Program Structures and Algorithms
Spring 2023(SEC 01)

NAME: Shivani Datar
NUID: 002772160

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

(a) evidence (screenshot) of your unit tests running (try to show the actual unit test code as well as the green strip).

(b) a spreadsheet showing your timing observations--using the doubling method for at least five values of N--for each of the algorithms (include cubic); Timing should be performed either with an actual stopwatch (e.g. your iPhone) or using the Stopwatch class in the repository.

(c) your brief explanation of why the quadratic method(s) work.

Relationship Conclusion:
After running 3-Sum code with different complexities, with the observed data we can see that Quadratic and Quadratic with Calipers are better in time complexity $O(n^2)$. Later on we have Quadrithmic with $O(n^2 \log n)$ and lastly cubic with O(n^3). We can verify this by using the time calculated for each execution using the Benchmark_Timer class.

3-sum Quadratic :-
Here we already have a middle index of one of the element whose sum is 0. We are using a while loop with 2 pointers such that the first one would be less than provided index and the second one would be more than the provided index. In this subspace we use complexity of O(N) and the function which is calling this subspace has a for loop giving us a quadratic time complexity.

```
4 usages  Shivani Datar
public List<Triple> getTriples(int j) {
    List<Triple> triples = new ArrayList<>();
    // FIXME : for each candidate, test if a[i] + a[j] + a[k] = 0.
    int l = 0;
    int h = length - 1;
    while (l < j && h>j) {
        Triple t = new Triple(a[l], a[j], a[h]);
        if (a[l]+a[j]+a[h] == 0) {
            triples.add(t);
            l++;
            h--;
        } else if (a[l]+a[j]+a[h] < 0)
            l++;
        else
            h--;
    }

    // END
    return triples;
}
```

3-Sum Quadrithmic:-
Here we already have a pair a[i] and a[j], we only need to find the third element, which is done by using the BinarySearch. This is divided into 2 subspaces, In first subspace we fix both elements a[i] and a[j] giving us a $O(n^2)$ time complexity. The second subspace has a complexity for O(log n) as we are using binary search to find the element. The total time taken here is quadrithmic or O(N^2 log n).

```
26 usages   Shivani Datar
public Triple[] getTriples() {
    List<Triple> triples = new ArrayList<>();
    for (int i = 0; i < length; i++)
        for (int j = i + 1; j < length; j++) {
            Triple triple = getTriple(i, j);
            if (triple != null) triples.add(triple);
        }
    Collections.sort(triples);
    return triples.stream().distinct().toArray(Triple[]::new);
}

1 usage   Shivani Datar
public Triple getTriple(int i, int j) {
    int index = Arrays.binarySearch(a,  key: -a[i] - a[j]);
    if (index >= 0 && index > j) return new Triple(a[i], a[j], a[index]);
    else return null;

}
```

3-Sum Quadratic with Calipers :-
In this scenario we have the first index out of the three elements, here as well we use two pointers method to calculate remaining two elements but the computation is less heavy as we know the first element's index and it's a sorted array we know for sure that the remaining two elements are after the first element's index. Here we are provided a function, which we use to compare the sum with 0.

