

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

```
ThreeSumQuadraticWithCalipers.java × ThreeSumBenchmark.java ThreeSumQuadratic.java × ThreeSumTest.java
1 package edu.neu.coe.info6205.threesum;
2
3 > import ...
4
5
6
7 /**
8  * Implementation of ThreeSum which follows the approach of dividing the solution-space into
9  * N sub-spaces where each sub-space corresponds to a fixed value for the middle index of the three values.
10 * Each sub-space is then solved by expanding the scope of the other two indices outwards from the starting point.
11 * Since each sub-space can be solved in O(N) time, the overall complexity is O(N^2).
12 * <p>
13 * NOTE: The array provided in the constructor MUST be ordered.
14 */
15 public class ThreeSumQuadratic implements ThreeSum { 16 usages 1 xiaohuanlin +2
16     /**
17      * Construct a ThreeSumQuadratic on a.
18      *
19      * @param a a sorted array.
20      */
21     @ public ThreeSumQuadratic(int[] a) { 12 usages 1 xiaohuanlin
22         this.a = a;
23         length = a.length;
24     }
25
26     public Triple[] getTriples() { 28 usages 1 xiaohuanlin
27         List<Triple> triples = new ArrayList<>();
28         for (int i = 0; i < length; i++) triples.addAll(getTriples(i));
29         Collections.sort(triples);
30         return triples.stream().distinct().toArray(Triple[]::new);
31     }
32
33     /**
34      * Get a list of Triples such that the middle index is the given value j.
35
36      * @param j the index of the middle value.
37      * @return a Triple such that
38      */
39     public List<Triple> getTriples(int j) { 4 usages 1 Sanskruti03 +2
40         List<Triple> triples = new ArrayList<>();
41         // TO BE IMPLEMENTED : for each candidate, test if a[i] + a[j] + a[k] = 0.
42         int i=j-1;
43         int k =j+1;
44
45         while(i>=0 && k<=a.length-1){
46             if(a[i]+a[j]+a[k]==0){
47                 triples.add(new Triple(a[i],a[j],a[k]));
48                 i--;
49                 k++;
50             }else if (a[i]+a[j]+a[k]>0){
51                 i--;
52             }else if(a[i]+a[j]+a[k]<0){
53                 k++;
54             }
55         }return triples;
56         //throw new RuntimeException("implementation missing");
57     }
58
59     private final int[] a; 14 usages
60     private final int length; 2 usages
61 }
```

```
main > java > edu > neu > coe > info6205 > threesum > ThreeSumQuadratic > ThreeSumQuadratic
```

```
ThreeSumQuadraticWithCalipers.java ThreeSumBenchmark.java ThreeSumQuadratic.java × ThreeSumTest.java
32
33 /**
34  * Get a list of Triples such that the middle index is the given value j.
35  *
36  * @param j the index of the middle value.
37  * @return a Triple such that
38  */
39 public List<Triple> getTriples(int j) { 4 usages 1 Sanskruti03 +2
40     List<Triple> triples = new ArrayList<>();
41     // TO BE IMPLEMENTED : for each candidate, test if a[i] + a[j] + a[k] = 0.
42     int i=j-1;
43     int k =j+1;
44
45     while(i>=0 && k<=a.length-1){
46         if(a[i]+a[j]+a[k]==0){
47             triples.add(new Triple(a[i],a[j],a[k]));
48             i--;
49             k++;
50         }else if (a[i]+a[j]+a[k]>0){
51             i--;
52         }else if(a[i]+a[j]+a[k]<0){
53             k++;
54         }
55     }return triples;
56     //throw new RuntimeException("implementation missing");
57 }
58
59 private final int[] a; 14 usages
60 private final int length; 2 usages
61 }
```

