-28 15:27:29 INFO  TimeLogger - Normalized time per run (n^2 log n):  .00
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.200 s
Finished at: 2023-01-28T15:27:30-05:00
------------------------------------------------------------------------
```

EVIDENCE:

To demonstrate the relationship between raw time and n value, I have attached a table and chart.

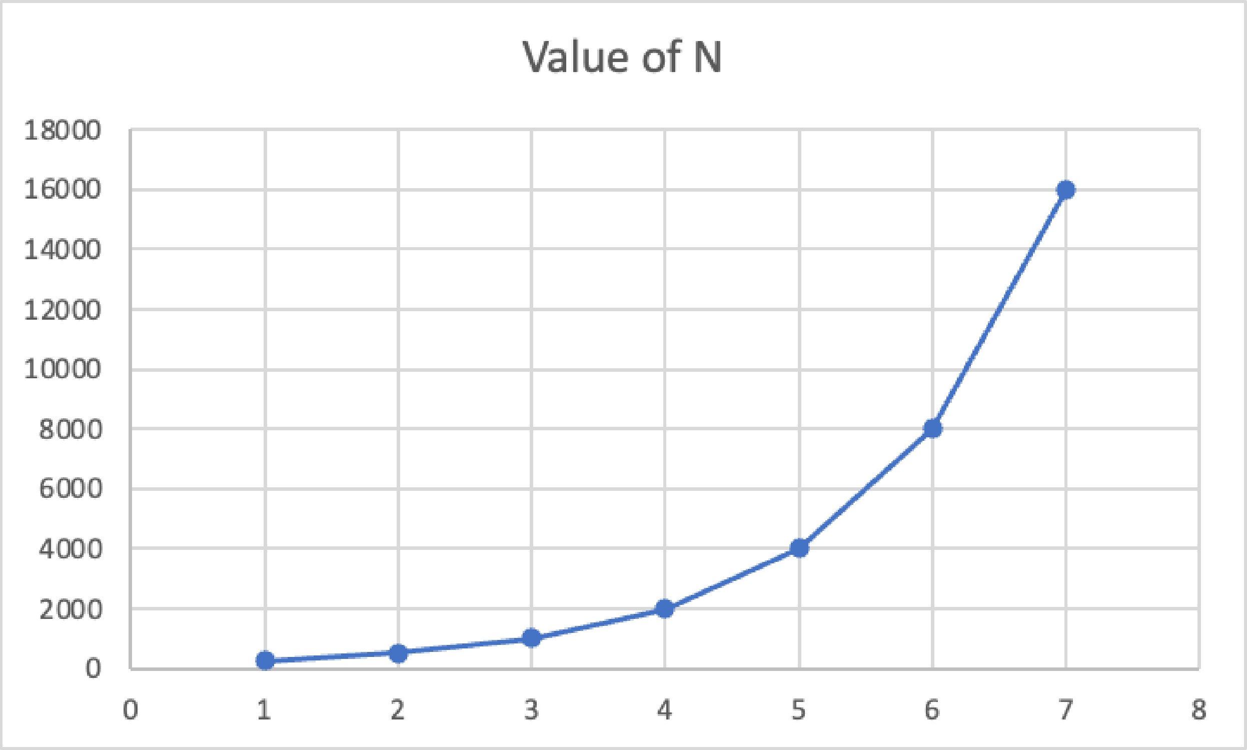


Value of N	Cubic Raw Time(ms)	Cubic Normalized Time(n^3)	No. of Runs
250	14.06	0.9	100
500	140.7	1.13	50
1000	1424	1.42	20
2000	9401.5	1.18	10
4000	57412.6	0.9	5
8000	NA	NA	3
16000	NA	NA	2

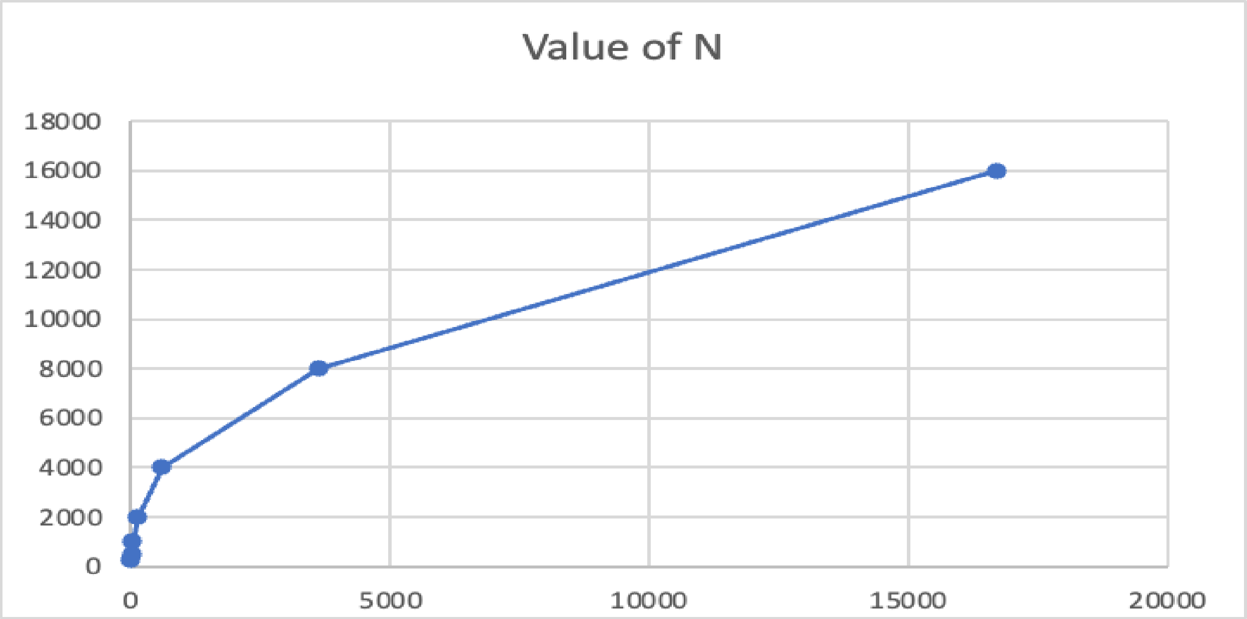
Value of N	Quadrithmic Raw Time(ms)	Quadrithmic Normalized Time(n^3)	No. of Runs
250	1.65	4.56	100
500	7.43	3.13	50
1000	31.09	2.56	20
2000	136	1.32	10
4000	590.89	3.54	5
8000	3610.67	4.45	3
