Program Structures and Algorithms
Fall 2024

NAME: Sanskruti Manoria
NUID: 002643300
GITHUB LINK: https://github.com/sannskruti/INFO6205

# Assignment 2

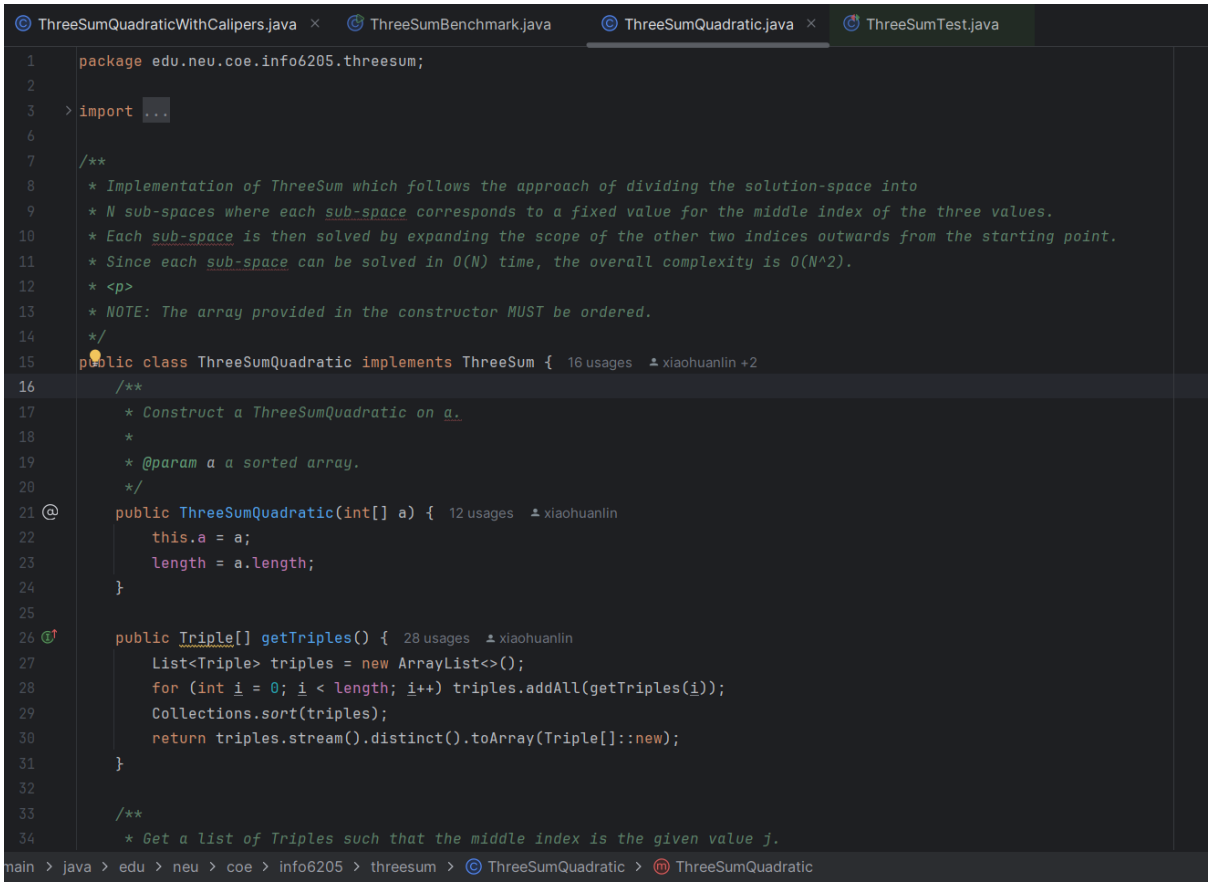
## Task:

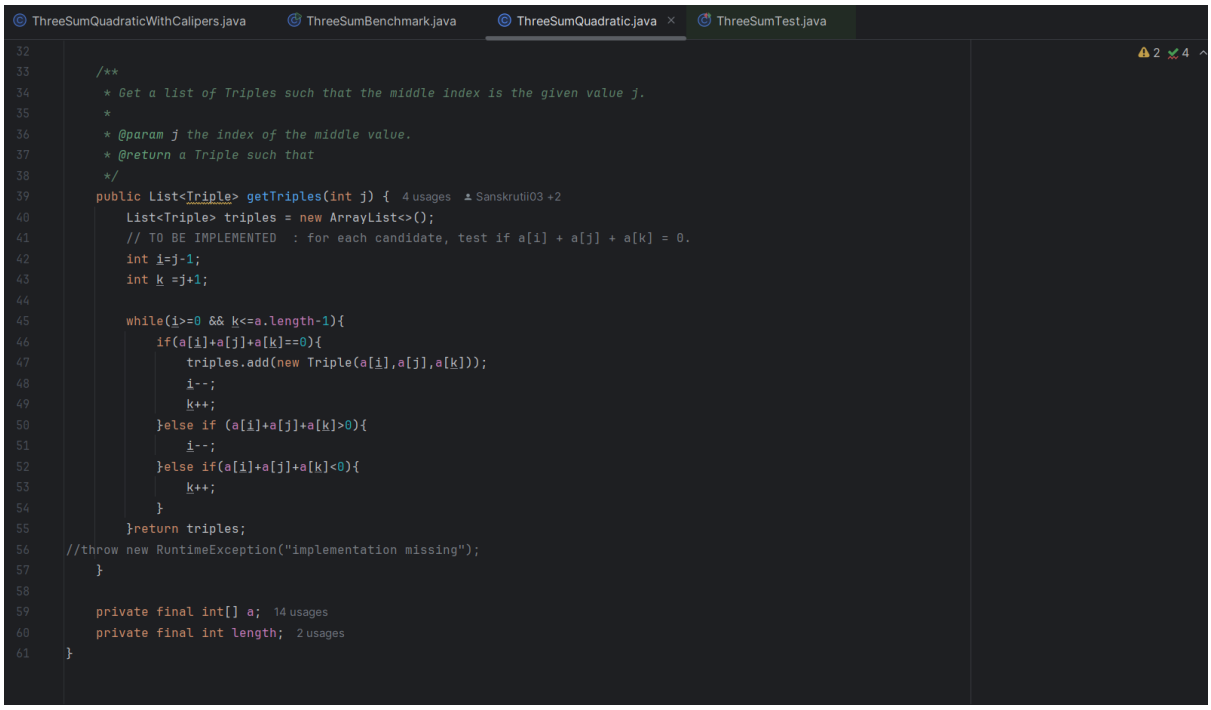
Solve 3-SUM using the *Quadrithmic*, *Quadratic*, and (bonus point) *quadratic-WithCalipers* approaches, as shown in skeleton code in the repository.