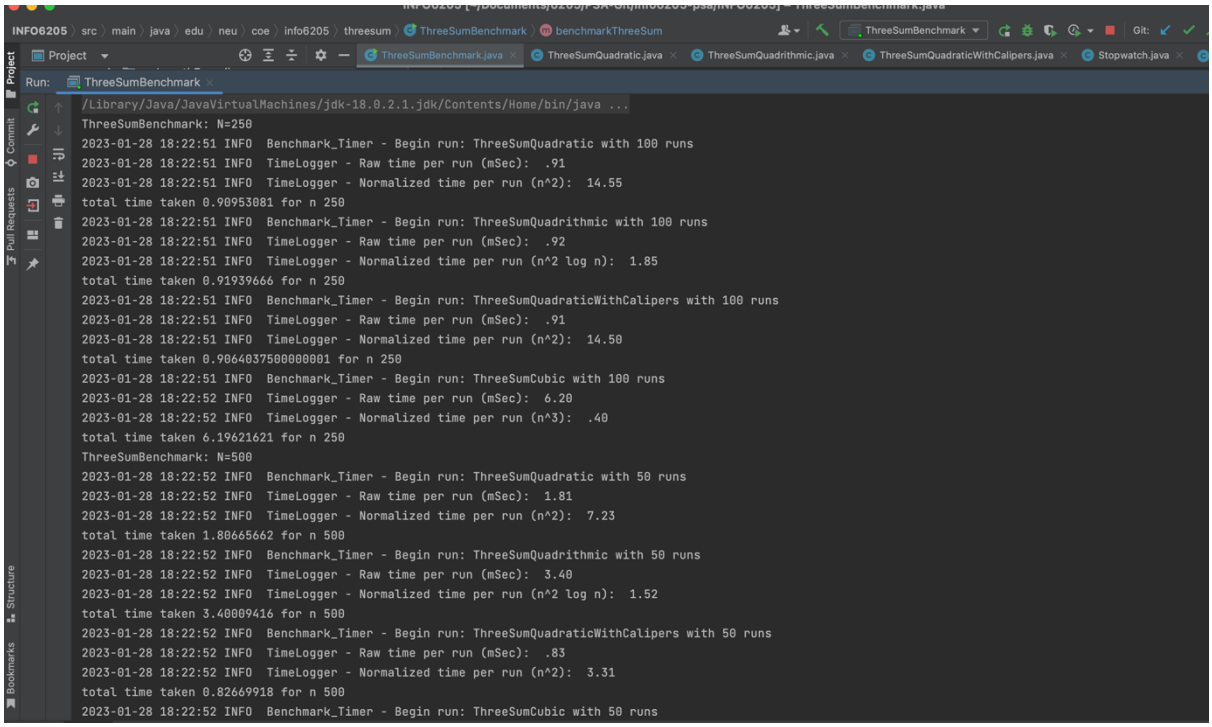
```
Shivani Datar
public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) {
    List<Triple> triples = new ArrayList<>();
    // FIXME : use function to qualify triples and to navigate otherwise.
    int l = i + 1;
    int h = a.length - 1;
    int x = a[i];
    while (l < h) {
        Triple t = new Triple(x, a[l], a[h]);
        if (function.apply(t) == 0) {
            triples.add(t);
            l++;
            h--;
        } else if (function.apply(t) < 0)
            l++;
        else
            h--;
    }
    // END
    return triples;
```