16000	16702.34	4.76	2

Value of N	Quadratic Raw Time(ms)	Quadratic Normalized Time(n^3)	No. of Runs
250	1.34	18.75	100
500	1.74	6.74	50
1000	6.5	6.2	20
2000	44.6	11.99	10
4000	285.56	18.52	5
8000	1224	20.26	3
16000	6137.89	24	2

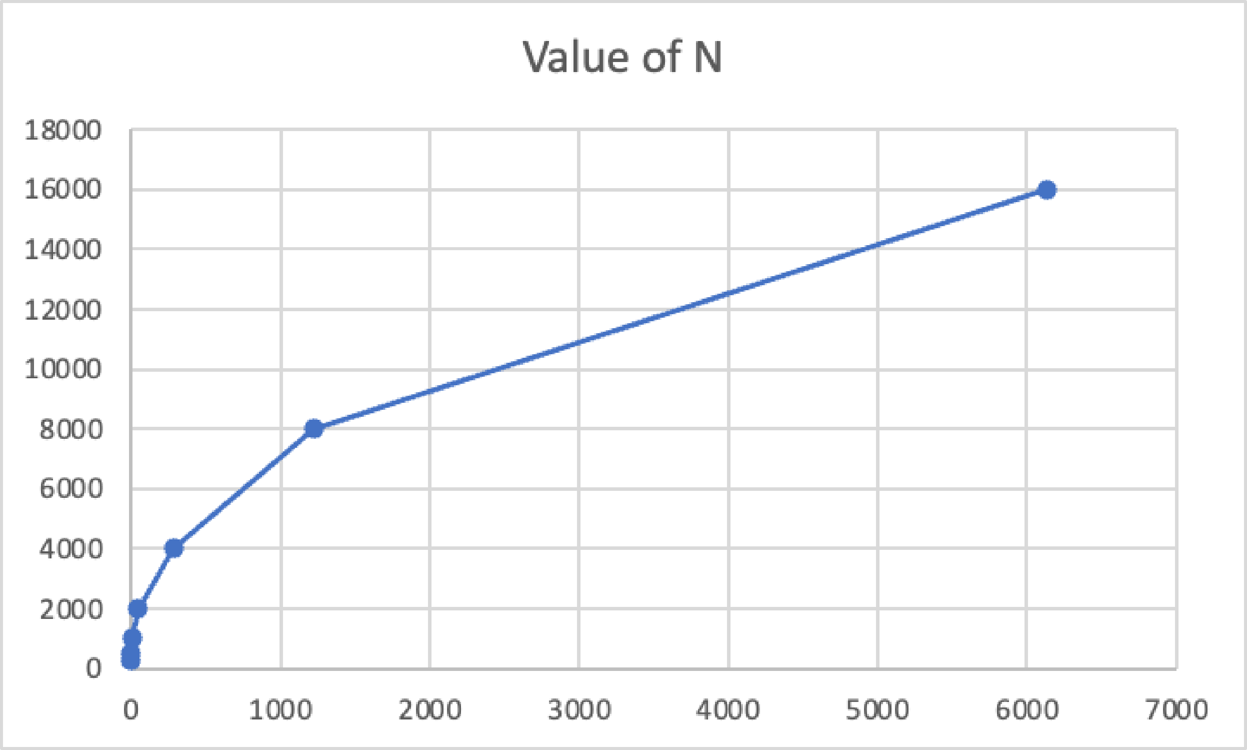
CUBIC



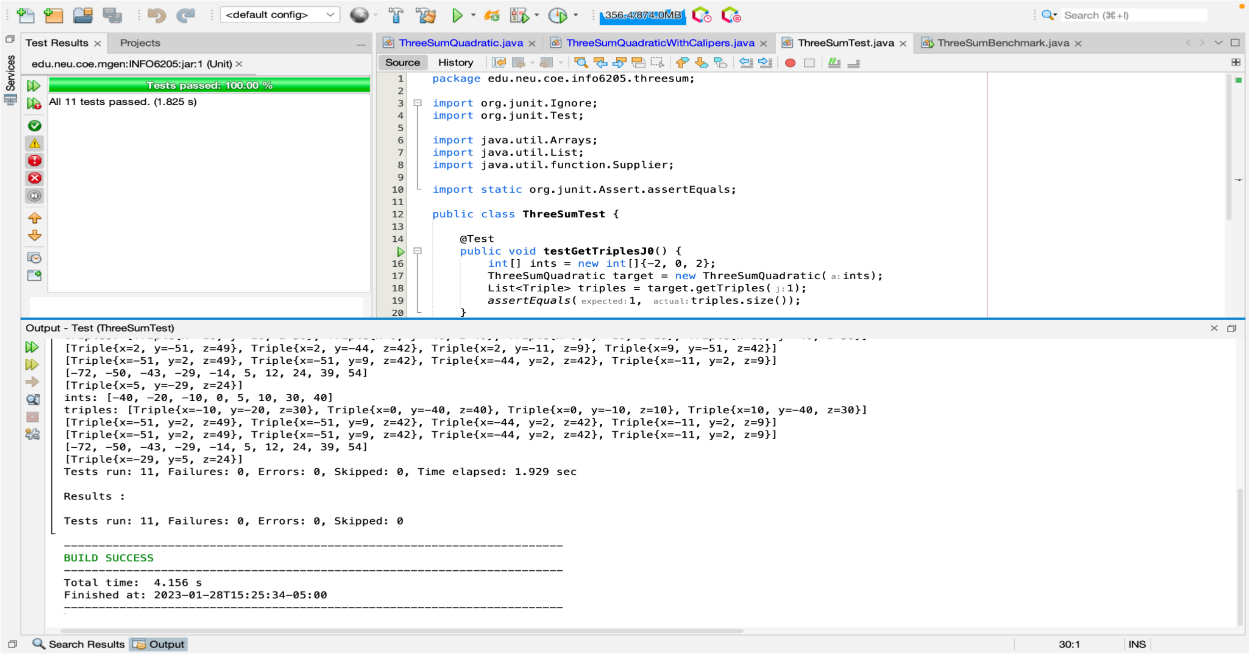
QUADRITHMIC



QUADRATIC



Unit Test Cases:



Tests passed: 100.00 %

All 11 tests passed. (1.825 s)

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

import org.junit.Ignore;
import org.junit.Test;

import java.util.Arrays;
import java.util.List;
import java.util.function.Supplier;

import static org.junit.Assert.assertEquals;

public class ThreeSumTest {

    @Test
    public void testGetTriplesJ0() {
        int[] ints = new int[]{-2, 0, 2};
        ThreeSumQuadratic target = new ThreeSumQuadratic( a:ints);
        List<Triple> triples = target.getTriples( j:1);
        assertEquals( expected:1,  actual:triples.size());
    }
```

```
[Triple{x=2, y=-51, z=49}, Triple{x=2, y=-44, z=42}, Triple{x=2, y=-11, z=9}, Triple{x=9, y=-51, z=42}]
[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=5, y=-29, z=24}]
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-10, y=-20, z=30}, Triple{x=0, y=-40, z=40}, Triple{x=0, y=-10, z=10}, Triple{x=10, y=-40, z=30}]
[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}]
Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.929 sec

Results :

Tests run: 11, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  4.156 s
Finished at: 2023-01-28T15:25:34-05:00
------------------------------------------------------------------------
```