(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.

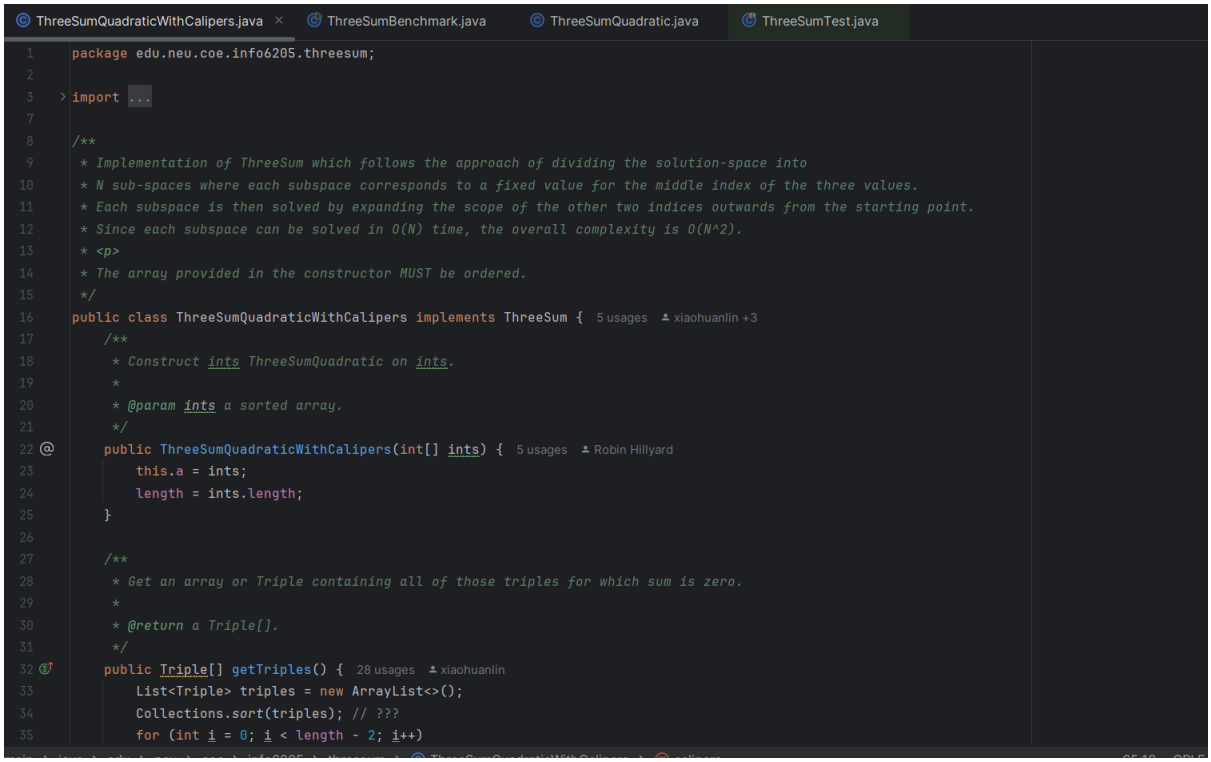
## a.Code Screenshots:

### 1. Quadratic

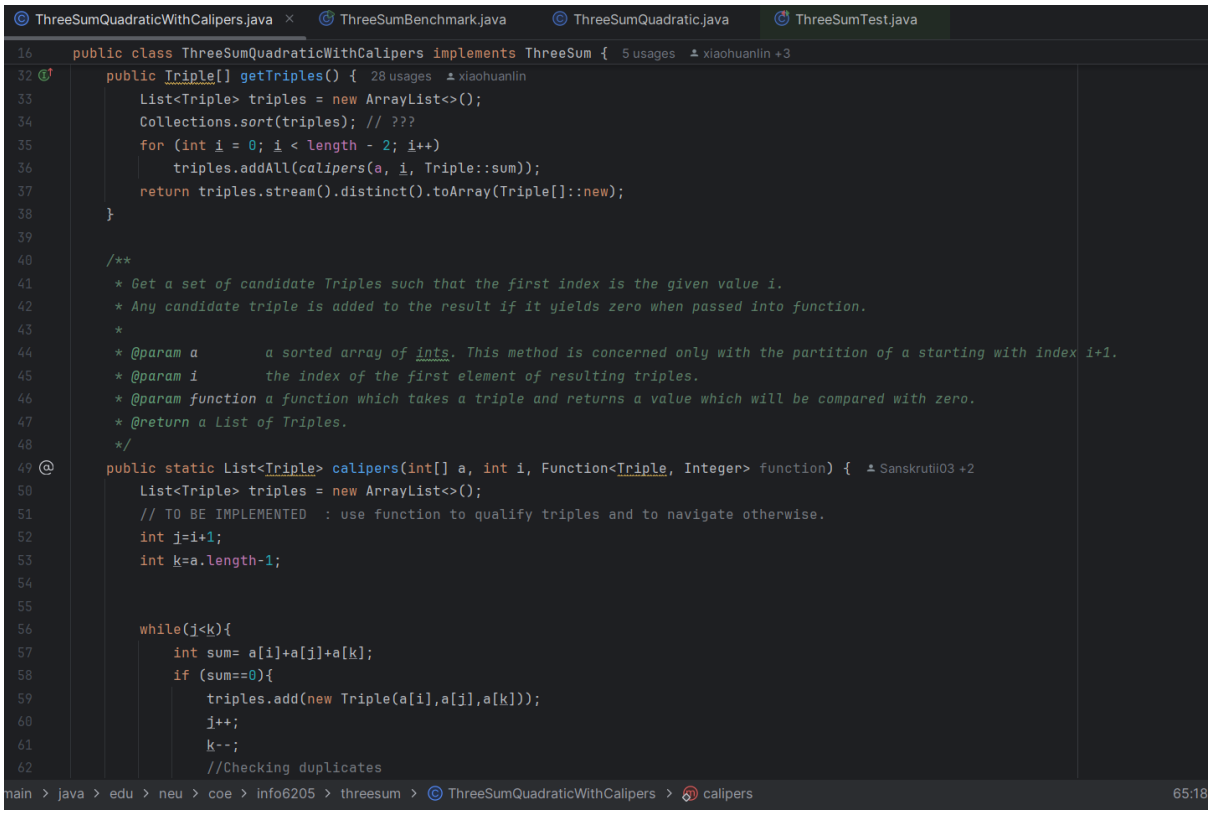


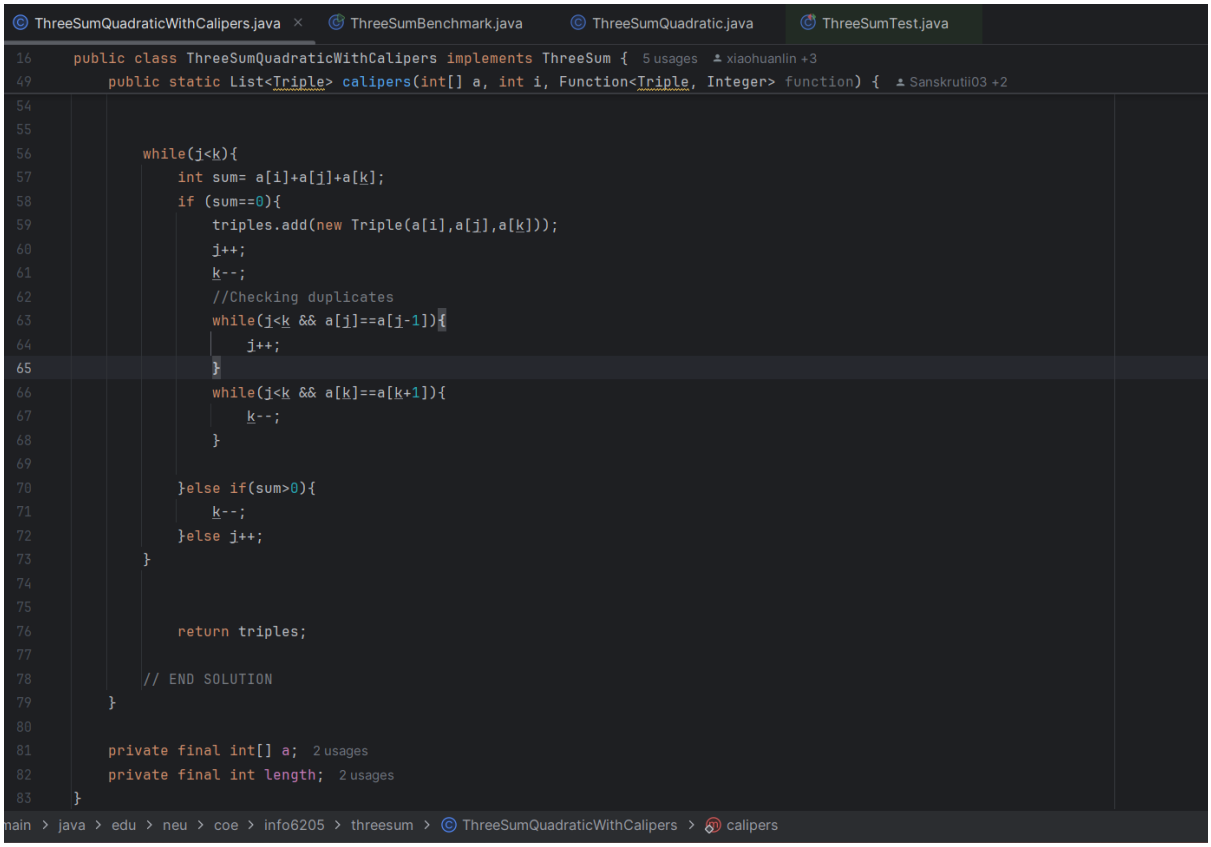

## 2. QuadriticWithCalipers



```
ThreeSumQuadraticWithCalipers.java | ThreeSumBenchmark.java | ThreeSumQuadratic.java | ThreeSumTest.java

1  package edu.neu.coe.info6205.threesum;
2
3  import ...
7
8  /**
9   * Implementation of ThreeSum which follows the approach of dividing the solution-space into
10  * N sub-spaces where each subspace corresponds to a fixed value for the middle index of the three values.
11  * Each subspace is then solved by expanding the scope of the other two indices outwards from the starting point.
12  * Since each subspace can be solved in O(N) time, the overall complexity is O(N^2).
13  * <p>
14  * The array provided in the constructor MUST be ordered.
15  */
16 public class ThreeSumQuadraticWithCalipers implements ThreeSum {  5 usages  xiaohuanlin +3
17     /**
18      * Construct ints ThreeSumQuadratic on ints.
19      *
20      * @param ints a sorted array.
21      */
22     public ThreeSumQuadraticWithCalipers(int[] ints) {  5 usages  Robin Hillyard
23         this.a = ints;
24         length = ints.length;
25     }
26
27     /**
28      * Get an array or Triple containing all of those triples for which sum is zero.
29      *
30      * @return a Triple[].
31      */
32     public Triple[] getTriples() {  28 usages  xiaohuanlin
33         List<Triple> triples = new ArrayList<>();
34         Collections.sort(triples); // ???
35         for (int i = 0; i < length - 2; i++)
```



```
ThreeSumQuadraticWithCalipers.java | ThreeSumBenchmark.java | ThreeSumQuadratic.java | ThreeSumTest.java

16 public class ThreeSumQuadraticWithCalipers implements ThreeSum {  5 usages  xiaohuanlin +3
32     public Triple[] getTriples() {  28 usages  xiaohuanlin
33         List<Triple> triples = new ArrayList<>();
34         Collections.sort(triples); // ???
35         for (int i = 0; i < length - 2; i++)
36             triples.addAll(calipers(a, i, Triple::sum));
37         return triples.stream().distinct().toArray(Triple[]::new);
38     }
39
40     /**
41      * Get a set of candidate Triples such that the first index is the given value i.
42      * Any candidate triple is added to the result if it yields zero when passed into function.
43      *
44      * @param a        a sorted array of ints. This method is concerned only with the partition of a starting with index i+1.
45      * @param i        the index of the first element of resulting triples.
46      * @param function a function which takes a triple and returns a value which will be compared with zero.
47      * @return a List of Triples.
48      */
49     public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) {  Sanskrutii03 +2
50         List<Triple> triples = new ArrayList<>();
51         // TO BE IMPLEMENTED  : use function to qualify triples and to navigate otherwise.
52         int j=i+1;
53         int k=a.length-1;
54
55
56         while(j<k){
57             int sum= a[i]+a[j]+a[k];
58             if (sum==0){
59                 triples.add(new Triple(a[i],a[j],a[k]));
60                 j++;
61                 k--;
62                 //Checking duplicates

main > java > edu > neu > coe > info6205 > threesum > ThreeSumQuadraticWithCalipers > calipers   65:18
```




## 3. ThreeSumBenchmark
## (for comparison in Timing, point b)

ThreeSumQuadraticWithCalipers.java | ThreeSumBenchmark.java | ThreeSumQuadratic.java | ThreeSumTest.java

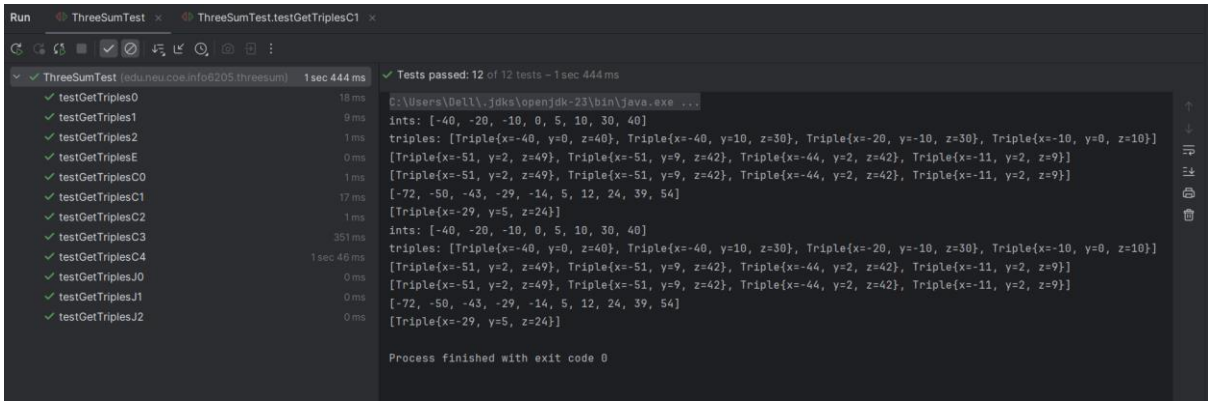
```java
package edu.neu.coe.info6205.threesum;

import ...

public class ThreeSumBenchmark {  xiaohuanlin +2
    public ThreeSumBenchmark(int runs, int n, int m) {  7 usages  xiaohuanlin
        this.runs = runs;