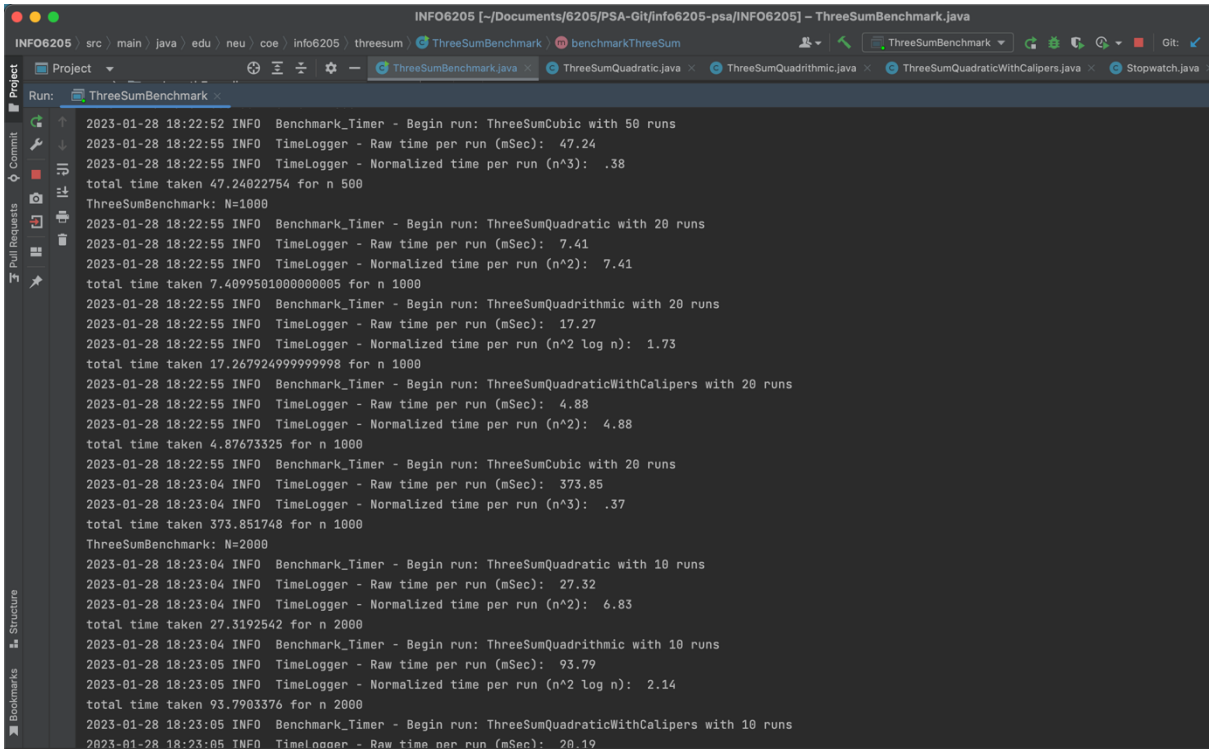
Changes in ThreeSumBenchmark file :-

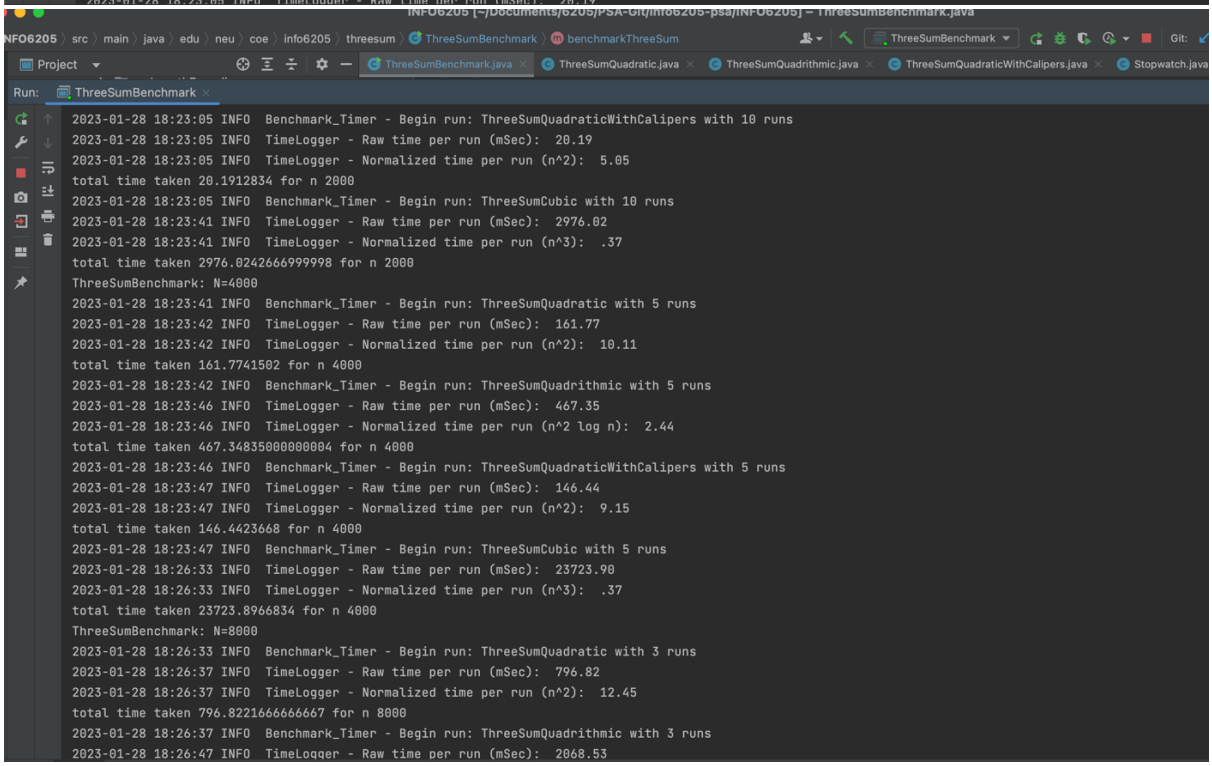
In the benchmark file we have different N values and a supplier which helps us to construct a random array given a seed for particular N and we also have total number of runs performed on the constructed array. This is approached by initializing a object of Benchmark_Timer class. This class has 3 functions fpre,frun and fpost as properties. Fpre is a function which is supposed to be executed before we run our function. fRun is a function which we actually want to run and fPost is some function if we want to run later on the main executed function. In our case we just to run the getTriples function from each different complexity 3-Sum class, hence our fpre and fpost are null. After initializing the object, we call runfromsupplier method, where we are passing the description and a lambda function, this is sent to the new runtime initialized object of Timer class, using the repeat method we can get the time utilised by our function to run in miliseconds. We later on use the timeloggers to print time in mili seconds and normalized time with the time complexity observed.

