Program Structures and Algorithms
Spring 2023(SEC - 8)

NAME: Dinesh Singh Shekhawat
NUID: 002776484

TASK:
Solve 3-SUM using the Quadrithmic, Quadratic, and quadraticWithCalipers approaches. Also showing the timing observations using the doubling method for at least five values of N, for each of the algorithms (including cubic).

RELATIONSHIP CONCLUSION:
The Quadratic method is a time and space efficient algorithm for the 3 Sum problem which utilizes sorting and two pointers technique. The array is sorted and two pointers are used to iterate through it, checking for the target sum. The movement of the pointers is determined by the comparison of the current sum with the target sum. This method eliminates the need for additional data structures.

The Quadratic method's efficiency in solving the 3 Sum problem is further reinforced by its ability to eliminate the need for checking all possible combinations of three numbers, resulting in significant time savings.

In addition to its efficiency, the Quadratic method is also a simple and intuitive algorithm, making it a suitable choice for solving the 3 Sum problem.

EVIDENCE TO SUPPORT THAT CONCLUSION

Source Code

ThreeSumQuadraticWithCalipers

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

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.function.Function;
/**
 * Implementation of ThreeSum which follows the approach of dividing 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).
 * <p>
 * The array provided in the constructor MUST be ordered.
 */
public class ThreeSumQuadraticWithCalipers implements ThreeSum {
    /**
     * Construct a ThreeSumQuadratic on a.
     * @param a a sorted array.
     */
    public ThreeSumQuadraticWithCalipers(int[] a) {
        this.a = a;
        length = a.length;
    }
    /**
     * Get an array or Triple containing all of those triples for which sum
is zero.
     * @return a Triple[].
     */
    public Triple[] getTriples() {
        List<Triple> triples = new ArrayList<>();
        Collections.sort(triples); // ???
        for (int i = 0; i < length - 2; i++)
            triples.addAll(calipers(a, i, Triple::sum));
```

```
        return triples.stream().distinct().toArray(Triple[]::new);
    }
    /**
     * Get a set of candidate Triples such that the first index is the
given value i.
     * Any candidate triple is added to the result if it yields zero when
passed into function
     * @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 a value
which will be compared with zero.
     * @return a List of Triples.
     */
    public static List<Triple> calipers(int[] a, int i, Function<Triple,
Integer> function) {
        List<Triple> triples = new ArrayList<>();
        if (i > 0 && a[i-1] == a[i+1]) {
            return triples;
        }
        int low = i + 1;
        int high = a.length - 1;
        while (low < high) {
            Triple triple = new Triple(a[i], a[low], a[high]);
            int sum = function.apply(triple);
            if (sum == 0) {
                  triples.add(triple);
                  low++;
                  while (a[low] == a[low - 1] && low < high) {
                        low++;
                  }
            } else if (sum < 0) {
                  low++;
            } else {
                  high--;
            }
        }