        this.supplier = new Source(n, m).intsSupplier( safetyFactor: 10);
        this.n = n;
    }

    public void runBenchmarks() {  7 usages  xiaohuanlin
        System.out.println("ThreeSumBenchmark: N=" + n);
        benchmarkThreeSum( description: "ThreeSumQuadratic", (xs) -> new ThreeSumQuadratic(xs).getTriples(), n, timeLogg
        benchmarkThreeSum( description: "ThreeSumQuadrithmic", (xs) -> new ThreeSumQuadrithmic(xs).getTriples(), n, time
        benchmarkThreeSum( description: "ThreeSumCubic", (xs) -> new ThreeSumCubic(xs).getTriples(), n, timeLoggersCubic
    }

    public static void main(String[] args) {  xiaohuanlin
        new ThreeSumBenchmark( runs: 100,  n: 250,  m: 250).runBenchmarks();
        new ThreeSumBenchmark( runs: 50,  n: 500,  m: 500).runBenchmarks();
        new ThreeSumBenchmark( runs: 20,  n: 1000,  m: 1000).runBenchmarks();
        new ThreeSumBenchmark( runs: 10,  n: 2000,  m: 2000).runBenchmarks();
        new ThreeSumBenchmark( runs: 5,  n: 4000,  m: 4000).runBenchmarks();
        new ThreeSumBenchmark( runs: 3,  n: 8000,  m: 8000).runBenchmarks();
        new ThreeSumBenchmark( runs: 2,  n: 16000,  m: 16000).runBenchmarks();
    }

    private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLogger
        //if (description.equals("ThreeSumCubic") && n > 4000) return;
        // TO BE IMPLEMENTED
```

```java
public class ThreeSumBenchmark {  xiaohuanlin +2
    private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLo
        //if (description.equals("ThreeSumCubic") && n > 4000) return;
        // TO BE IMPLEMENTED

        int[] ints= supplier.get();
        if(description.equals("ThreeSumQuadratic")){
            Stopwatch stopwatch=new Stopwatch();
            new ThreeSumQuadratic(ints).getTriples();

            System.out.println(description + ":"+ stopwatch.lap());
        }
        else if (description.equals("ThreeSumQuadrithmic")){
            Stopwatch stopwatch=new Stopwatch();
            new ThreeSumQuadrithmic(ints).getTriples();
            System.out.println(description + ":"+ stopwatch.lap());

        }
        else if (description.equals("ThreeSumQuadraticWithCalipers")){
            Stopwatch stopwatch=new Stopwatch();
            new ThreeSumQuadraticWithCalipers(ints).getTriples();
            System.out.println(description + ":"+ stopwatch.lap());

        }else if (description.equals("ThreeSumCubic")){
            Stopwatch stopwatch=new Stopwatch();
            new ThreeSumCubic(ints).getTriples();
            System.out.println(description + ":"+ stopwatch.lap());
        }
```

```java
public class ThreeSumBenchmark {  xiaohuanlin +2
    private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final Time

//throw new RuntimeException("implementation missing");
    }

    private final static TimeLogger[] timeLoggersCubic = {  1 usage
            new TimeLogger( prefix: "Raw time per run (mSec): ",  minimumComparisons: null),
            new TimeLogger( prefix: "Normalized time per run (n^3): ", n -> 1.0 / 6 * n * n * n)
    };
    private final static TimeLogger[] timeLoggersQuadrithmic = {  1 usage
            new TimeLogger( prefix: "Raw time per run (mSec): ",  minimumComparisons: null),
            new TimeLogger( prefix: "Normalized time per run (n^2 log n): ", n -> n * n * Utilities.lg(n))
    };
    private final static TimeLogger[] timeLoggersQuadratic = {  1 usage
            new TimeLogger( prefix: "Raw time per run (mSec): ",  minimumComparisons: null),
            new TimeLogger( prefix: "Normalized time per run (n^2): ", n -> 1.0 / 2 * n * n)
    };

    private final int runs;  1 usage
    private final Supplier<int[]> supplier;  2 usages
    private final int n;  5 usages
}
```

