

## Program Structures and Algorithms

### Spring 2023(SEC –1)

**NAME: Foram Kamani**

**NUID: 002732551**

### Task:

Solve 3-SUM using the Quadrithmic, Quadratic, and quadraticWithCalipers approaches.

### Relationship Conclusion:

After executing the main method in the ThreeSumBenchmark.java class multiple times, I conclude that the ThreeSumQuadratic with  $O(N^2)$  is the best way to resolve the problem of ThreeSums as it consumes significantly less time than the other three approaches.

Following are the Time Complexities:

ThreeSumQuadratic:  $O(N^2)$

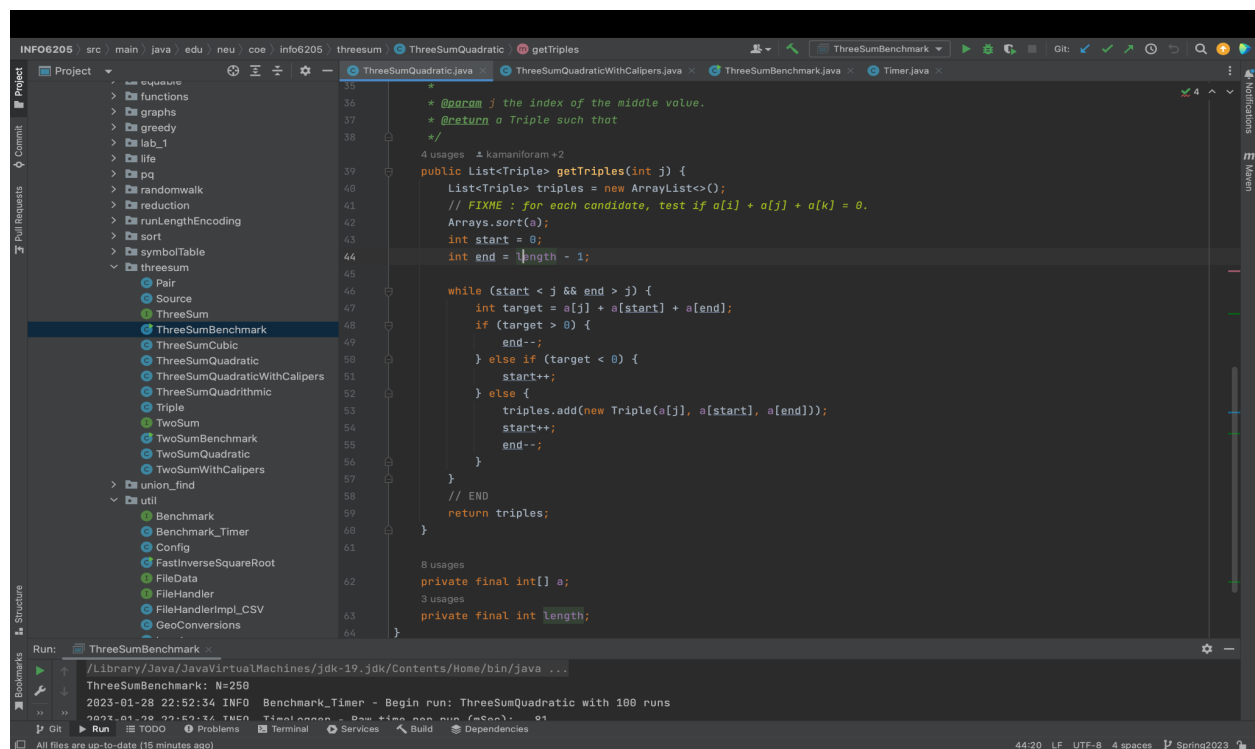
ThreeSumCubic:  $O(N^3)$

ThreeSumQuadrathmic:  $O(N^2 \log N)$

ThreeSumQuadraticWithCalipers:  $O(N^2)$

### Code Changes:

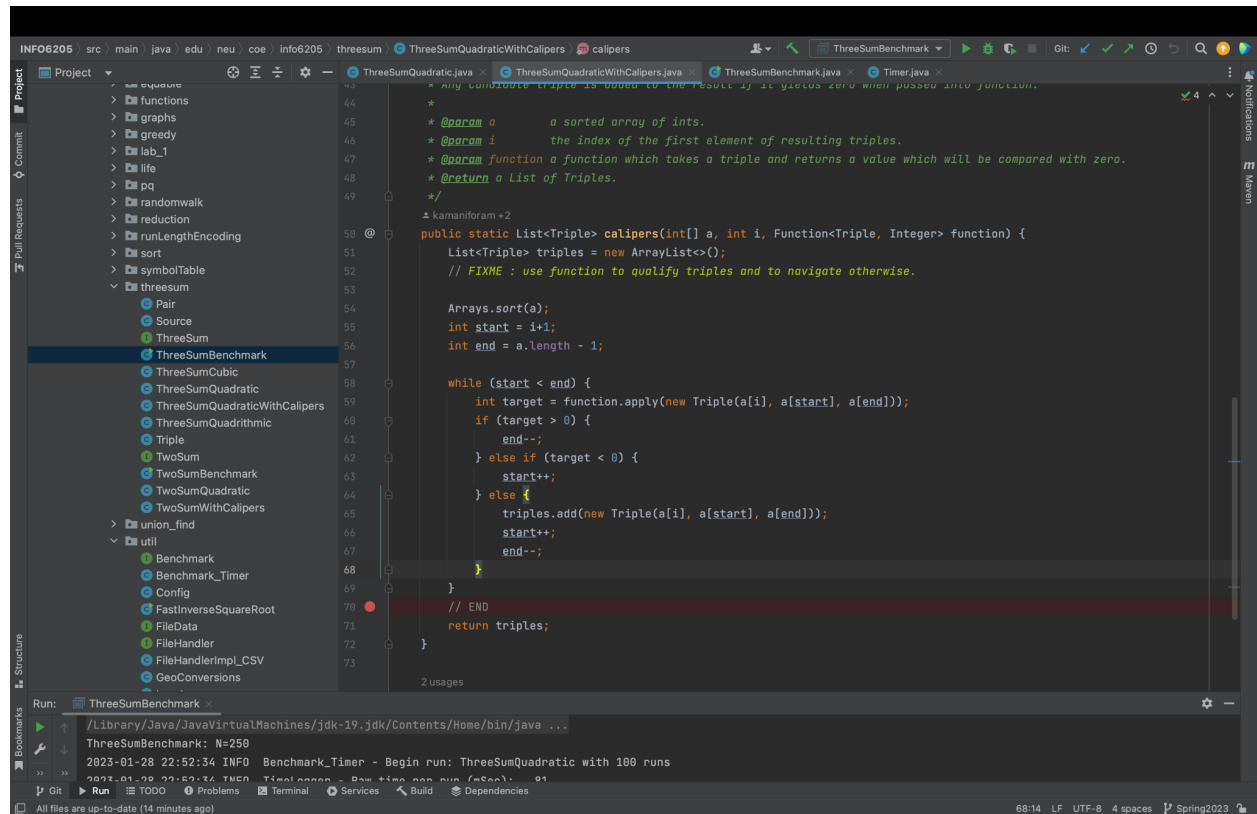
#### ThreeSumQuadratic:



```
35
36 * @param j the index of the middle value.
37 * @return a Triple such that
38 */
39 4 usages kamaniforam +2
40 public List<Triple> getTriples(int j) {
41     List<Triple> triples = new ArrayList<>();
42     // FIXME : for each candidate, test if a[i] + a[j] + a[k] = 0.
43     Arrays.sort(a);
44     int start = 0;
45     int end = i.length - 1;
46
47     while (start < j && end > j) {
48         int target = a[j] + a[start] + a[end];
49         if (target > 0) {
50             end--;
51         } else if (target < 0) {
52             start++;
53         } else {
54             triples.add(new Triple(a[j], a[start], a[end]));
55             start++;
56             end--;
57         }
58     }
59     // END
60     return triples;
61 }
62 8 usages
63 private final int[] a;
64 3 usages
65 private final int length;
```