        return triples;
    }

    private final int[] a;
    private final int length;
}
```

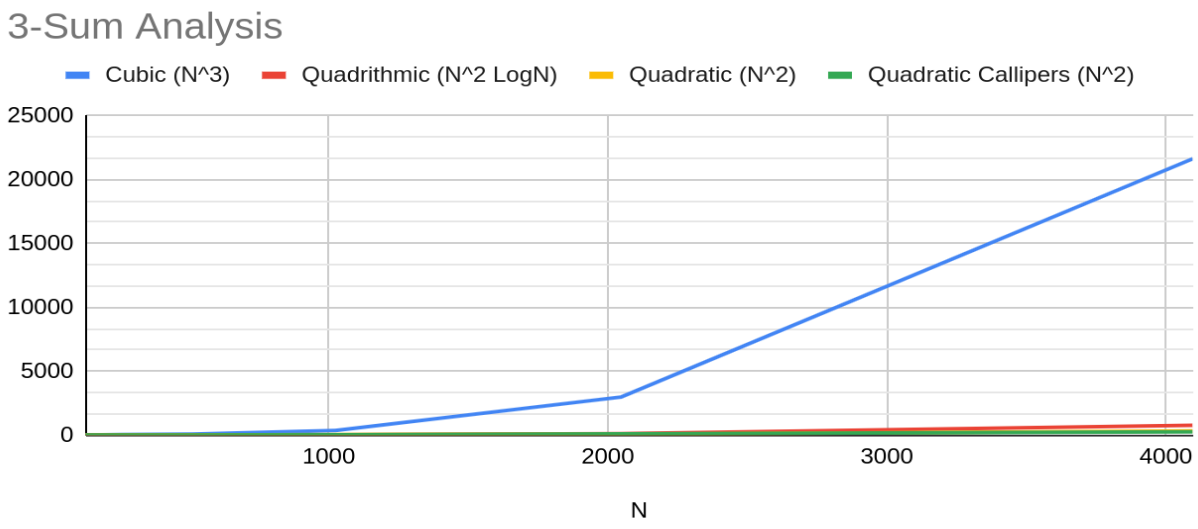
ThreeSumQuadratic

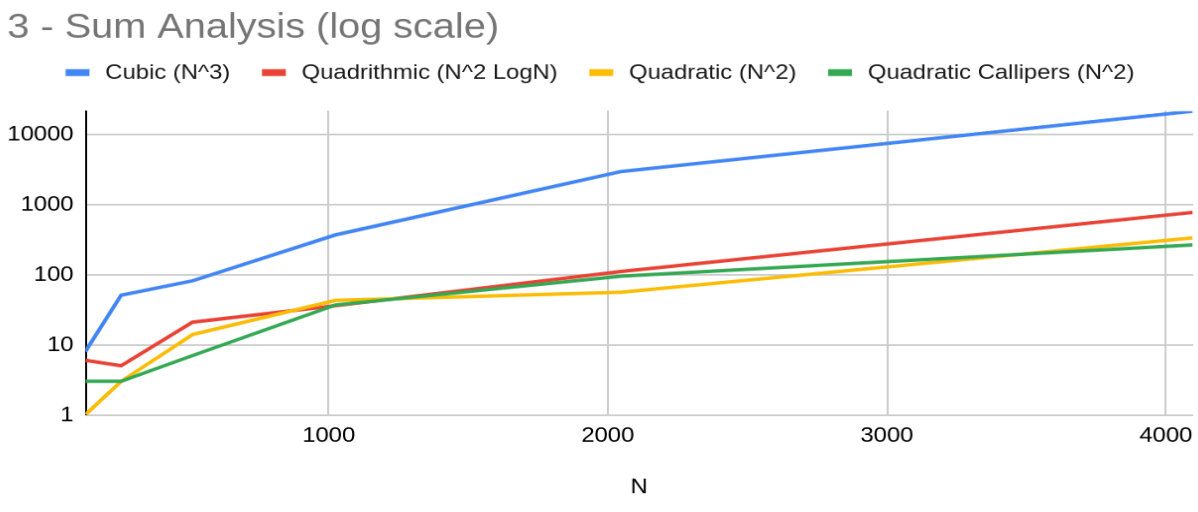
```
package edu.neu.coe.info6205.threesum;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Implementation of ThreeSum which follows the approach of dividing 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).
 * <p>
 * NOTE: The array provided in the constructor MUST be ordered.
 */
public class ThreeSumQuadratic implements ThreeSum {
    /**
     * Construct a ThreeSumQuadratic on a.
     * @param a a sorted array.
     */
    public ThreeSumQuadratic(int[] a) {
        this.a = a;
        length = a.length;
    }

    public Triple[] getTriples() {
        List<Triple> triples = new ArrayList<>();
        for (int i = 0; i < length; i++) triples.addAll(getTriples(i));
        Collections.sort(triples);
        return triples.stream().distinct().toArray(Triple[]::new);
    }

    /**
     * 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 such that
     */
    public List<Triple> getTriples(int j) {
        List<Triple> triples = new ArrayList<>();

        int low = 0;
        int high = a.length - 1;

        while (low < j && high > j) {
            int sum = a[low] + a[j] + a[high];

            if (sum == 0) {
                  Triple triple = new Triple(a[low], a[j], a[high]);
                  triples.add(triple);

                  high--;
                  low++;
            } else if (sum > 0) {
                  high--;
            } else {
                  low++;
            }
        }