## 2. QuadraticWithCalipers

```
ThreeSumQuadraticWithCalipers.java × ThreeSumBenchmark.java ThreeSumQuadratic.java ThreeSumTest.java
1 package edu.neu.coe.info6205.threesum;
2
3 > import ...
4
5 /**
6  * Implementation of ThreeSum which follows the approach of dividing the solution-space into
7  * N sub-spaces where each subspace corresponds to a fixed value for the middle index of the three values.
8  * Each subspace is then solved by expanding the scope of the other two indices outwards from the starting point.
9  * Since each subspace can be solved in O(N) time, the overall complexity is O(N^2).
10  * <p>
11  * The array provided in the constructor MUST be ordered.
12  */
13
14 public class ThreeSumQuadraticWithCalipers implements ThreeSum { 5 usages  ▲ xiaohuanlin +3
15     /**
16      * Construct ints ThreeSumQuadratic on ints.
17      *
18      * @param ints a sorted array.
19      */
20     public ThreeSumQuadraticWithCalipers(int[] ints) { 5 usages  ▲ Robin Hillyard
21         this.a = ints;
22         length = ints.length;
23     }
24
25     /**
26      * Get an array or Triple containing all of those triples for which sum is zero.
27      *
28      * @return a Triple[].
29      */
30     public Triple[] getTriples() { 28 usages  ▲ xiaohuanlin
31         List<Triple> triples = new ArrayList<>();
32         Collections.sort(triples); // ???
33         for (int i = 0; i < length - 2; i++)
34             ...
35     }
```

```
ThreeSumQuadraticWithCalipers.java × ThreeSumBenchmark.java ThreeSumQuadratic.java ThreeSumTest.java
16 public class ThreeSumQuadraticWithCalipers implements ThreeSum { 5 usages  ▲ xiaohuanlin +3
17     public Triple[] getTriples() { 28 usages  ▲ xiaohuanlin
18         List<Triple> triples = new ArrayList<>();
19         Collections.sort(triples); // ???
20         for (int i = 0; i < length - 2; i++)
21             triples.addAll(calipers(a, i, Triple::sum));
22         return triples.stream().distinct().toArray(Triple[]::new);
23     }
24
25     /**
26      * Get a set of candidate Triples such that the first index is the given value i.
27      * Any candidate triple is added to the result if it yields zero when passed into function.
28      *
29      * @param a a sorted array of ints. This method is concerned only with the partition of a starting with index i+1.
30      * @param i the index of the first element of resulting triples.
31      * @param function a function which takes a triple and returns a value which will be compared with zero.
32      * @return a List of Triples.
33      */
34     public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) { ▲ Sanskruti03 +2
35         List<Triple> triples = new ArrayList<>();
36         // TO BE IMPLEMENTED : use function to qualify triples and to navigate otherwise.
37         int j=i+1;
38         int k=a.length-1;
39
40         while(j<k){
41             int sum= a[i]+a[j]+a[k];
42             if (sum==0){
43                 triples.add(new Triple(a[i],a[j],a[k]));
44                 j++;
45                 k--;
46             }
47             //Checking duplicates
48         }
49     }
```

```
ThreeSumQuadraticWithCalipers.java × ThreeSumBenchmark.java ThreeSumQuadratic.java ThreeSumTest.java
16 public class ThreeSumQuadraticWithCalipers implements ThreeSum { 5 usages ± xiaohuanlin +3
49 public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) { ± Sanskrutii03 +2
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
63             while(j<k && a[j]==a[j-1]){
64                 j++;
65             }
66             while(j<k && a[k]==a[k+1]){
67                 k--;
68             }
69
70             }else if(sum>0){
71                 k--;
72             }else j++;
73         }
74
75
76         return triples;
77
78         // END SOLUTION
79     }
80
81     private final int[] a; 2 usages
82     private final int length; 2 usages
83 }
```

main > java > edu > neu > coe > info6205 > threesum > ThreeSumQuadraticWithCalipers > calipers