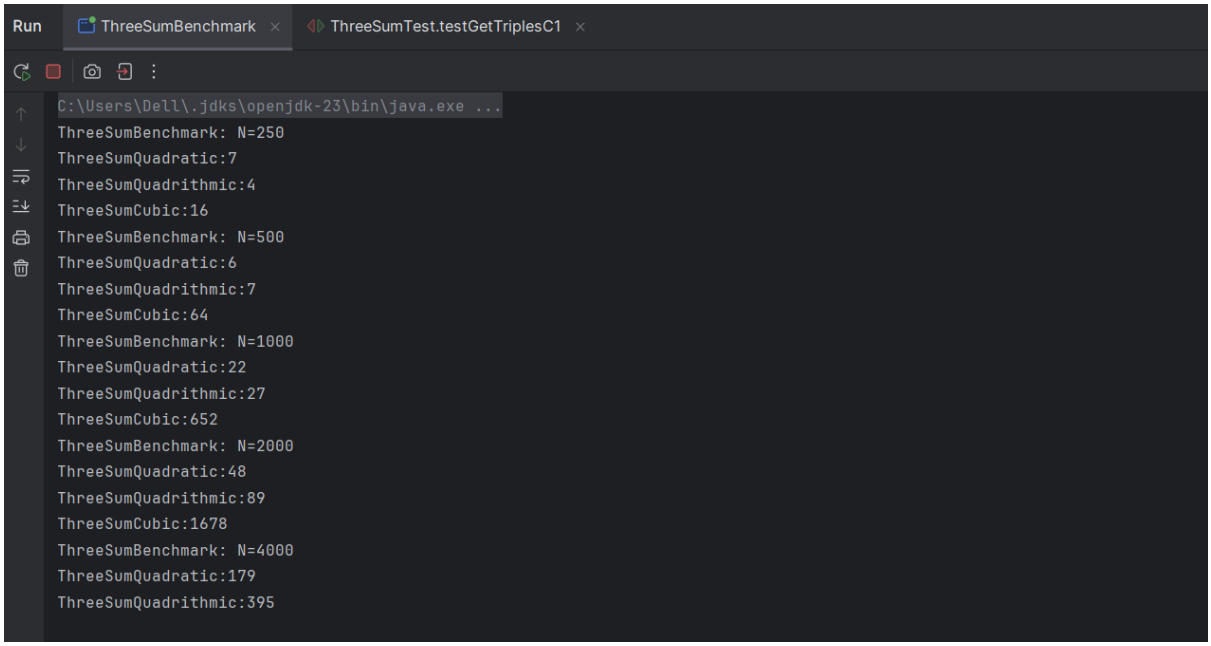
# Output-

ThreeSumQuadratic and  ThreeSumQuadraticWithCaliper (Quadrithmic already implemented)

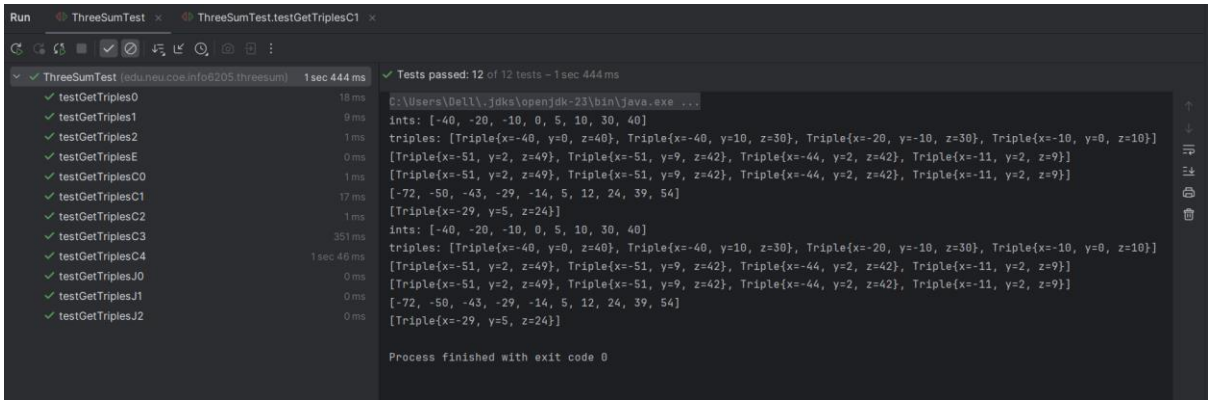



**Benchmark class for all 4-method comparison-**

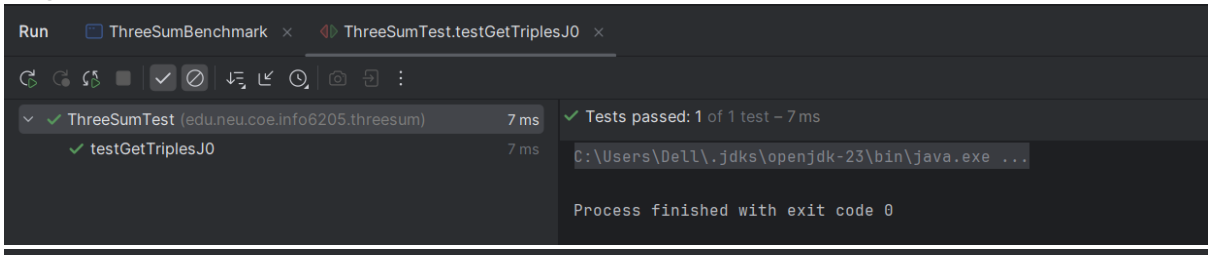
ThreeSumQuadratic and  ThreeSumQuadraticWithCaliper, Quadrithmic and Cubic

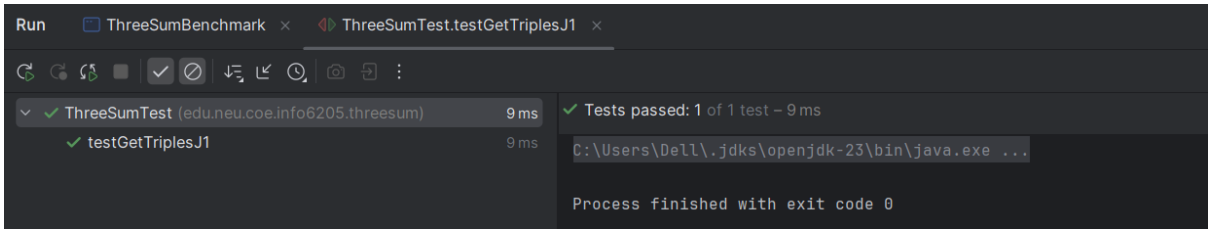


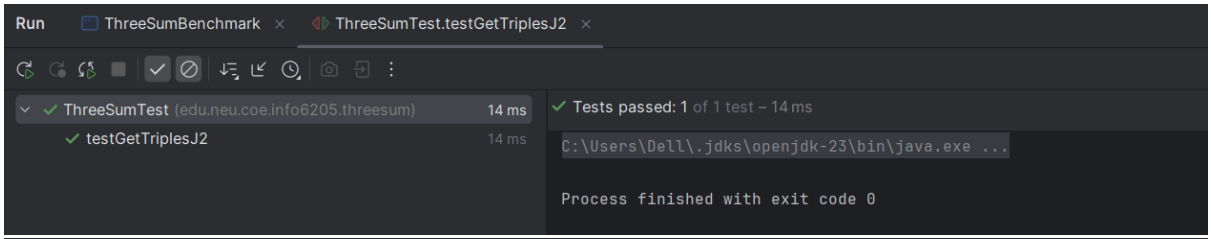

# Unit Test Screenshots:

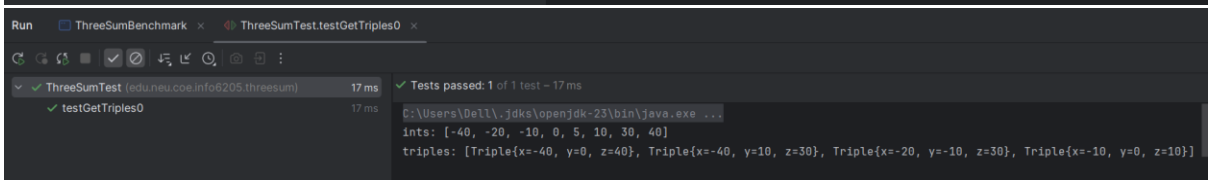



**Single unit test execution-**

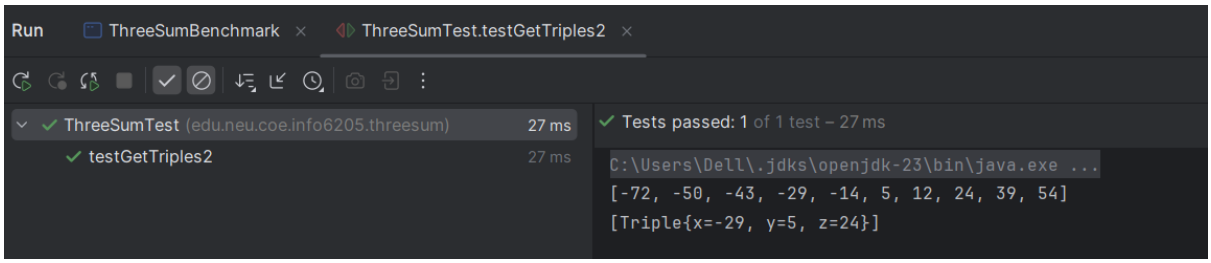


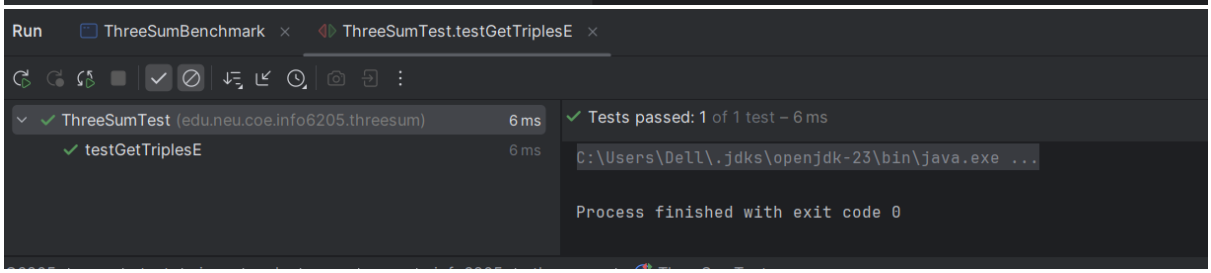




Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesJ2
ThreeSumTest (edu.neu.coe.info6205.threesum) 14 ms
testGetTriplesJ2 14 ms
Tests passed: 1 of 1 test – 14 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Process finished with exit code 0

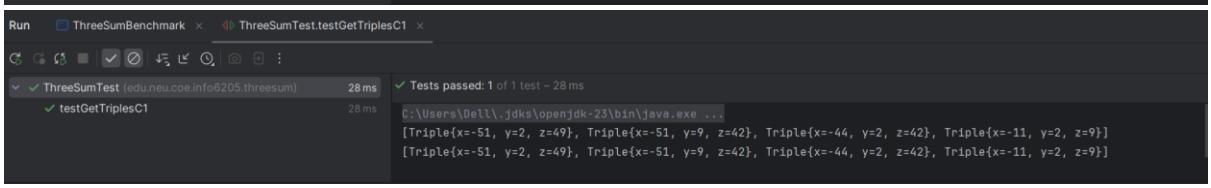

Run
ThreeSumBenchmark
ThreeSumTest.testGetTriples0
ThreeSumTest (edu.neu.coe.info6205.threesum) 17 ms
testGetTriples0 17 ms
Tests passed: 1 of 1 test – 17 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]

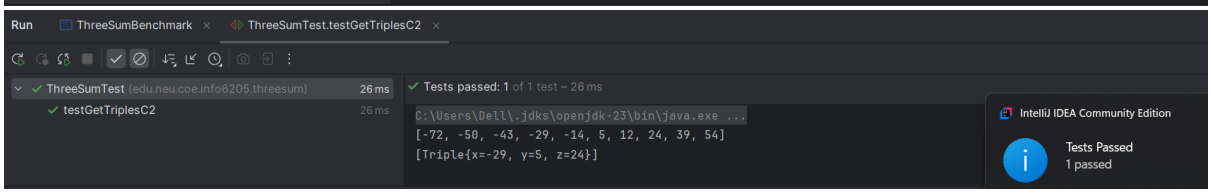

Run
ThreeSumBenchmark
ThreeSumTest.testGetTriples1
ThreeSumTest (edu.neu.coe.info6205.threesum) 23 ms
testGetTriples1 23 ms
Tests passed: 1 of 1 test – 23 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, 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}]

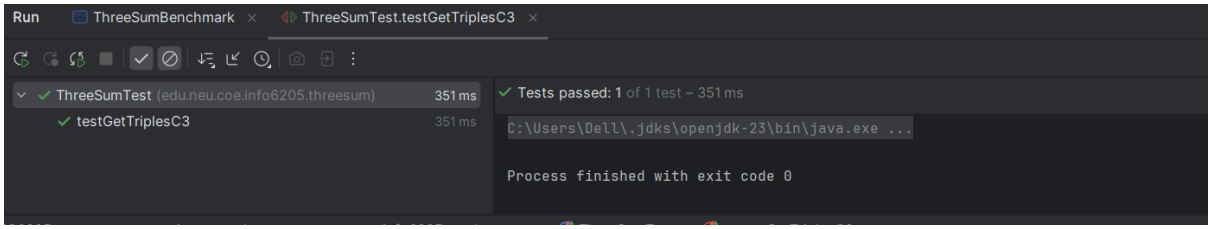

Run
ThreeSumBenchmark
ThreeSumTest.testGetTriples2
ThreeSumTest (edu.neu.coe.info6205.threesum) 27 ms
testGetTriples2 27 ms
Tests passed: 1 of 1 test – 27 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]

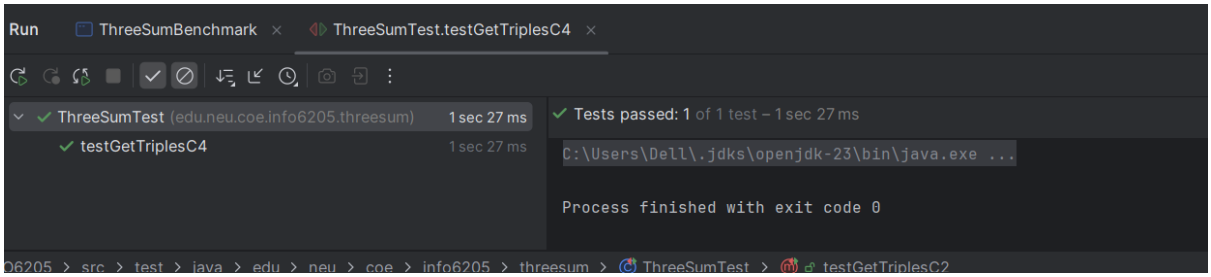

Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesE
ThreeSumTest (edu.neu.coe.info6205.threesum) 6 ms
testGetTriplesE 6 ms
Tests passed: 1 of 1 test – 6 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Process finished with exit code 0


Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesC0
ThreeSumTest (edu.neu.coe.info6205.threesum) 26 ms
testGetTriplesC0 26 ms
Tests passed: 1 of 1 test – 26 ms
C:\Users\Dell\.jdks\openjdk-23\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=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]


Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesC1
ThreeSumTest (edu.neu.coe.info6205.threesum) 28 ms
testGetTriplesC1 28 ms
Tests passed: 1 of 1 test – 28 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
[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}]

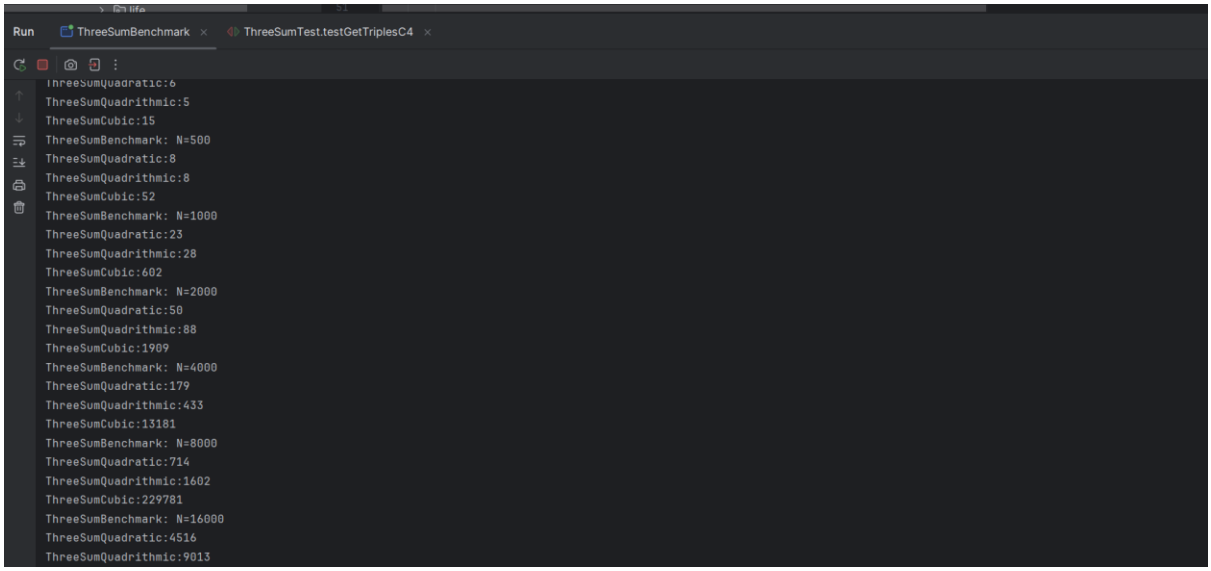

Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesC2
ThreeSumTest (edu.neu.coe.info6205.threesum) 26 ms
testGetTriplesC2 26 ms
Tests passed: 1 of 1 test – 26 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
IntelliJ IDEA Community Edition
Tests Passed
1 passed


Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesC3
ThreeSumTest (edu.neu.coe.info6205.threesum) 351 ms
testGetTriplesC3 351 ms
Tests passed: 1 of 1 test – 351 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Process finished with exit code 0


Run
ThreeSumBenchmark
ThreeSumTest.testGetTriplesC4
ThreeSumTest (edu.neu.coe.info6205.threesum) 1 sec 27 ms
testGetTriplesC4 1 sec 27 ms
Tests passed: 1 of 1 test – 1 sec 27 ms
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Process finished with exit code 0
O6205 › src › test › java › edu › neu › coe › info6205 › threesum › ThreeSumTest › testGetTriplesC2

## b. Evidence: (spreadsheet- comparison of Timings in execution, see output screenhot of benchmark class)

| No. of Int | Time for Quadratic(ns) | Time for Quadrithmic (ns) | Time for Cubic(ns) |
|---|---|---|---|
| 250 | 6 | 5 | 15 |
| 500 | 8 | 8 | 52 |
| 1000 | 23 | 28 | 602 |
| 2000 | 50 | 88 | 1909 |
| 4000 | 179 | 433 | 13181 |
| 8000 | 714 | 1602 | 229781 |
| 16000 | 4516 | 9013 | Taking lot of time in loading |




## c. Observations – Quadratic faster!

From the experiments performed, it can be concluded that the speed of the 4 algorithms for 3-SUM can be ranked as below(Fastest to Slowest)

1. Quadratic With Calipers $O(N^2)$
2. Quadratic $O(N^2)$
3. Quadrithmic $O(N^2 LogN)$
4. Cubic $O(N^3)$

From the data, we can see that the **Quadratic** approach consistently outperforms all the other methods are works faster than others.

This is because the array is sorted, and by using two pointers—one moving from the start and the other from the end—the chances of quickly finding pairs that add up to zero are higher.

**Cubic** is the slowest because it uses an extra loop to find triplets, compared to just two loops in the other methods.

As a result, **Cubic** takes much longer to run as the input size (N) increases.