        return triples;
    }

    private final int[] a;
    private final int length;
}
```

GRAPHICAL REPRESENTATION

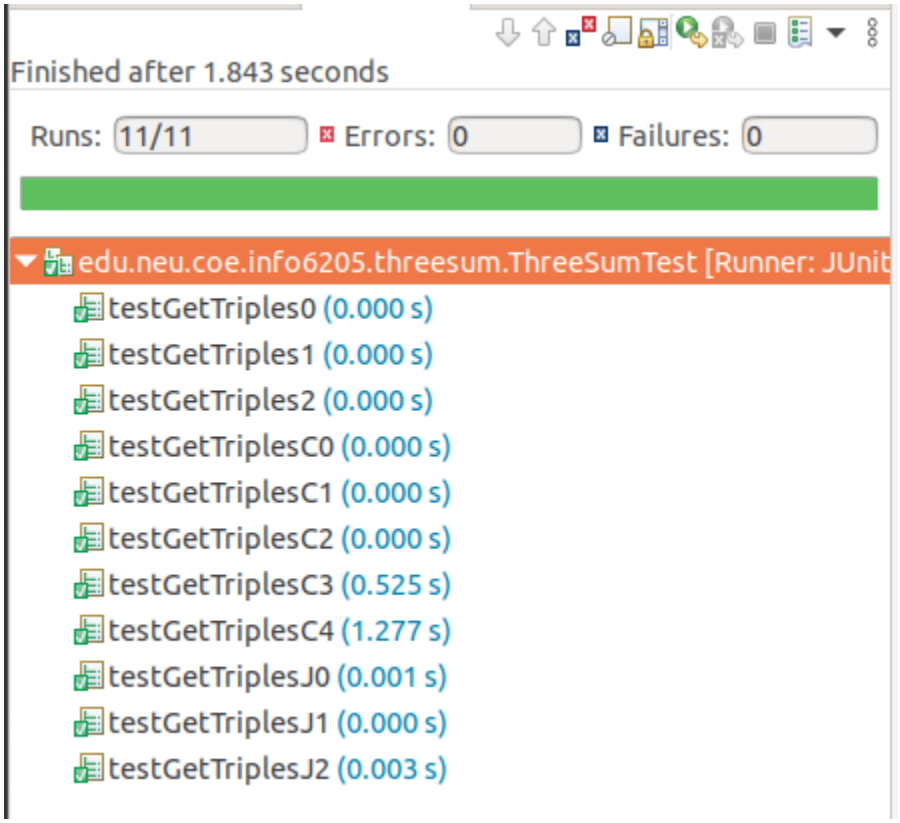
N	Time Taken (milliseconds)			
	Cubic (N^3)		Quadratic (N^2)	Quadratic Callipers (N^2)
128	8	6	1	3
256	51	5	3	3
512	82	21	14	7
1024	370	36	43	37
2048	2981	111	56	95
4096	21629	775	336	266





UNIT TEST SCREENSHOTS

Report



Code

```
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(ints);
        List<Triple> triples = target.getTriples(1);
        assertEquals(1, triples.size());
    }

    @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(3);
        assertEquals(2, triples.size());
    }

    @Test
    public void testGetTriplesJ2() {
        Supplier<int[]> intsSupplier = new Source(10, 15,
2L).intsSupplier(10);
        int[] ints = intsSupplier.get();
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples(5);
        assertEquals(1, triples.size());
    }

    @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(4, triples.length);
        assertEquals(4, new ThreeSumCubic(ints).getTriples().length);
    }

    @Test
    public void testGetTriples1() {
        Supplier<int[]> intsSupplier = new Source(20, 20,
1L).intsSupplier(10);
        int[] ints = intsSupplier.get();
```

```
        ThreeSum target = new ThreeSumQuadratic(ints);
        Triple[] triples = target.getTriples();
        assertEquals(4, triples.length);
        System.out.println(Arrays.toString(triples));
        Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
        System.out.println(Arrays.toString(triples2));
        assertEquals(4, triples2.length);
    }

    @Test
    public void testGetTriples2() {
        Supplier<int[]> intsSupplier = new Source(10, 15,
3L).intsSupplier(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(1, triples.length);
        assertEquals(1, new ThreeSumCubic(ints).getTriples().length);
    }

    @Ignore // Slow
    public void testGetTriples3() {
        Supplier<int[]> intsSupplier = new Source(1000,
1000).intsSupplier(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);
    }

    @Ignore // Slow
    public void testGetTriples4() {
        Supplier<int[]> intsSupplier = new Source(1500,
1000).intsSupplier(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);
    }

    @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(4, triples.length);
        assertEquals(4, new ThreeSumCubic(ints).getTriples().length);
    }

    @Test
    public void testGetTriplesC1() {
        Supplier<int[]> intsSupplier = new Source(20, 20,
1L).intsSupplier(10);
        int[] ints = intsSupplier.get();
        ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
        Triple[] triples = target.getTriples();
        assertEquals(4, triples.length);
        System.out.println(Arrays.toString(triples));
        Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
        System.out.println(Arrays.toString(triples2));
        assertEquals(4, triples2.length);
    }

    @Test
    public void testGetTriplesC2() {
        Supplier<int[]> intsSupplier = new Source(10, 15,
3L).intsSupplier(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(1, triples.length);
        assertEquals(1, new ThreeSumCubic(ints).getTriples().length);
    }
```

```
    @Test
    public void testGetTriplesC3() {
        Supplier<int[]> intsSupplier = new Source(1000,
1000).intsSupplier(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(1500,
1000).intsSupplier(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);
    }

}
```