ASSIGNMENT 2 – 3 SUM – SATYAM JAGTAP (NUID:- 002778701)

Question:

Solve 3-SUM using the Quadrithmic, *Quadratic*, and *quadraticWithCalipers* approaches, as shown in skeleton code in the repository.

Approach:

For Cubic:

Implementing ThreeSum involves testing each option in the solution-space using brute force. An array given to the constructor can be ordered randomly.
* Construct a ThreeSumCubic on a.
* @param a :an array.

For Quadratic:

By implementing ThreeSum, an approach that partitions 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).
* Construct a ThreeSumQuadratic on a.
* @param a :a sorted array.
* 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

For Quadratic with Calipers:

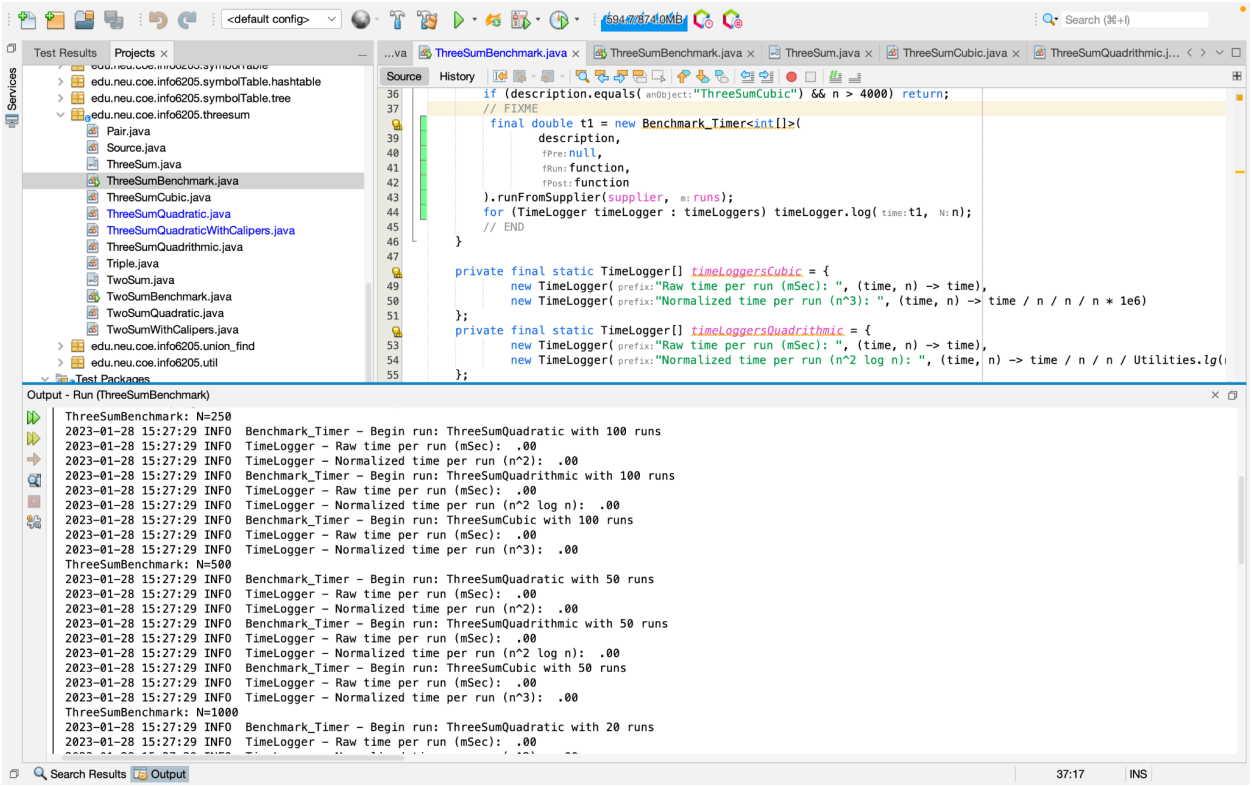
ThreeSum is an implementation strategy that segments the solution space into
* N sub-spaces where each sub-space corresponds to a fixed value for the middle index of the three values.
* Each sub-space is then solved by expanding the scope of the other two indices outwards from the starting point.
* Since each sub-space can be solved in O(N) time, the overall complexity is O(N^2). The array provided in the constructor MUST be ordered.
* Construct a ThreeSumQuadratic on a. * @param a: a sorted array.

