**Program Structures and Algorithms**

**Spring 2023(SEC – 01)**

**NAME:** Neha Rastogi

**NUID**: 002709191

**ASSIGNMENT**: 2

**Task:** Solve 3-SUM using the *Quadratic*, and *quadraticWithCalipers* approaches and showing timing observations (using the doubling method for at least five values of N) for each of the algorithms (Cubic, Quadrithmic, Quadratic and Quadratic using Calipers) using the ThreeSumBenchmark class followed by an explanation for why the quadratic methods work so well.

**Relationship Conclusion:** It is clear from the values obtained on running the different algorithms of varying time complexities that the Calipers method outperforms the rest of the methods. The Quadratic solution follows as a close second, then the Quadrithmic and finally the Cubic method which is the brute force and is the worst.

**Brief explanation of why the quadratic methods work :**

The quadratic methods work better because the number of operations it scales in proportion to the *square* of the input. Therefore, even when the problem size of N increases a lot the time taken will not increase a lot.

Quadratic method – In this method the values of the array are passed in the outer loop as the middle index j among i, j and k. the other two indexes are to the left and right of j and move outward towards the edges of the sorted array. If at any instance the sum of the values at the I and j indexes is equivalent to negative of value at j index, we can simply add this Triplet to the List of Triplets. If the sum is lesser, the k index is increased moving towards the higher values. If the sum is higher then, the i index is reduced towards a lower value .Once we reach the ends, the iteration completes and the triplets obtained in the list are the result. The outer loop has a complexity of O(N) and the inner loop also has linear complexity thereby resulting in O(N2)

Quadratic using Calipers method – In this method the values of the array are passed in the outer loop as the index i. The other two indexes are to the left and right, l and r and move closer towards each other. The triple is passed to a function using apply which returns the sum of the triplets. If it is zero, we add this Triplet to the List of Triplets. If the sum is greater, we need a lower value which can be obtained by decreasing the right index and for a lower value we need to increase the left index. Here again, the outer loop has a complexity of O(N) and the inner loop also has linear complexity thereby resulting in O(N2)

The primary reason for the Quadratic methods to work is that we are passing a sorted array and by keeping any one of the indexes fixed(middle or left) and updating the other two indexes based on the sum within a second loop we do not introduce any additional computations or searching and the decision to increase or decrease any index relies on the intuition that the larger values lie to the right side of the array and the smaller values are towards the left.

On the other hand, the Quadrithmic method uses an additional binary search to find the array element with value equal to the complement of sum of two of the values thereby increasing complexity and thereby time taken.

The Cubic performs the worst since it uses 3 loops nested within each other and increases run time massively.

**Evidence to support that conclusion:**

The following are screenshots of the output obtained on running the BenchmarkSum class and they show the raw and normalized values for all 4 approaches used.

Text

Description automatically generated

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

The following are screenshots of the code changes made for

ThreeSumQuadratic

**Graphical user interface, text

Description automatically generated**

ThreeSumQuadraticWithCalipers**Text

Description automatically generated**

ThreeSumBenchmark

**Graphical user interface, text

Description automatically generated**

**Graphical Representation:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Cubic O(N**3**)** | **Quadrithmic O(N2logN)** | **Quadratic O(N2)** | **Quadratic using Calipers O(N2)** |
| **250** | 3.15 | 0.85 | 0.86 | 1.14 |
| **500** | 22.04 | 2.74 | 1.34 | 1.24 |
| **1000** | 169.2 | 12.4 | 4.35 | 3.6 |
| **2000** | 1303.2 | 56.3o | 18.2 | 18.5 |
| **4000** | 10400.8 | 293 | 106 | 127 |
| **8000** | - | 1261 | 480.67 | 703.67 |
| **16000** | - | 5306.5 | 1973 | 1883 |

**Chart, line chart

Description automatically generated**

The values for cubic are computed only till N=4000 as the time that it takes is too much in comparison and is the steepest of all the lines obtained.

Though Quadrithmic does better than the first approach however when the problem size increases substantially the growth order is very large.

Both Quadratic and Quadratic using Calipers show a similar trend for lower value but with larger N Calipers does a little better.

Below we can see the individual graphs for each of the 4 approaches with the times taken for different values of N

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Unit Test Screenshots:**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**