# Program Structures and Algorithms

## Spring 2023(SEC-01)

### Assignment 2

Name: Raja Shekar Reddy Siriganagari

NUID: 002762145

**Task:**

Solve 3-SUM using the Quadrithmic, Quadratic, and (bonus point) quadraticWithCalipers approaches, as shown in skeleton code in the repository. There are hints at the end of Lesson 2.5 Entropy.

There are also hints in the comments of the existing code.  There are a number of unit tests which you should be able to run successfully.

Submit (in your own repository--see instructions elsewhere--include the source code and the unit tests of course):

(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.

**Benchmarks for different solutions:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N |  | Quadratic | | Quadrathmic | | Cubic | |
|  |  | Time (in ms) | lg ratio | Time (in ms) | lg ratio | Time (in ms) | lg ratio |
| 250 | Raw time | 1.12 |  | 1.42 |  | 4.14 |  |
|  | Normalized time | 17.92 |  | 2.85 |  | 0.26 |  |
| 500 | Raw time | 1.86 | 0.73180389 | 4.18 | 1.55761201 | 25.16 | 2.603429249 |
|  | Normalized time | 7.44 |  | 1.86 |  | 0.20 |  |
| 1000 | Raw time | 5.25 | 1.4970148 | 16.2 | 1.95441897 | 169.30 | 2.750378146 |
|  | Normalized time | 5.25 |  | 1.63 |  | 0.17 |  |
| 2000 | Raw time | 28.5 | 2.44057259 | 78.7 | 2.28036982 | 1336.20 | 2.980482086 |
|  | Normalized time | 7.12 |  | 1.79 |  | 0.17 |  |
| 4000 | Raw time | 150.8 | 2.4036026 | 389.2 | 2.30607617 | 10630.80 | 2.992042299 |
|  | Normalized time | 9.43 |  | 2.03 |  | 0.17 |  |
| 8000 | Raw time | 867.33 | 2.52394458 | 1669.67 | 2.10097938 | 87485.67 | 3.041 |
|  | Normalized time | 13.55 |  | 2.01 |  | 0.17 |  |
| 16000 | Raw time | 4457.5 | 2.36158188 | 7946 | 2.2506658 | 728633.00 | 3.058 |
|  | Normalized time | 17.41 |  | 2.22 |  | 0.18 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| N |  | QuadraticWithCalipers | |
|  |  | Time (in ms) | lg ratio |
| 250 | Raw time | 0.85 |  |
|  | Normalized time | 13.6 |  |
| 500 | Raw time | 1.18 | 0.473252 |
|  | Normalized time | 4.72 |  |
| 1000 | Raw time | 5.2 | 2.139725 |
|  | Normalized time | 5.2 |  |
| 2000 | Raw time | 25.6 | 2.29956 |
|  | Normalized time | 6.4 |  |
| 4000 | Raw time | 178.4 | 2.8009 |
|  | Normalized time | 11.15 |  |
| 8000 | Raw time | 690.67 | 1.952881 |
|  | Normalized time | 10.79 |  |
| 16000 | Raw time | 3919.5 | 2.504601 |
|  | Normalized time | 15.31 |  |

**Why the quadratic solution works:**

The quadratic solution for three-sum is a significant improvement over the solution.

Instead of finding triplets by forming a pair and searching for the third element that sums to zero we can use a two-pointer approach on the sorted array and compares elements carefully and eliminates the need to perform binary search for each element pair achieving a worst-case runtime of . This provides us the improvement over the solution which fixes a pair and searches for the third element in the triplet performing a binary search on the sorted array.

The two-pointer quadratic approach has the following steps:

1. Fix an element in the array of size **n**.
2. Let **i=j-1** and **k=j+1** be two pointers
3. Form a triplet **()** and check if it sums to zero.
4. If the sum is less than zero move the pointer **k** forwards else pointer **i** backwards
5. Repeat the process until **i>=0 & k<n**

For every value of **j** in the range **(0, n-1)** we do **n** comparisons across the array.

So, the total number of comparisons amount to taking the worst-case time complexity to .

**Quadratic with callipers:**

A variation of this is fixing the first index and converging from opposite ends of the array. In this case we always fix the first index and move forward and not consider the index again.

The steps are as follows:

1. Fix an element in the array of size **n**.
2. Let **j=i+1** and **k=n-1** be two pointers
3. Form a triplet **()** and check if it sums to zero.
4. If the sum is less than zero move the pointer **j** forwards else pointer **k** backwards
5. Repeat the process until

For value **i=0** we go through each value of at most **n-1** times. For **i=1** we go through the array at most **n-2** times and so on.

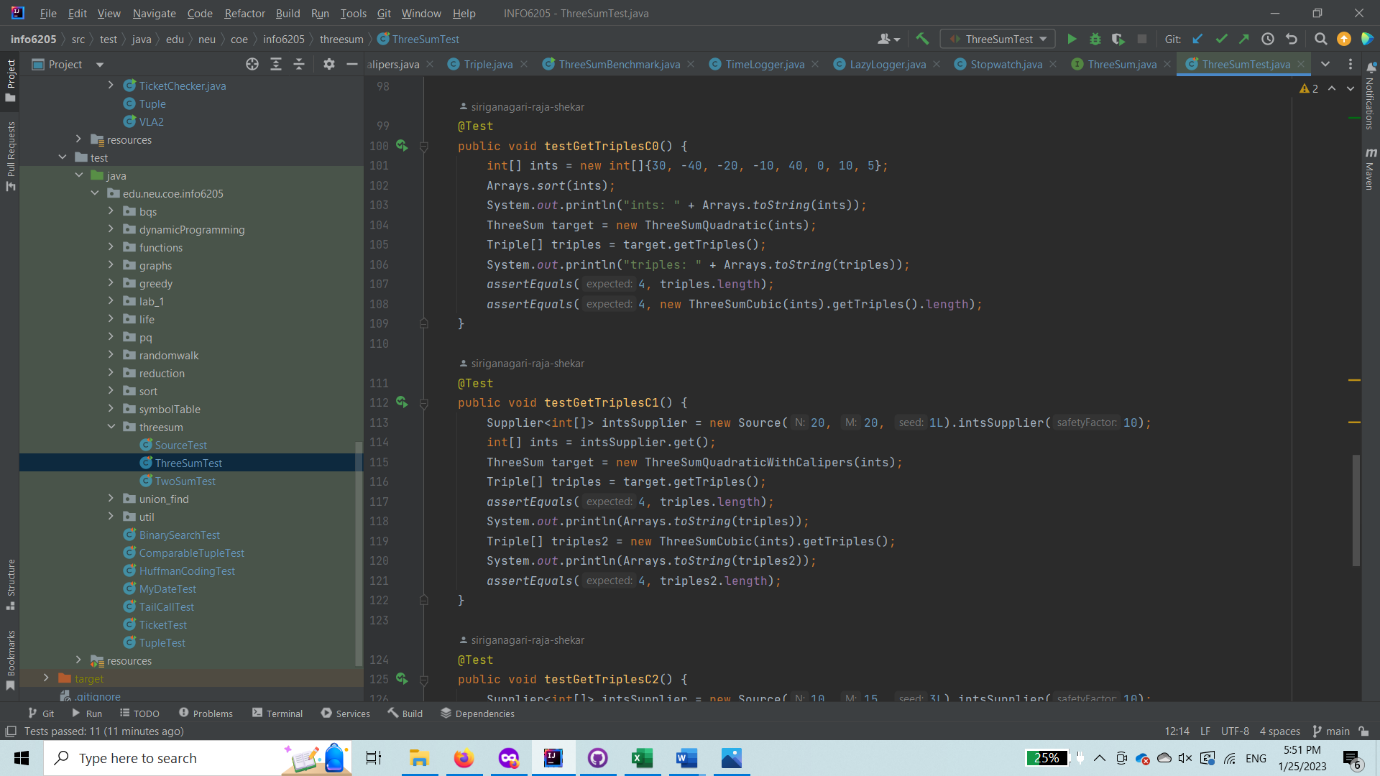
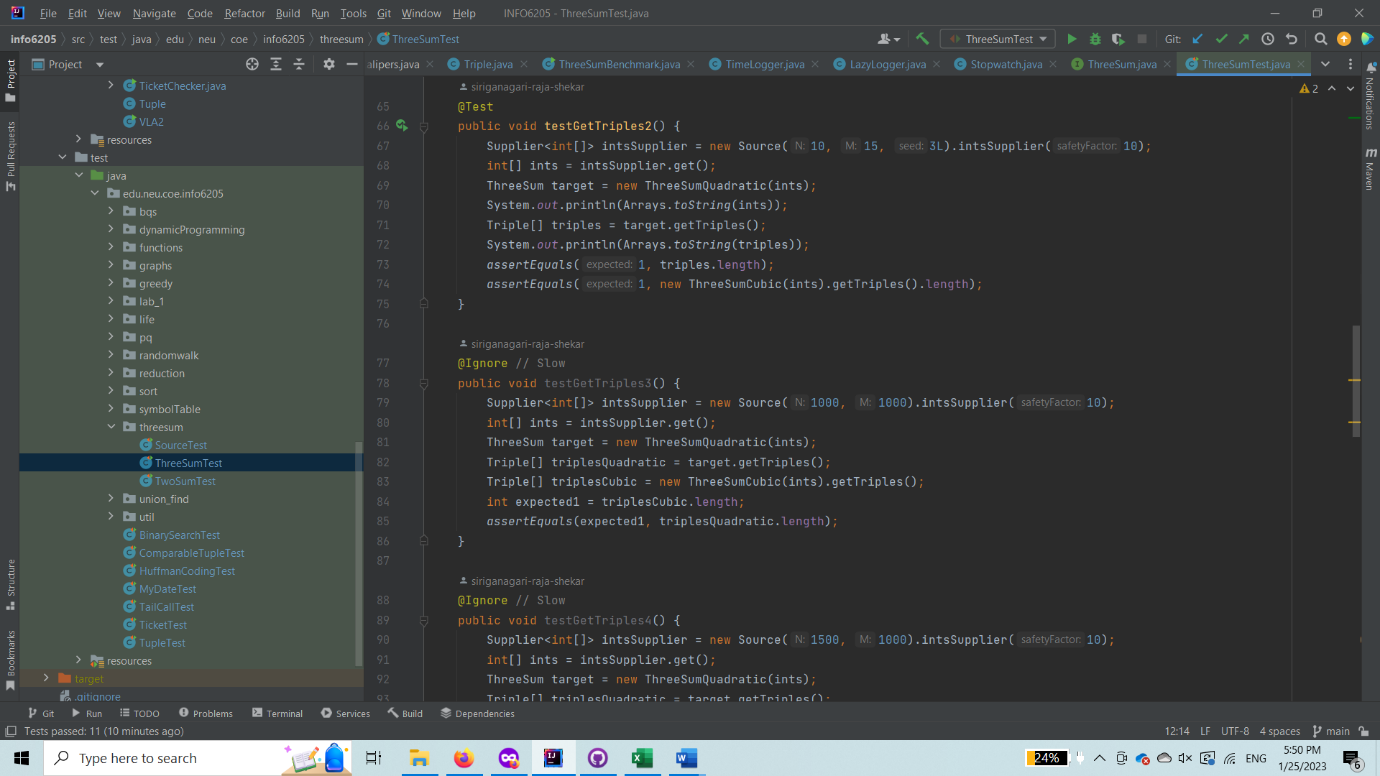
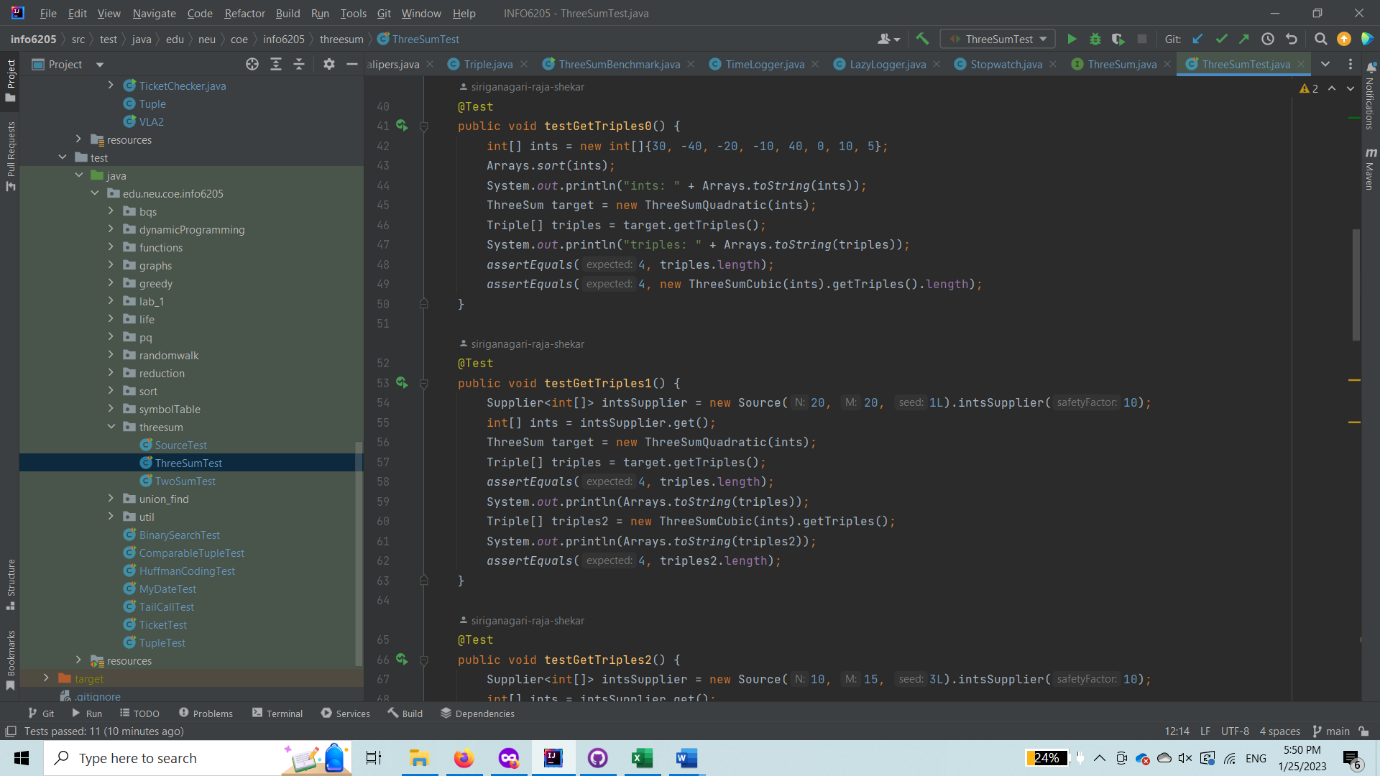
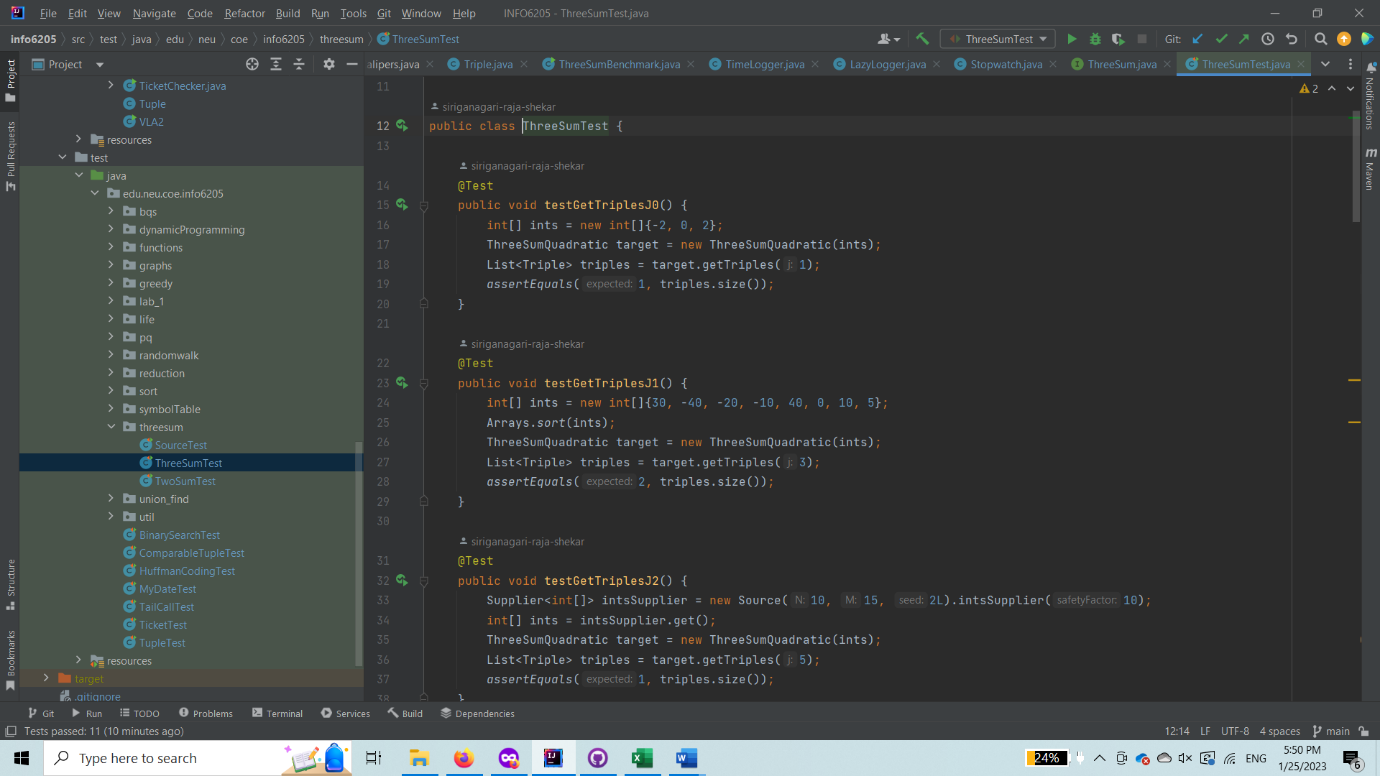
The total number of comparisons are:

This gives us the worst-case complexity of .

Below is a graph that shows the comparisons between cubic, quadratic and the solution(quadrathmic):

Chart, line chart

Description automatically generated

**Unit Tests Screenshots:A screenshot of a computer

Description automatically generated**