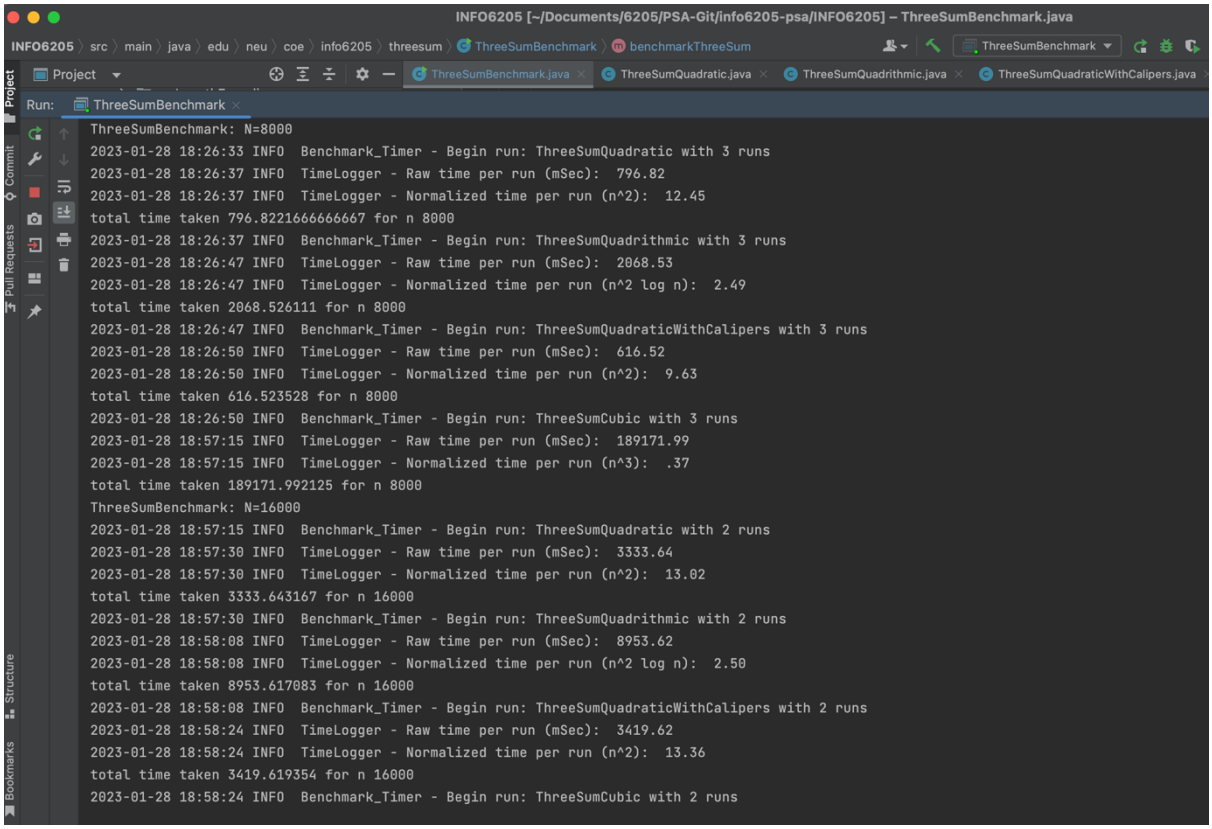
```
4 usages  Shivani Datar *
private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLogger
    if (description.equals("ThreeSumQuadratic") && n >=250){
            Benchmark_Timer time1 = new Benchmark_Timer(description,function);
            double timeQ = time1.runFromSupplier(supplier,runs);
            timeLoggers[0].log(timeQ,n);
            timeLoggers[1].log(timeQ,n);
            System.out.println("total time taken "+ timeQ+ " for n "+n);
    }
    else if(description.equals("ThreeSumQuadrithmic") && n >=250){
        Benchmark_Timer time2 = new Benchmark_Timer(description,function);
        double timeL = time2.runFromSupplier(supplier,runs);
        timeLoggers[0].log(timeL,n);
        timeLoggers[1].log(timeL,n);
        System.out.println("total time taken "+ timeL+ " for n "+n);
    }
    else if(description.equals("ThreeSumQuadraticWithCalipers") && n >=250){
        Benchmark_Timer time4 = new Benchmark_Timer(description,function);
        double timeQC = time4.runFromSupplier(supplier,runs);
        timeLoggers[0].log(timeQC,n);
        timeLoggers[1].log(timeQC,n);
        System.out.println("total time taken "+ timeQC+ " for n "+n);
    }
    else if(description.equals("ThreeSumCubic") && n >=250){
        Benchmark_Timer time3 = new Benchmark_Timer(description,function);
        double timeC = time3.runFromSupplier(supplier,runs);
        timeLoggers[0].log(timeC,n);
        timeLoggers[1].log(timeC,n);
        System.out.println("total time taken "+ timeC + " for n "+n);
    }
```