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

```
© ThreeSumQuadraticWithCalipers.java  © ThreeSumBenchmark.java ×  © ThreeSumQuadratic.java  © ThreeSumTest.java
1  package edu.neu.coe.info6205.threesum;
2
3  > import ...
11
12 ▶ public class ThreeSumBenchmark {  ± xiaohuanlin +2
13      public ThreeSumBenchmark(int runs, int n, int m) {  7 usages  ± xiaohuanlin
14          this.runs = runs;
15          this.supplier = new Source(n, m).intsSupplier( safetyFactor: 10);
16          this.n = n;
17      }
18
19      public void runBenchmarks() {  7 usages  ± xiaohuanlin
20          System.out.println("ThreeSumBenchmark: N=" + n);
21          benchmarkThreeSum( description: "ThreeSumQuadratic", (xs) -> new ThreeSumQuadratic(xs).getTriples(), n, timeLogg
22          benchmarkThreeSum( description: "ThreeSumQuadrithmic", (xs) -> new ThreeSumQuadrithmic(xs).getTriples(), n, time
23          benchmarkThreeSum( description: "ThreeSumCubic", (xs) -> new ThreeSumCubic(xs).getTriples(), n, timeLoggersCubic
24      }
25
26 ▶ public static void main(String[] args) {  ± xiaohuanlin
27     new ThreeSumBenchmark( runs: 100, n: 250, m: 250).runBenchmarks();
28     new ThreeSumBenchmark( runs: 50, n: 500, m: 500).runBenchmarks();
29     new ThreeSumBenchmark( runs: 20, n: 1000, m: 1000).runBenchmarks();
30     new ThreeSumBenchmark( runs: 10, n: 2000, m: 2000).runBenchmarks();
31     new ThreeSumBenchmark( runs: 5, n: 4000, m: 4000).runBenchmarks();
32     new ThreeSumBenchmark( runs: 3, n: 8000, m: 8000).runBenchmarks();
33     new ThreeSumBenchmark( runs: 2, n: 16000, m: 16000).runBenchmarks();
34 }
35
36 @ private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLogger
37     //if (description.equals("ThreeSumCubic") && n > 4000) return;
38     // TO BE IMPLEMENTED
39 }
```

```
12 public class ThreeSumBenchmark {  ± xiaohuanlin +2
36 @ private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLd
37     //if (description.equals("ThreeSumCubic") && n > 4000) return;
38     // TO BE IMPLEMENTED
39
40     int[] ints= supplier.get();
41     if(description.equals("ThreeSumQuadratic")){
42         Stopwatch stopwatch=new Stopwatch();
43         new ThreeSumQuadratic(ints).getTriples();
44
45         System.out.println(description + ":" + stopwatch.lap());
46     }
47     else if (description.equals("ThreeSumQuadrithmic")){
48         Stopwatch stopwatch=new Stopwatch();
49         new ThreeSumQuadrithmic(ints).getTriples();
50         System.out.println(description + ":" + stopwatch.lap());
51     }
52 }
53 else if (description.equals("ThreeSumQuadraticWithCalipers")){
54     Stopwatch stopwatch=new Stopwatch();
55     new ThreeSumQuadraticWithCalipers(ints).getTriples();
56     System.out.println(description + ":" + stopwatch.lap());
57 }
58 }else if (description.equals("ThreeSumCubic")){
59     Stopwatch stopwatch=new Stopwatch();
60     new ThreeSumCubic(ints).getTriples();
61     System.out.println(description + ":" + stopwatch.lap());
62 }
63
64
65 }
```

```

ThreeSumQuadraticWithCalipers.java  ThreeSumBenchmark.java  ThreeSumQuadratic.java  ThreeSumTest.java
12  public class ThreeSumBenchmark {  xiaohuanlin +2
36      private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final Time
65
66
67
68
69      //throw new RuntimeException("implementation missing");
70      }
71
72      private final static TimeLogger[] timeLoggersCubic = { 1 usage
73          new TimeLogger( prefix: "Raw time per run (mSec): ", minimumComparisons: null),
74          new TimeLogger( prefix: "Normalized time per run (n^3): ", n -> 1.0 / 6 * n * n * n)
75      };
76      private final static TimeLogger[] timeLoggersQuadrithmic = { 1 usage
77          new TimeLogger( prefix: "Raw time per run (mSec): ", minimumComparisons: null),
78          new TimeLogger( prefix: "Normalized time per run (n^2 log n): ", n -> n * n * Utilities.lg(n))
79      };
80      private final static TimeLogger[] timeLoggersQuadratic = { 1 usage
81          new TimeLogger( prefix: "Raw time per run (mSec): ", minimumComparisons: null),
82          new TimeLogger( prefix: "Normalized time per run (n^2): ", n -> 1.0 / 2 * n * n)
83      };
84
85      private final int runs; 1 usage
86      private final Supplier<int[]> supplier; 2 usages
87      private final int n; 5 usages
88  }

```

## Output-

ThreeSumQuadratic and ThreeSumQuadraticWithCaliper (Quadrithmic already implemented)

```

Run  ThreeSumTest  x ThreeSumTest.testGetTriplesC1  x
C:\Users\Bell\jdk\openjdk-23\bin\java.exe ...
Tests passed: 12 of 12 tests - 1 sec 444 ms
testGetTriples0 18 ms
testGetTriples1 9 ms
testGetTriples2 1 ms
testGetTriplesE 0 ms
testGetTriplesC0 1 ms
testGetTriplesC1 17 ms
testGetTriplesC2 1 ms
testGetTriplesC3 351 ms
testGetTriplesC4 1 sec 46 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 0 ms
testGetTriplesJ2 0 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), 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

```

## Benchmark class for all 4-method comparison-

ThreeSumQuadratic and ThreeSumQuadraticWithCaliper, Quadrithmic and Cubic

```
Run  ThreeSumBenchmark x ThreeSumTest.testGetTriplesC1 x

C:\Users\De11\.jdk\openjdk-23\bin\java.exe ...
ThreeSumBenchmark: N=250
ThreeSumQuadratic:7
ThreeSumQuadrnithmic:4
ThreeSumCubic:16
ThreeSumBenchmark: N=500
ThreeSumQuadratic:6
ThreeSumQuadrnithmic:7
ThreeSumCubic:64
ThreeSumBenchmark: N=1000
ThreeSumQuadratic:22
ThreeSumQuadrnithmic:27
ThreeSumCubic:652
ThreeSumBenchmark: N=2000
ThreeSumQuadratic:48
ThreeSumQuadrnithmic:89
ThreeSumCubic:1678
ThreeSumBenchmark: N=4000
ThreeSumQuadratic:179
ThreeSumQuadrnithmic:395
```

## Unit Test Screenshots:

```
Run  ThreeSumTest x ThreeSumTest.testGetTriplesC1 x

ThreeSumTest (edu.neu.coe.info6205.threesum) 1 sec 444 ms Tests passed: 12 of 12 tests - 1 sec 444 ms
testGetTriples0 18 ms
testGetTriples1 9 ms
testGetTriples2 1 ms
testGetTriplesE 0 ms
testGetTriplesC0 1 ms
testGetTriplesC1 17 ms
testGetTriplesC2 1 ms
testGetTriplesC3 351 ms
testGetTriplesC4 1 sec 46 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 0 ms
testGetTriplesJ2 0 ms

C:\Users\De11\.jdk\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}]
[Triples{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}]
[Triples{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]
[Triples{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}]
[Triples{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}]
[Triples{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]
[Triples{x=-29, y=5, z=24}]

Process finished with exit code 0
```

## Single unit test execution-

```
Run  ThreeSumBenchmark x ThreeSumTest.testGetTriplesJ0 x

ThreeSumTest (edu.neu.coe.info6205.threesum) 7 ms Tests passed: 1 of 1 test - 7 ms
testGetTriplesJ0 7 ms
C:\Users\De11\.jdk\openjdk-23\bin\java.exe ...

Process finished with exit code 0

Run  ThreeSumBenchmark x ThreeSumTest.testGetTriplesJ1 x

ThreeSumTest (edu.neu.coe.info6205.threesum) 9 ms Tests passed: 1 of 1 test - 9 ms
testGetTriplesJ1 9 ms
C:\Users\De11\.jdk\openjdk-23\bin\java.exe ...

Process finished with exit code 0
```

Run

ThreeSumBenchmark x ThreeSumTest.testGetTriplesJ2 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 14 ms

✓ testGetTriplesJ2 14 ms

✓ Tests passed: 1 of 1 test – 14 ms

C:\Users\De11\jdk\openjdk-23\bin\java.exe ...

Process finished with exit code 0

Run

ThreeSumBenchmark x ThreeSumTest.testGetTriples0 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 17 ms

✓ testGetTriples0 17 ms

✓ Tests passed: 1 of 1 test – 17 ms

C:\Users\De11\jdk\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 x ThreeSumTest.testGetTriples1 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 23 ms

✓ testGetTriples1 23 ms

✓ Tests passed: 1 of 1 test – 23 ms

C:\Users\De11\jdk\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 x ThreeSumTest.testGetTriples2 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 27 ms

✓ testGetTriples2 27 ms

✓ Tests passed: 1 of 1 test – 27 ms

C:\Users\De11\jdk\openjdk-23\bin\java.exe ...

[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]

[Triple{x=-29, y=5, z=24}]

Run

ThreeSumBenchmark x ThreeSumTest.testGetTriplesE x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 6 ms

✓ testGetTriplesE 6 ms

✓ Tests passed: 1 of 1 test – 6 ms

C:\Users\De11\jdk\openjdk-23\bin\java.exe ...

Process finished with exit code 0

Run

ThreeSumBenchmark x ThreeSumTest.testGetTriplesC0 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 26 ms

✓ testGetTriplesC0 26 ms

✓ Tests passed: 1 of 1 test – 26 ms

C:\Users\De11\jdk\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 x ThreeSumTest.testGetTriplesC1 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 28 ms

✓ testGetTriplesC1 28 ms

✓ Tests passed: 1 of 1 test – 28 ms

C:\Users\De11\jdk\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 x ThreeSumTest.testGetTriplesC2 x

✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 26 ms

✓ testGetTriplesC2 26 ms

✓ Tests passed: 1 of 1 test – 26 ms

C:\Users\De11\jdk\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 x ThreeSumTest.testGetTriplesC3 x
[Icons]
✓ ThreeSumTest (edu.neu.coe.info6205.threesum) 351 ms
  ✓ testGetTriplesC3 351 ms
  ✓ Tests passed: 1 of 1 test - 351 ms
  C:\Users\DeLL\.jdk\openjdk-23\bin\java.exe ...
  Process finished with exit code 0
```

```
Run  ThreeSumBenchmark x ThreeSumTest.testGetTriplesC4 x
[Icons]
✓ 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\.jdk\openjdk-23\bin\java.exe ...
  Process finished with exit code 0
06205 > src > test > java > edu > neu > coe > info6205 > threesum > ThreeSumTest > testGetTriplesC2
```

**b. Evidence:** (spreadsheet- comparison of Timings in execution, see output screenshot 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

```

Run ThreeSumBenchmark x ThreeSumTest.testGetTriplesC4 x
ThreeSumQuadratic:6
ThreeSumQuadrithmic:5
ThreeSumCubic:15
ThreeSumBenchmark: N=500
ThreeSumQuadratic:8
ThreeSumQuadrithmic:8
ThreeSumCubic:52
ThreeSumBenchmark: N=1000
ThreeSumQuadratic:23
ThreeSumQuadrithmic:28
ThreeSumCubic:602
ThreeSumBenchmark: N=2000
ThreeSumQuadratic:50
ThreeSumQuadrithmic:88
ThreeSumCubic:1909
ThreeSumBenchmark: N=4000
ThreeSumQuadratic:179
ThreeSumQuadrithmic:433
ThreeSumCubic:13181
ThreeSumBenchmark: N=8000
ThreeSumQuadratic:714
ThreeSumQuadrithmic:1602
ThreeSumCubic:229781
ThreeSumBenchmark: N=16000
ThreeSumQuadratic:4516
ThreeSumQuadrithmic:9013
  
```

**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  $O(N^2 \log N)$
4. Cubic  $O(N^3)$

From the data, we can see that the **Quadratic** approach consistently outperforms all the other methods and 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.