Run: ThreeSumBenchmark x  
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...  
ThreeSumBenchmark: N=250  
2023-01-28 22:52:34 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 100 runs

## ThreeSumQuadraticWithCalipers:



```
44  * Any candidate triple is added to the result if it yields zero when passed into junction.
45  *
46  * @param a      a sorted array of ints.
47  * @param i      the index of the first element of resulting triples.
48  * @param function a function which takes a triple and returns a value which will be compared with zero.
49  * @return a List of Triples.
50  */
51  public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) {
52      List<Triple> triples = new ArrayList<>();
53      // FIXME : use function to qualify triples and to navigate otherwise.
54
55      Arrays.sort(a);
56      int start = i+1;
57      int end = a.length - 1;
58
59      while (start < end) {
60          int target = function.apply(new Triple(a[i], a[start], a[end]));
61          if (target > 0) {
62              end--;
63          } else if (target < 0) {
64              start++;
65          } else {
66              triples.add(new Triple(a[i], a[start], a[end]));
67              start++;
68              end--;
69          }
70      }
71      // END
72      return triples;
73  }
```

Run: ThreeSumBenchmark ...  
ThreeSumBenchmark: N=250  
2023-01-28 22:52:34 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 100 runs  
2023-01-28 22:52:34 INFO TimeLogger - Raw time per run (mSec): .81  
2023-01-28 22:52:34 INFO TimeLogger - Normalized time per run (n^2): 12.96  
2023-01-28 22:52:34 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 100 runs  
2023-01-28 22:52:35 INFO TimeLogger - Raw time per run (mSec): 2.00  
2023-01-28 22:52:35 INFO TimeLogger - Normalized time per run (n^2 log n): 4.02  
2023-01-28 22:52:35 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 100 runs  
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 7.86  
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run (n^3): .50

## Evidence to support that conclusion:

I have tested the algorithms against various values of N, and you can see that for N=250 the raw time per run for **ThreeSumQuadratic** is **.81**, **ThreeSumQuadrithmic** is **2.00**, and for **ThreeSumCubic** is **7.86**. Initially, the difference is negligible but as the value of N increases the value increases exponentially. And hence **ThreeSumQuadratic** method works the best.

## ThreeSumBenchmark: N=250

2023-01-28 22:52:34 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 100 runs  
2023-01-28 22:52:34 INFO TimeLogger - Raw time per run (mSec): .81  
2023-01-28 22:52:34 INFO TimeLogger - Normalized time per run (n^2): 12.96  
2023-01-28 22:52:34 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 100 runs  
2023-01-28 22:52:35 INFO TimeLogger - Raw time per run (mSec): 2.00  
2023-01-28 22:52:35 INFO TimeLogger - Normalized time per run (n^2 log n): 4.02  
2023-01-28 22:52:35 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 100 runs  
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 7.86  
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run (n^3): .50

### **ThreeSumBenchmark: N=500**

2023-01-28 22:52:36 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 50 runs  
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 1.70  
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run ( $n^2$ ): 6.80  
2023-01-28 22:52:36 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 50 runs  
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 7.14  
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 3.19  
2023-01-28 22:52:36 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 50 runs  
2023-01-28 22:52:39 INFO TimeLogger - Raw time per run (mSec): 50.50  
2023-01-28 22:52:39 INFO TimeLogger - Normalized time per run ( $n^3$ ): .40

### **ThreeSumBenchmark: N=1000**

2023-01-28 22:52:39 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 20 runs  
2023-01-28 22:52:39 INFO TimeLogger - Raw time per run (mSec): 3.85  
2023-01-28 22:52:39 INFO TimeLogger - Normalized time per run ( $n^2$ ): 3.85  
2023-01-28 22:52:39 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 20 runs  
2023-01-28 22:52:40 INFO TimeLogger - Raw time per run (mSec): 16.85  
2023-01-28 22:52:40 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 1.69  
2023-01-28 22:52:40 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 20 runs  
2023-01-28 22:52:46 INFO TimeLogger - Raw time per run (mSec): 273.00  
2023-01-28 22:52:46 INFO TimeLogger - Normalized time per run ( $n^3$ ): .27