Output for the benchmark code :-




INFO6205 [~/Documents/6205/PSA-Git/info6205-psa/INFO6205] – ThreeSumBenchmark.java
Run: ThreeSumBenchmark
2023-01-28 18:22:52 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 18:22:55 INFO TimeLogger - Raw time per run (mSec): 47.24
2023-01-28 18:22:55 INFO TimeLogger - Normalized time per run (n^3): .38
total time taken 47.24022754 for n 500
ThreeSumBenchmark: N=1000
2023-01-28 18:22:55 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 20 runs
2023-01-28 18:22:55 INFO TimeLogger - Raw time per run (mSec): 7.41
2023-01-28 18:22:55 INFO TimeLogger - Normalized time per run (n^2): 7.41
total time taken 7.4099501000000005 for n 1000
2023-01-28 18:22:55 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 20 runs
2023-01-28 18:22:55 INFO TimeLogger - Raw time per run (mSec): 17.27
2023-01-28 18:22:55 INFO TimeLogger - Normalized time per run (n^2 log n): 1.73
total time taken 17.26792499999998 for n 1000
2023-01-28 18:22:55 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 20 runs
2023-01-28 18:22:55 INFO TimeLogger - Raw time per run (mSec): 4.88
2023-01-28 18:22:55 INFO TimeLogger - Normalized time per run (n^2): 4.88
total time taken 4.87673325 for n 1000
2023-01-28 18:22:55 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 20 runs
2023-01-28 18:23:04 INFO TimeLogger - Raw time per run (mSec): 373.85
2023-01-28 18:23:04 INFO TimeLogger - Normalized time per run (n^3): .37
total time taken 373.851748 for n 1000
ThreeSumBenchmark: N=2000
2023-01-28 18:23:04 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 10 runs
2023-01-28 18:23:04 INFO TimeLogger - Raw time per run (mSec): 27.32
2023-01-28 18:23:04 INFO TimeLogger - Normalized time per run (n^2): 6.83
total time taken 27.3192542 for n 2000
2023-01-28 18:23:04 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 10 runs
2023-01-28 18:23:05 INFO TimeLogger - Raw time per run (mSec): 93.79
2023-01-28 18:23:05 INFO TimeLogger - Normalized time per run (n^2 log n): 2.14
total time taken 93.7903376 for n 2000
2023-01-28 18:23:05 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 10 runs


Run: ThreeSumBenchmark
2023-01-28 18:23:05 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 10 runs
2023-01-28 18:23:05 INFO TimeLogger - Raw time per run (mSec): 20.19
2023-01-28 18:23:05 INFO TimeLogger - Normalized time per run (n^2): 5.05
total time taken 20.1912834 for n 2000
2023-01-28 18:23:05 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 10 runs
2023-01-28 18:23:41 INFO TimeLogger - Raw time per run (mSec): 2976.02
2023-01-28 18:23:41 INFO TimeLogger - Normalized time per run (n^3): .37
total time taken 2976.0242666999998 for n 2000
ThreeSumBenchmark: N=4000
2023-01-28 18:23:41 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 5 runs
2023-01-28 18:23:42 INFO TimeLogger - Raw time per run (mSec): 161.77
2023-01-28 18:23:42 INFO TimeLogger - Normalized time per run (n^2): 10.11
total time taken 161.7741502 for n 4000
2023-01-28 18:23:42 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 5 runs
2023-01-28 18:23:46 INFO TimeLogger - Raw time per run (mSec): 467.35
2023-01-28 18:23:46 INFO TimeLogger - Normalized time per run (n^2 log n): 2.44
total time taken 467.34835000000004 for n 4000
2023-01-28 18:23:46 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 5 runs
2023-01-28 18:23:47 INFO TimeLogger - Raw time per run (mSec): 146.44
2023-01-28 18:23:47 INFO TimeLogger - Normalized time per run (n^2): 9.15
total time taken 146.4423668 for n 4000
2023-01-28 18:23:47 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 5 runs
2023-01-28 18:26:33 INFO TimeLogger - Raw time per run (mSec): 23723.90
2023-01-28 18:26:33 INFO TimeLogger - Normalized time per run (n^3): .37
total time taken 23723.8966834 for n 4000
ThreeSumBenchmark: N=8000
2023-01-28 18:26:33 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 3 runs
2023-01-28 18:26:37 INFO TimeLogger - Raw time per run (mSec): 796.82
2023-01-28 18:26:37 INFO TimeLogger - Normalized time per run (n^2): 12.45
total time taken 796.8221666666667 for n 8000
2023-01-28 18:26:37 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 3 runs
2023-01-28 18:26:47 INFO TimeLogger - Raw time per run (mSec): 2068.53



Evidence to support that conclusion:
The following table is a consolidated count of the time required for each algorithm with respective amount of N. The 3-sum cubic algorithm for n = 16000 took lot of time. It was in process for more that 1 and half hour and then I exited the process by stopping the code.

Table :-

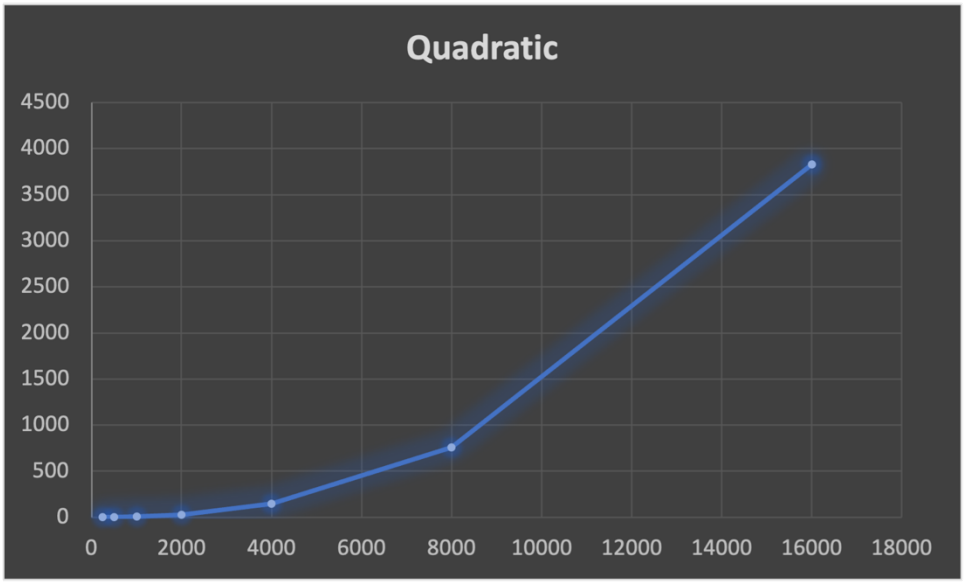
N	Quadratic(ms)	Quadrithimic(ms)	Cubic(ms)	QuadracticCaliper(ms)
250	0.91	0.92	6.2	0.91
500	1.81	3.4	47.24	0.83
1000	7.41	17.27	373.85	4.88
2000	27.32	93.79	2976.02	20.19
4000	161.77	467.35	23723.9	146.44
8000	796.82	2068.53	189171.99	616.52
16000	3333.64	8953.62		3419.62

Graphical Representation:

Graph for Quadratic:-

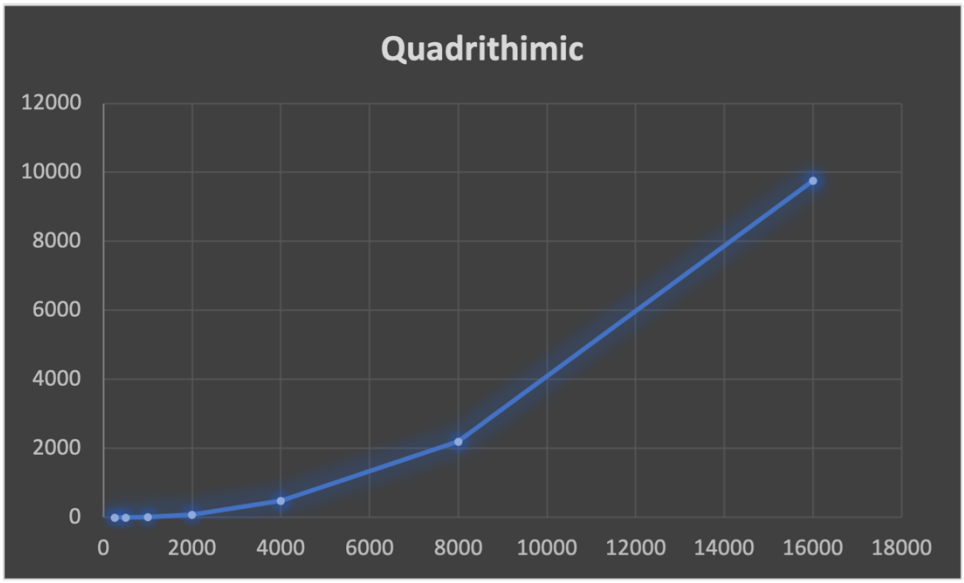
Observations:-

N	Quadratic(ms)
250	0.91
500	1.81
1000	7.41
2000	27.32
4000	161.77
8000	796.82
16000	3333.64



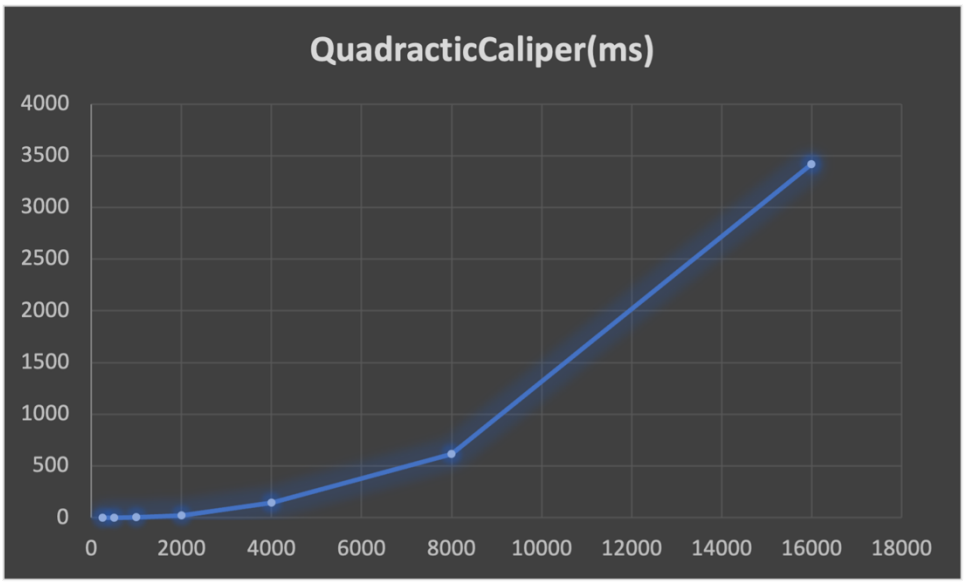
Graph for Quadrithmic:-

N	Quadrithimic
250	0.79
500	3.23
1000	17.26
2000	86.9
4000	476.3
8000	2204.5
16000	9763.8



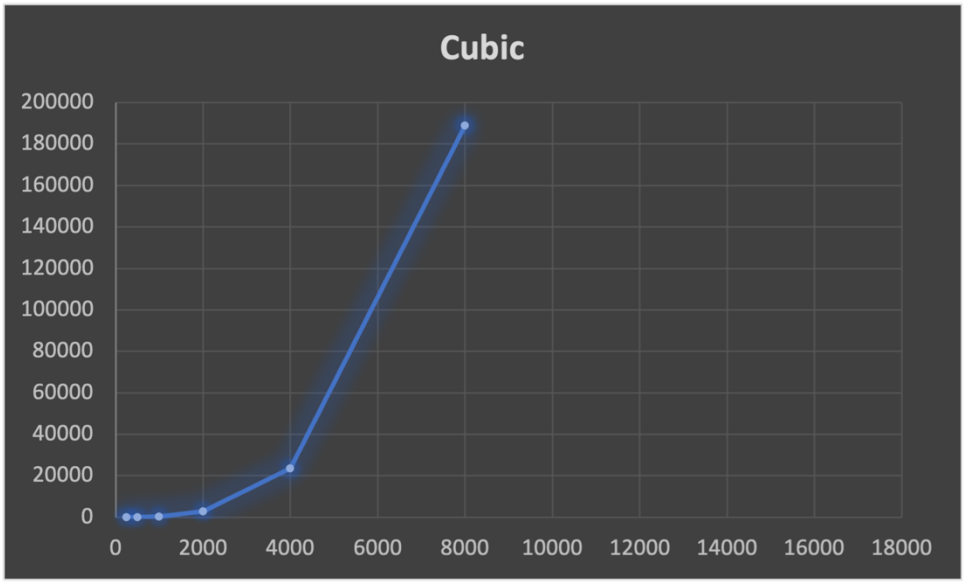
Graph for Quadratic with calipers :-

N	QuadracticCaliper(ms)
250	0.91
500	0.83
1000	4.88
2000	20.19
4000	146.44
8000	616.52
16000	3419.62

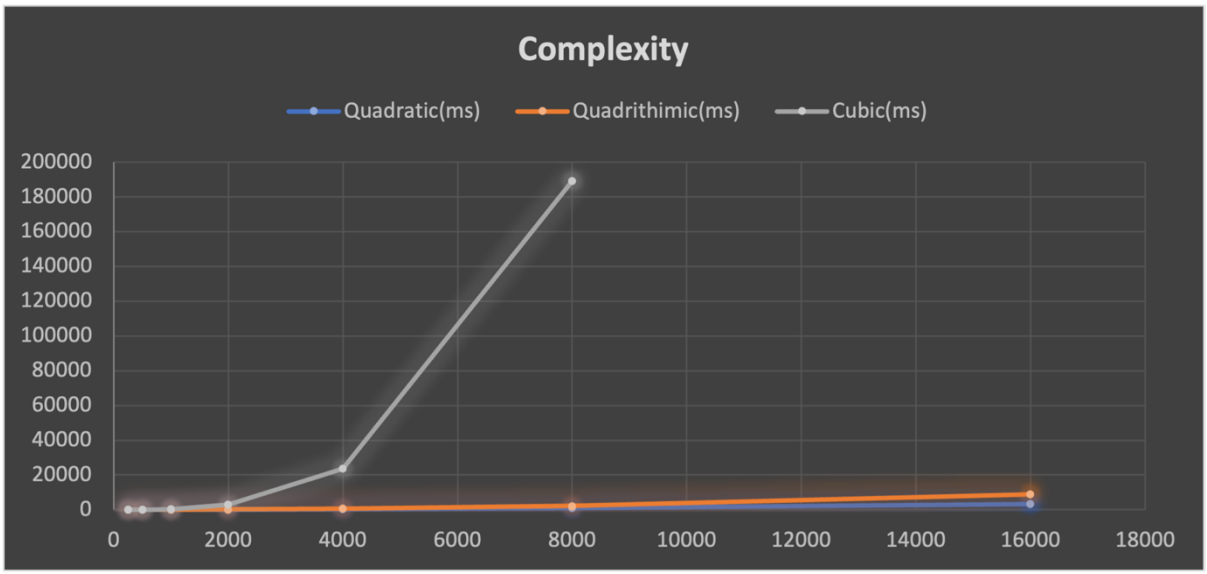


Graph for cubic:-

N	Cubic
250	6.12
500	48.16
1000	374.27
2000	2977.68
4000	23750.67
8000	189061.88
16000	

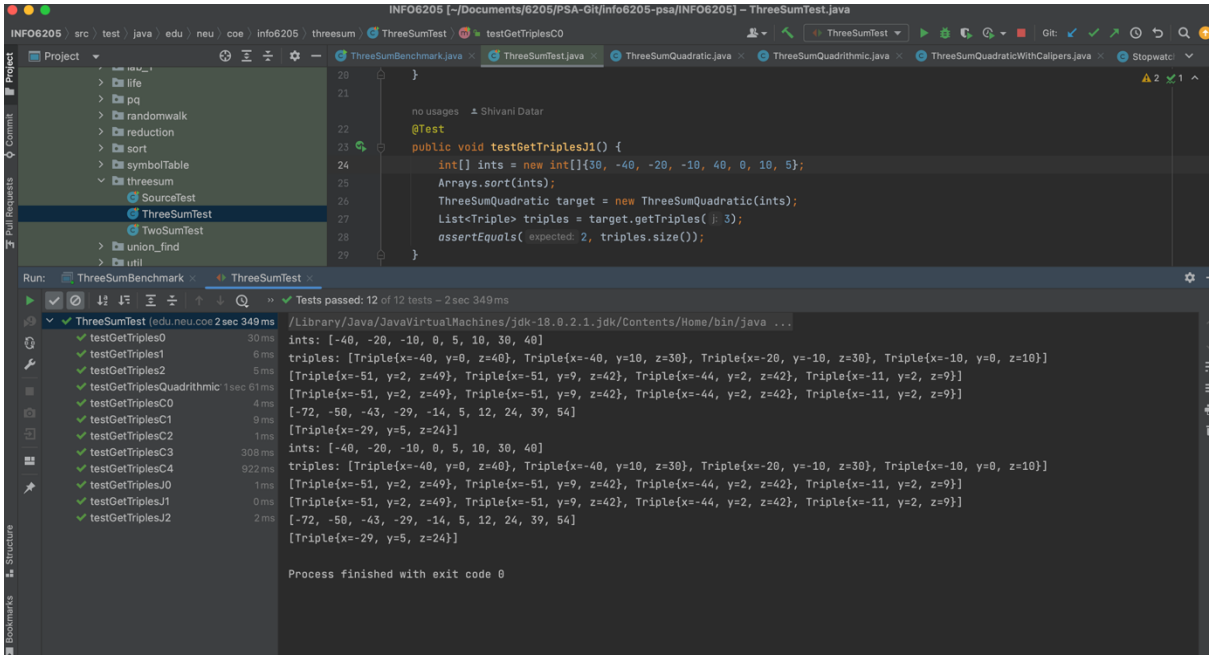


Graph For Complexity: -



Unit Test Screenshots:

Following is the screenshot of all 12 test cases clearing.


INFO6205 [~/Documents/6205/PSA-Git/info6205-psa/INFO6205] – ThreeSumTest.java
INFO6205 › src › test › java › edu › neu › coe › info6205 › threesum › ThreeSumTest › testGetTriplesC0
ThreeSumTest
Git:
Project
ThreeSumBenchmark.java
ThreeSumTest.java
ThreeSumQuadratic.java
ThreeSumQuadrithmic.java
ThreeSumQuadraticWithCalipers.java
Stopwatch
life
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
}
no usages Shivani Datar
@Test
public void testGetTriplesJ1() {
int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
Arrays.sort(ints);
ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
List<Triple> triples = target.getTriples(j: 3);
assertEquals(expected: 2, triples.size());
}
Run:
ThreeSumBenchmark
ThreeSumTest
Tests passed: 12 of 12 tests – 2 sec 349 ms
ThreeSumTest (edu.neu.coe 2 sec 349 ms
testGetTriples0 30 ms
testGetTriples1 6 ms
testGetTriples2 5 ms
testGetTriplesQuadrithmic 1 sec 61 ms
testGetTriplesC0 4 ms
testGetTriplesC1 9 ms
testGetTriplesC2 1 ms
testGetTriplesC3 308 ms
testGetTriplesC4 922 ms
testGetTriplesJ0 1 ms
testGetTriplesJ1 0 ms
testGetTriplesJ2 2 ms
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
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}]
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
Process finished with exit code 0