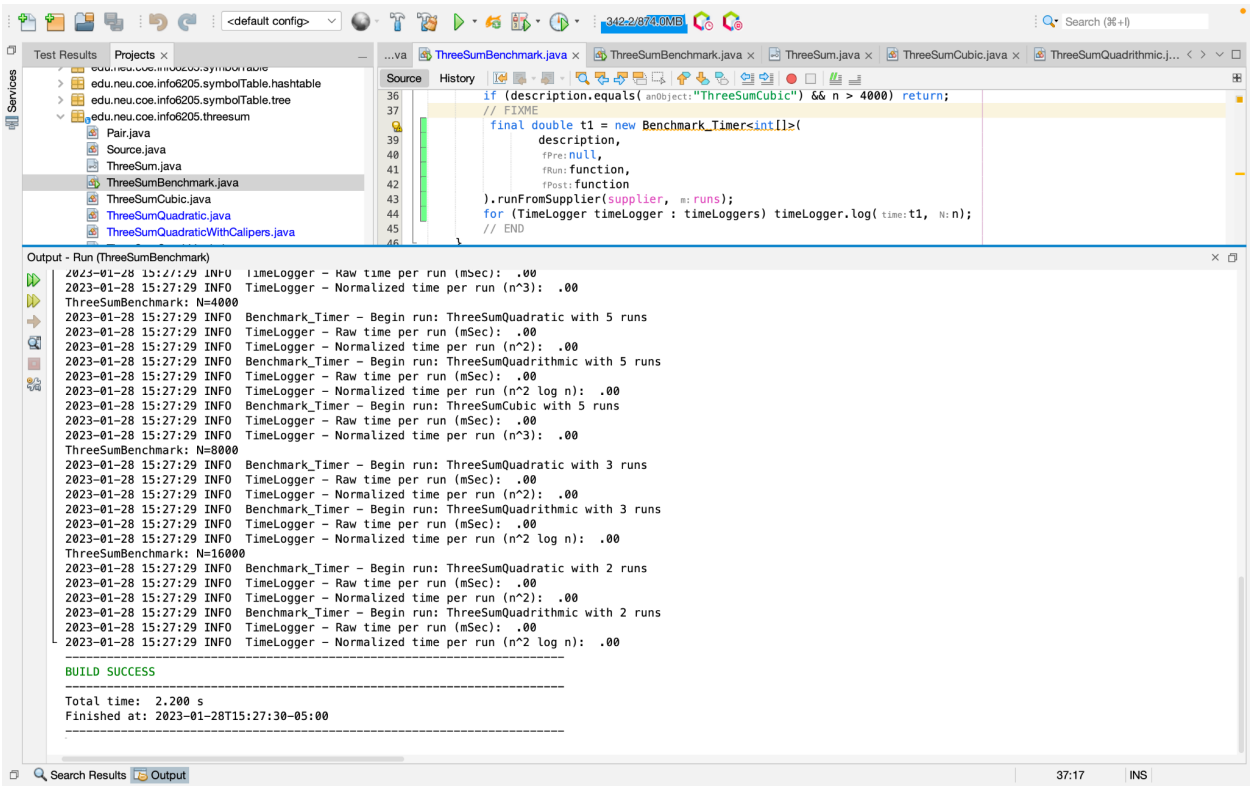
* Get a list of Triples such that the middle index is the given value i.
* @param a : a sorted array of ints.
* @param i : the index of the first element of resulting triples.
* @param function : a function which takes a triple and returns the comparison of sum of the triple with zero.
* @return a Triple

Relationship Conclusion: The outcomes of the benchmark tests demonstrate that: When we generate all possible triplets and compare the sum of each triplet with the input value, the worst-case scenario, which takes place, proceeds as follows in cubic time: O(n^3).

The average and best case scenarios both employ the method of dividing the solution-space into N sub-spaces, where each corresponds to a fixed value for the middle index of the three values. The issue is then solved by increasing the range of the first two indices for each sub-space. It is necessary to sort the provided array. Since each subspace may be solved in O(N) time, the overall complexity is O(N2).

OUTPUT:





EVIDENCE:

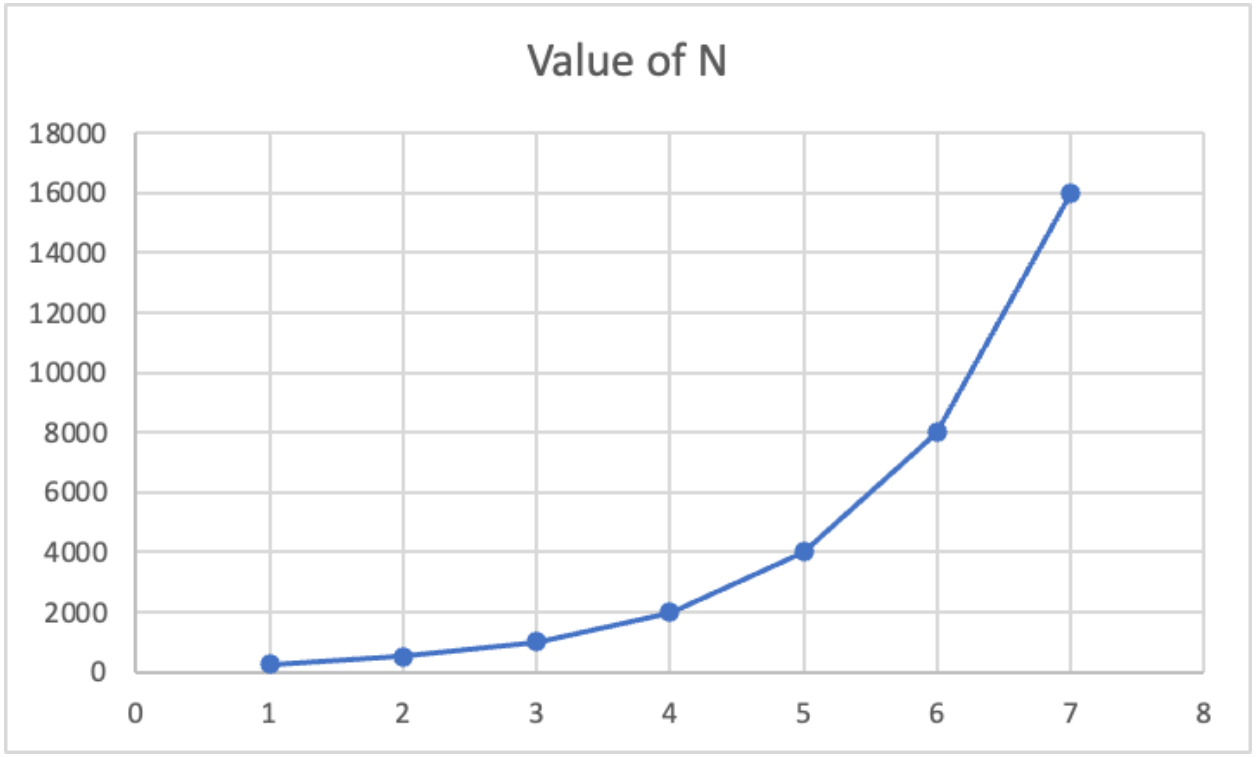
I've included a table and a chart to show how raw time and n value are related.

Value of N	Cubic Raw Time(ms)	Cubic Normalized Time(n^3)	No. of Runs
250	14.06	0.9	100
500	140.7	1.13	50
1000	1424	1.42	20
2000	9401.5	1.18	10
4000	57412.6	0.9	5
8000	NA	NA	3
16000	NA	NA	2

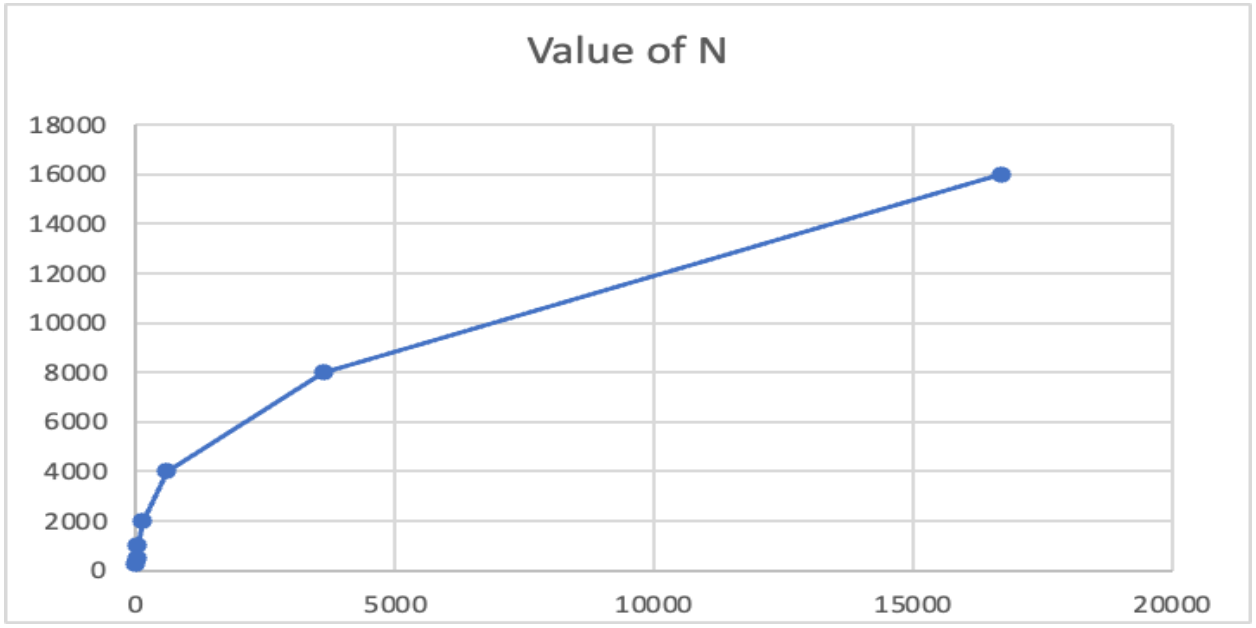
Value of N	Quadrithmic Raw Time(ms)	Quadrithmic Normalized Time(n^3)	No. of Runs
250	1.65	4.56	100
500	7.43	3.13	50
1000	31.09	2.56	20
2000	136	1.32	10
4000	590.89	3.54	5
8000	3610.67	4.45	3
16000	16702.34	4.76	2