### **ThreeSumBenchmark: N=2000**

2023-01-28 22:52:46 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 10 runs  
2023-01-28 22:52:46 INFO TimeLogger - Raw time per run (mSec): 15.10  
2023-01-28 22:52:46 INFO TimeLogger - Normalized time per run ( $n^2$ ): 3.77  
2023-01-28 22:52:46 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 10 runs  
2023-01-28 22:52:47 INFO TimeLogger - Raw time per run (mSec): 90.90  
2023-01-28 22:52:47 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 2.07  
2023-01-28 22:52:47 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 10 runs  
2023-01-28 22:53:20 INFO TimeLogger - Raw time per run (mSec): 2827.00  
2023-01-28 22:53:20 INFO TimeLogger - Normalized time per run ( $n^3$ ): .35

### **ThreeSumBenchmark: N=4000**

2023-01-28 22:53:20 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 5 runs  
2023-01-28 22:53:21 INFO TimeLogger - Raw time per run (mSec): 90.20  
2023-01-28 22:53:21 INFO TimeLogger - Normalized time per run ( $n^2$ ): 5.64  
2023-01-28 22:53:21 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 5 runs  
2023-01-28 22:53:24 INFO TimeLogger - Raw time per run (mSec): 464.20

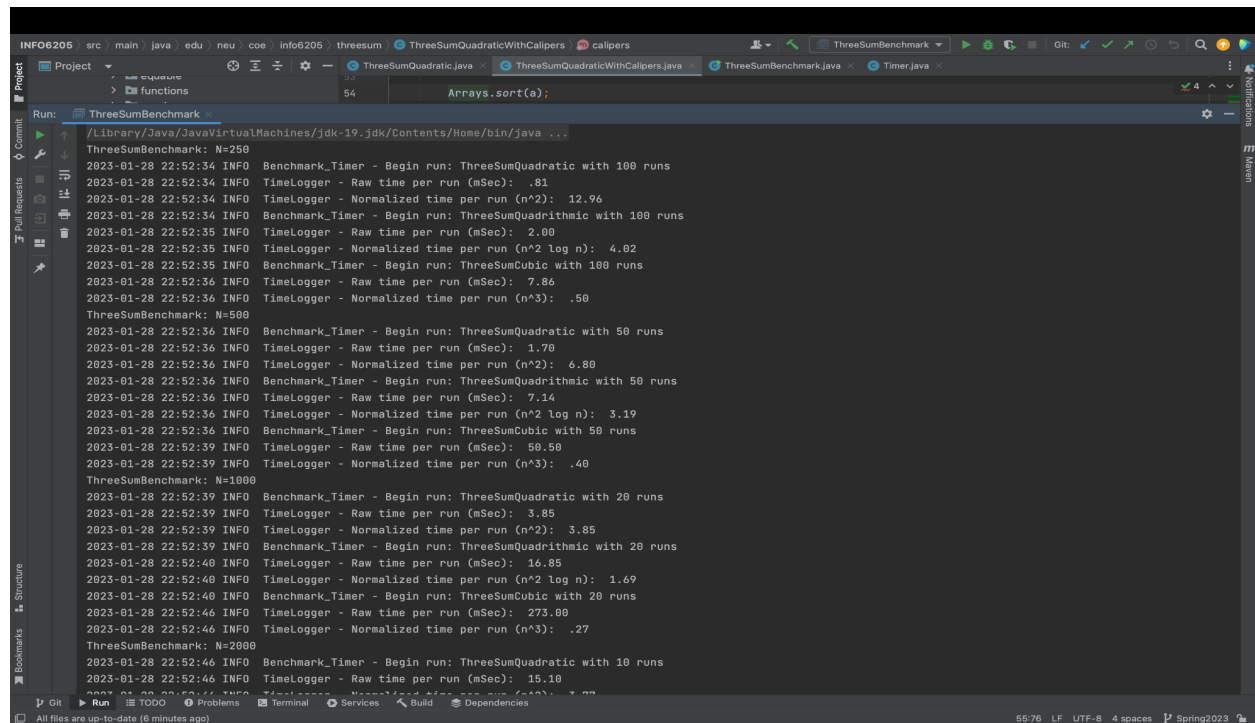
2023-01-28 22:53:24 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 2.42  
2023-01-28 22:53:24 INFO Benchmark\_Timer - Begin run: ThreeSumCubic with 5 runs  
2023-01-28 22:55:29 INFO TimeLogger - Raw time per run (mSec): 17893.00  
2023-01-28 22:55:29 INFO TimeLogger - Normalized time per run ( $n^3$ ): .28

### **ThreeSumBenchmark: N=8000**

2023-01-28 22:55:29 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 3 runs  
2023-01-28 22:55:32 INFO TimeLogger - Raw time per run (mSec): 556.00  
2023-01-28 22:55:32 INFO TimeLogger - Normalized time per run ( $n^2$ ): 8.69  
2023-01-28 22:55:32 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 3 runs  
2023-01-28 22:55:43 INFO TimeLogger - Raw time per run (mSec): 2128.33  
2023-01-28 22:55:43 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 2.56

### **ThreeSumBenchmark: N=16000**

2023-01-28 22:55:43 INFO Benchmark\_Timer - Begin run: ThreeSumQuadratic with 2 runs  
2023-01-28 22:55:55 INFO TimeLogger - Raw time per run (mSec): 2939.00  
2023-01-28 22:55:55 INFO TimeLogger - Normalized time per run ( $n^2$ ): 11.48  
2023-01-28 22:55:55 INFO Benchmark\_Timer - Begin run: ThreeSumQuadrithmic with 2 runs  
2023-01-28 22:56:36 INFO TimeLogger - Raw time per run (mSec): 10934.50  
2023-01-28 22:56:36 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 3.06



```
INFO6205 /src/main/java/edu/neu/coe/info6205/threesum/ThreeSumQuadraticWithCalipers calipers ThreeSumBenchmark ThreeSumBenchmark.java Timer.java
Project functions 54 Arrays.sort(a);
Run: ThreeSumBenchmark
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
ThreeSumBenchmark: N=250
2023-01-28 22:52:34 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 100 runs
2023-01-28 22:52:34 INFO TimeLogger - Raw time per run (mSec): .81
2023-01-28 22:52:34 INFO TimeLogger - Normalized time per run ( $n^2$ ): 12.96
2023-01-28 22:52:34 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 100 runs
2023-01-28 22:52:35 INFO TimeLogger - Raw time per run (mSec): 2.00
2023-01-28 22:52:35 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 4.02
2023-01-28 22:52:35 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 100 runs
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 7.86
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run ( $n^3$ ): .50
ThreeSumBenchmark: N=500
2023-01-28 22:52:36 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 50 runs
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 1.70
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run ( $n^2$ ): 6.80
2023-01-28 22:52:36 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 50 runs
2023-01-28 22:52:36 INFO TimeLogger - Raw time per run (mSec): 7.14
2023-01-28 22:52:36 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 3.19
2023-01-28 22:52:36 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 22:52:39 INFO TimeLogger - Raw time per run (mSec): 50.50
2023-01-28 22:52:39 INFO TimeLogger - Normalized time per run ( $n^3$ ): .40
ThreeSumBenchmark: N=1000
2023-01-28 22:52:39 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 20 runs
2023-01-28 22:52:39 INFO TimeLogger - Raw time per run (mSec): 3.85
2023-01-28 22:52:39 INFO TimeLogger - Normalized time per run ( $n^2$ ): 3.85
2023-01-28 22:52:39 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 20 runs
2023-01-28 22:52:40 INFO TimeLogger - Raw time per run (mSec): 16.85
2023-01-28 22:52:40 INFO TimeLogger - Normalized time per run ( $n^2 \log n$ ): 1.69
2023-01-28 22:52:40 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 20 runs
2023-01-28 22:52:46 INFO TimeLogger - Raw time per run (mSec): 273.00
2023-01-28 22:52:46 INFO TimeLogger - Normalized time per run ( $n^3$ ): .27
ThreeSumBenchmark: N=2000
2023-01-28 22:52:46 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 10 runs
2023-01-28 22:52:46 INFO TimeLogger - Raw time per run (mSec): 15.10
2023-01-28 22:52:46 INFO TimeLogger - Normalized time per run ( $n^2$ ): 1.27
```

The screenshot shows an IDE with the `ThreeSumBenchmark` class selected. The console output displays benchmark results for three different algorithms: `ThreeSumQuadratic`, `ThreeSumQuadrithmic`, and `ThreeSumCubic`. The results are grouped by the number of runs (10, 5, and 3) and the number of elements (N=2000, N=4000, N=8000, and N=16000). Each group shows the raw time per run in milliseconds and the normalized time per run in seconds.

```
Run: ThreeSumBenchmark
ThreeSumBenchmark: N=2000
2023-01-28 22:52:46 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 10 runs
2023-01-28 22:52:46 INFO TimerLogger - Raw time per run (mSec): 15.10
2023-01-28 22:52:46 INFO TimerLogger - Normalized time per run (n^2): 3.77
2023-01-28 22:52:46 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 10 runs
2023-01-28 22:52:47 INFO TimerLogger - Raw time per run (mSec): 90.90
2023-01-28 22:52:47 INFO TimerLogger - Normalized time per run (n^2 log n): 2.07
2023-01-28 22:52:47 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 10 runs
2023-01-28 22:53:20 INFO TimerLogger - Raw time per run (mSec): 2827.00
2023-01-28 22:53:20 INFO TimerLogger - Normalized time per run (n^3): .35
ThreeSumBenchmark: N=4000
2023-01-28 22:53:20 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 5 runs
2023-01-28 22:53:21 INFO TimerLogger - Raw time per run (mSec): 90.20
2023-01-28 22:53:21 INFO TimerLogger - Normalized time per run (n^2): 5.64
2023-01-28 22:53:21 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 5 runs
2023-01-28 22:53:24 INFO TimerLogger - Raw time per run (mSec): 464.20
2023-01-28 22:53:24 INFO TimerLogger - Normalized time per run (n^2 log n): 2.42
2023-01-28 22:53:24 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 5 runs
2023-01-28 22:55:29 INFO TimerLogger - Raw time per run (mSec): 17893.00
2023-01-28 22:55:29 INFO TimerLogger - Normalized time per run (n^3): .28
ThreeSumBenchmark: N=8000
2023-01-28 22:55:29 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 3 runs
2023-01-28 22:55:32 INFO TimerLogger - Raw time per run (mSec): 556.00
2023-01-28 22:55:32 INFO TimerLogger - Normalized time per run (n^2): 8.69
2023-01-28 22:55:32 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 3 runs
2023-01-28 22:55:43 INFO TimerLogger - Raw time per run (mSec): 2128.33
2023-01-28 22:55:43 INFO TimerLogger - Normalized time per run (n^2 log n): 2.56
ThreeSumBenchmark: N=16000
2023-01-28 22:55:43 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 2 runs
2023-01-28 22:55:55 INFO TimerLogger - Raw time per run (mSec): 2939.00
2023-01-28 22:55:55 INFO TimerLogger - Normalized time per run (n^2): 11.40
2023-01-28 22:55:55 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 2 runs
2023-01-28 22:56:36 INFO TimerLogger - Raw time per run (mSec): 10934.50
2023-01-28 22:56:36 INFO TimerLogger - Normalized time per run (n^2 log n): 3.06
```

## Unit Test Screenshots:

The screenshot shows an IDE with the `ThreeSumTest` class selected. The console output displays the results of 11 unit tests, all of which passed. The tests are grouped by the number of runs (10, 5, and 3) and the number of elements (N=2000, N=4000, N=8000, and N=16000). Each group shows the raw time per run in milliseconds and the normalized time per run in seconds.

```
Run: ThreeSumTest
Tests passed: 11 of 11 tests - 3sec 7ms
ThreeSumTest (edu.neu.coe.info6205) 3sec 7ms
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
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=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}]
Process finished with exit code 0
```