Value of N	Quadratic Raw Time(ms)	Quadratic Normalized Time(n^3)	No. of Runs
250	1.34	18.75	100
500	1.74	6.74	50
1000	6.5	6.2	20
2000	44.6	11.99	10
4000	285.56	18.52	5
8000	1224	20.26	3
16000	6137.89	24	2

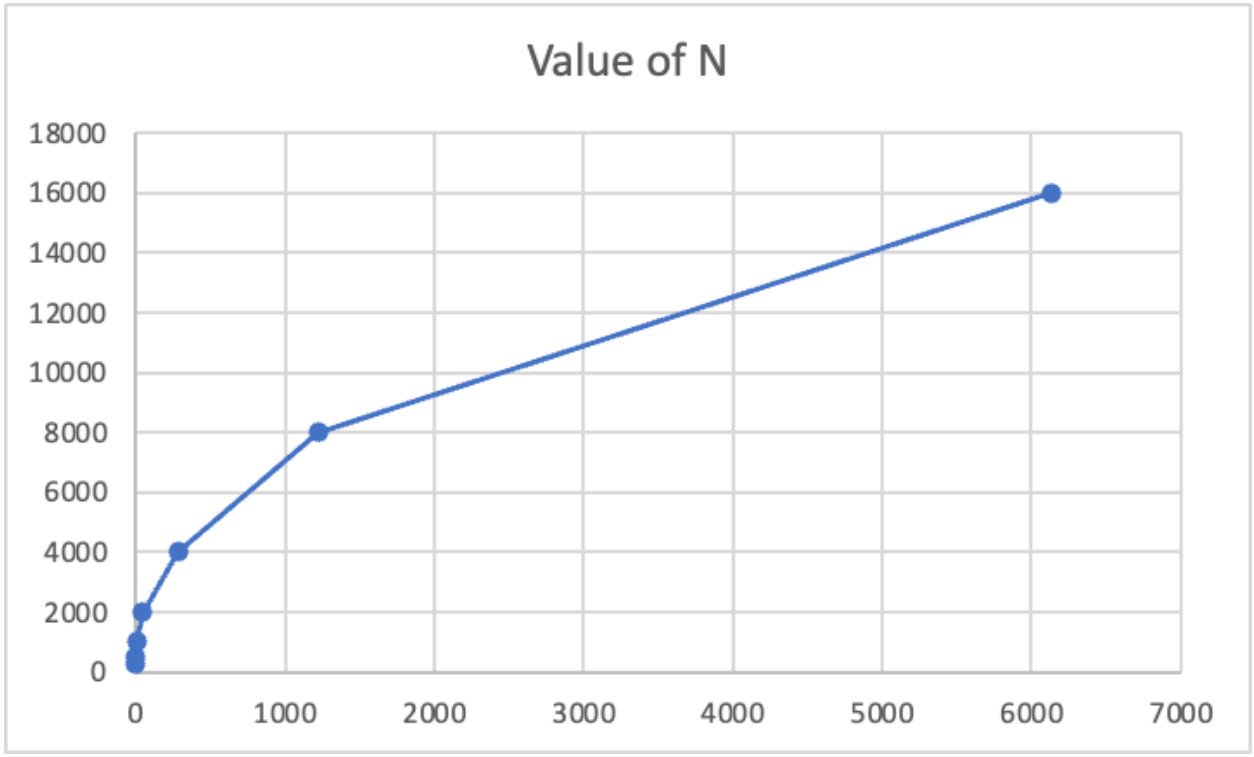
CUBIC



QUADRITHMIC



QUADRATIC



Unit Test Cases: