

SHREYA BALIGA (NUID:002795178)

PROGRAM STRUCTURES AND ALGORITHM ASSIGNMENT-2

TASK:

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

Approach:

For Cubic:

Implementation of ThreeSum which follows the brute-force approach of testing every candidate in the solution-space. The array provided in the constructor may be randomly ordered.

- * Construct a ThreeSumCubic on a.

- * @param a :an array.

For Quadratic:

Implementation of ThreeSum which follows the approach of dividing 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:

Implementation of ThreeSum which follows the approach of dividing 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: It can be observed from the results of the benchmark test: In the worst case scenario which happens when we generate all possible triplets and compare the sum of every triplet with the given value and therefore, runs in cubic time: $O(n^3)$.

In the average and best case scenario which follows the approach of dividing 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. The array provided must be sorted. Since each sub-space can be solved in $O(N)$ time, the overall complexity is $O(N^2)$.

OUTPUT:

```
Console x
ThreeSumBenchmark [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (28-Jan-2023, 9:27:24 am) [pid: 9728]
ThreeSumBenchmark: N=250
2023-01-28 09:27:29 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 100 runs
2023-01-28 09:27:30 INFO TimeLogger - Raw time per run (mSec): 1.00
2023-01-28 09:27:30 INFO TimeLogger - Normalized time per run (n^2): 16.00
2023-01-28 09:27:30 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 100 runs
2023-01-28 09:27:30 INFO TimeLogger - Raw time per run (mSec): 1.46
2023-01-28 09:27:30 INFO TimeLogger - Normalized time per run (n^2 log n): 2.93
2023-01-28 09:27:30 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 100 runs
2023-01-28 09:27:33 INFO TimeLogger - Raw time per run (mSec): 14.06
2023-01-28 09:27:33 INFO TimeLogger - Normalized time per run (n^3): .90

ThreeSumBenchmark: N=500
2023-01-28 09:27:33 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 50 runs
2023-01-28 09:27:33 INFO TimeLogger - Raw time per run (mSec): 1.94
2023-01-28 09:27:33 INFO TimeLogger - Normalized time per run (n^2): 7.76
2023-01-28 09:27:33 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 50 runs
2023-01-28 09:27:34 INFO TimeLogger - Raw time per run (mSec): 7.70
2023-01-28 09:27:34 INFO TimeLogger - Normalized time per run (n^2 log n): 3.44
2023-01-28 09:27:34 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 09:27:49 INFO TimeLogger - Raw time per run (mSec): 140.70
2023-01-28 09:27:49 INFO TimeLogger - Normalized time per run (n^3): 1.13

ThreeSumBenchmark: N=1000
2023-01-28 09:27:49 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 20 runs
2023-01-28 09:27:49 INFO TimeLogger - Raw time per run (mSec): 11.90
2023-01-28 09:27:49 INFO TimeLogger - Normalized time per run (n^2): 11.90
2023-01-28 09:27:49 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 20 runs
2023-01-28 09:27:51 INFO TimeLogger - Raw time per run (mSec): 37.95
2023-01-28 09:27:51 INFO TimeLogger - Normalized time per run (n^2 log n): 3.81
2023-01-28 09:27:51 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 20 runs
2023-01-28 09:28:50 INFO TimeLogger - Raw time per run (mSec): 1424.00
2023-01-28 09:28:50 INFO TimeLogger - Normalized time per run (n^3): 1.42

ThreeSumBenchmark [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (28-Jan-2023, 9:27:24 am) [pid: 9728]
2023-01-28 09:28:50 INFO TimeLogger - Raw time per run (mSec): 1424.00
2023-01-28 09:28:50 INFO TimeLogger - Normalized time per run (n^3): 1.42
ThreeSumBenchmark: N=2000
2023-01-28 09:28:50 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 10 runs
2023-01-28 09:28:51 INFO TimeLogger - Raw time per run (mSec): 65.50
2023-01-28 09:28:51 INFO TimeLogger - Normalized time per run (n^2): 16.38
2023-01-28 09:28:51 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 10 runs
2023-01-28 09:28:57 INFO TimeLogger - Raw time per run (mSec): 240.70
2023-01-28 09:28:57 INFO TimeLogger - Normalized time per run (n^2 log n): 5.49
2023-01-28 09:28:57 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 10 runs

ThreeSumBenchmark: N=4000
2023-01-28 09:32:18 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 5 runs
2023-01-28 09:32:21 INFO TimeLogger - Raw time per run (mSec): 252.60
2023-01-28 09:32:21 INFO TimeLogger - Normalized time per run (n^2): 15.79
2023-01-28 09:32:21 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 5 runs
2023-01-28 09:32:28 INFO TimeLogger - Raw time per run (mSec): 553.00
2023-01-28 09:32:28 INFO TimeLogger - Normalized time per run (n^2 log n): 2.89
2023-01-28 09:32:28 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 5 runs
```

```

Console x
ThreeSumBenchmark [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (28-Jan-2023, 10:02:26 pm) [pid: 10168]
ThreeSumBenchmark: N=250
2023-01-28 22:02:26 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 100 runs
2023-01-28 22:02:27 INFO TimeLogger - Raw time per run (mSec): .91
2023-01-28 22:02:27 INFO TimeLogger - Normalized time per run (n^2): 14.56
2023-01-28 22:02:27 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 100 runs
2023-01-28 22:02:27 INFO TimeLogger - Raw time per run (mSec): 1.34
2023-01-28 22:02:27 INFO TimeLogger - Normalized time per run (n^2 log n): 2.69
2023-01-28 22:02:27 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 100 runs
2023-01-28 22:02:30 INFO TimeLogger - Raw time per run (mSec): 13.95
2023-01-28 22:02:30 INFO TimeLogger - Normalized time per run (n^3): .89
2023-01-28 22:02:30 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 100 runs
ThreeSumBenchmark: N=500
2023-01-28 22:02:30 INFO Benchmark_Timer - Begin run: ThreeSumQuadratic with 50 runs
2023-01-28 22:02:31 INFO TimeLogger - Raw time per run (mSec): 2.48
2023-01-28 22:02:31 INFO TimeLogger - Normalized time per run (n^2): 9.92
2023-01-28 22:02:31 INFO Benchmark_Timer - Begin run: ThreeSumQuadrithmic with 50 runs
2023-01-28 22:02:32 INFO TimeLogger - Raw time per run (mSec): 9.74
2023-01-28 22:02:32 INFO TimeLogger - Normalized time per run (n^2 log n): 4.35
2023-01-28 22:02:32 INFO Benchmark_Timer - Begin run: ThreeSumCubic with 50 runs
2023-01-28 22:02:44 INFO TimeLogger - Raw time per run (mSec): 119.40
2023-01-28 22:02:44 INFO TimeLogger - Normalized time per run (n^3): .96
2023-01-28 22:02:44 INFO Benchmark_Timer - Begin run: ThreeSumQuadraticWithCalipers with 50 runs

```

EVIDENCE:

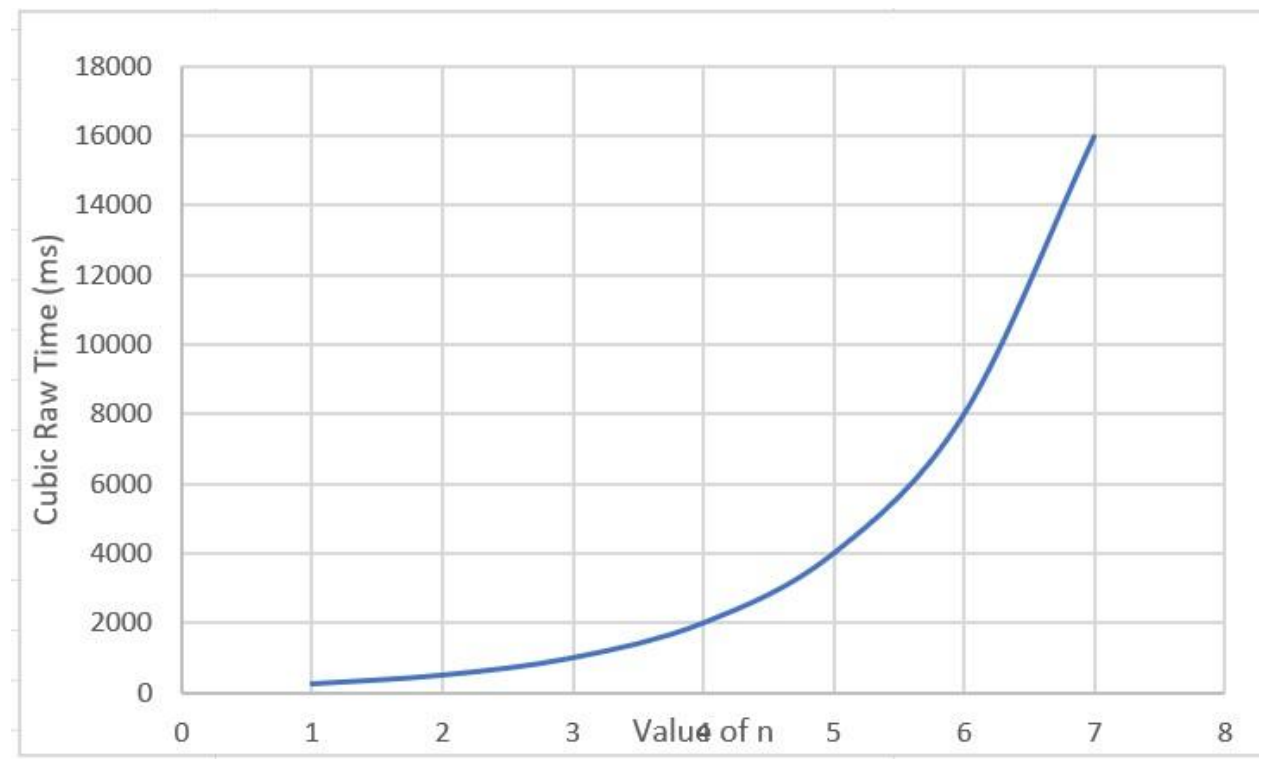
I have attached a table and chart to show the relationship between raw time and value of n.

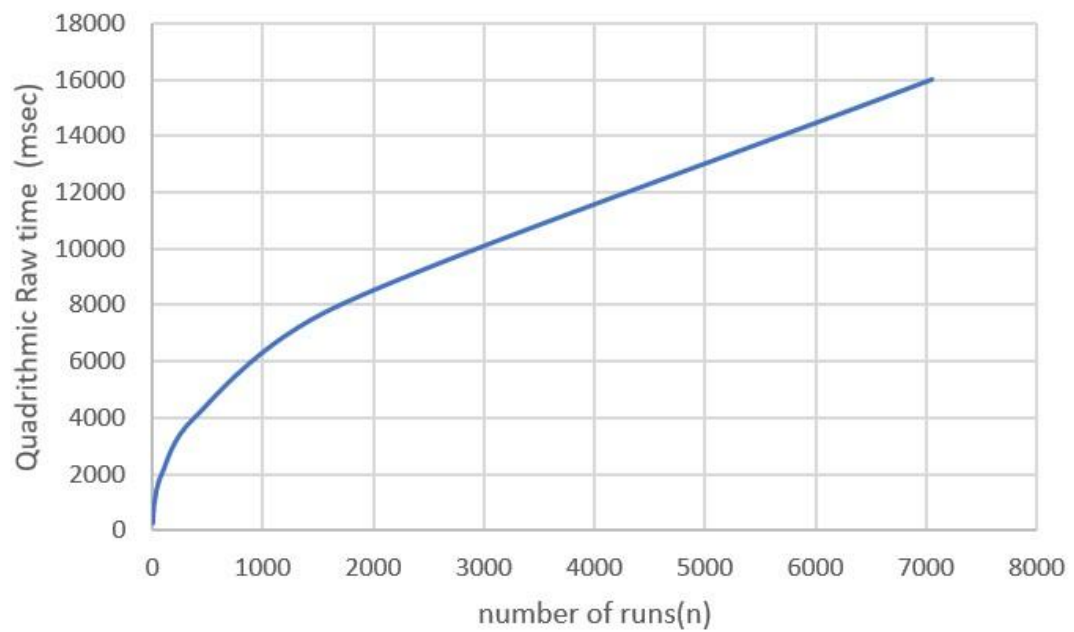
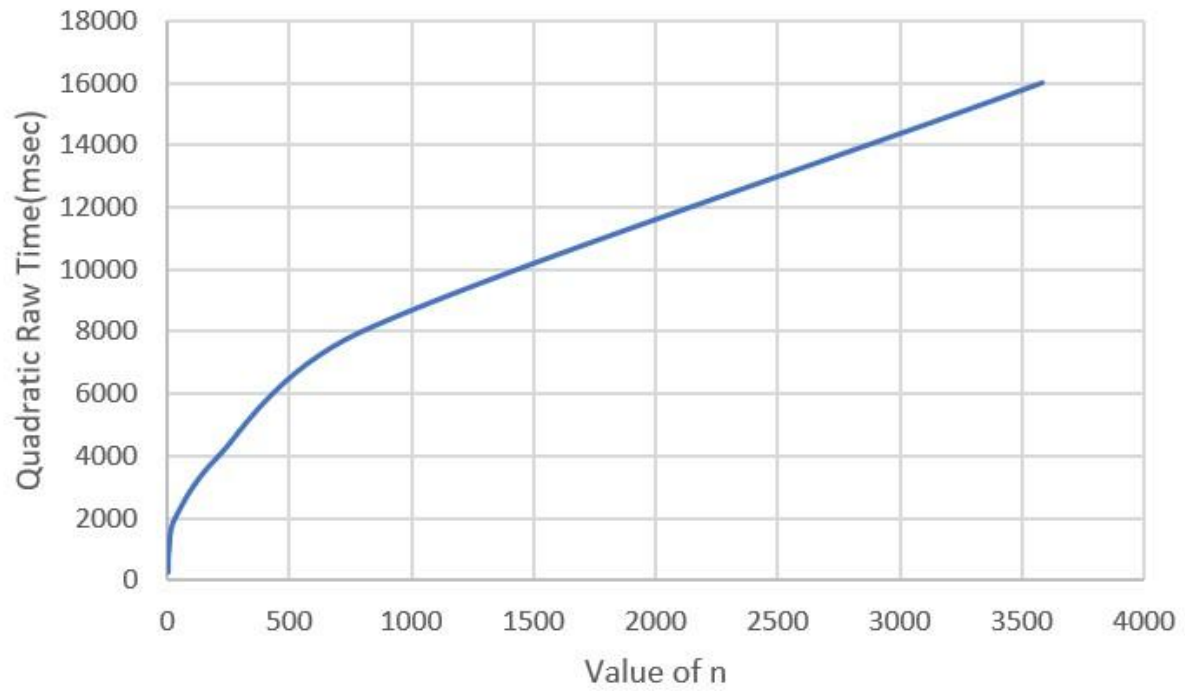
Value of n	No. of runs	Cubic Raw Time(ms)	Cubic Normalized Time(n^3)
250	100	14.06	0.9
500	50	140.7	1.13
1000	20	1424	1.42
2000	10	9401.5	1.18
4000	5	57412.6	0.9
8000	3 NA	NA	NA
16000	2 NA	NA	NA

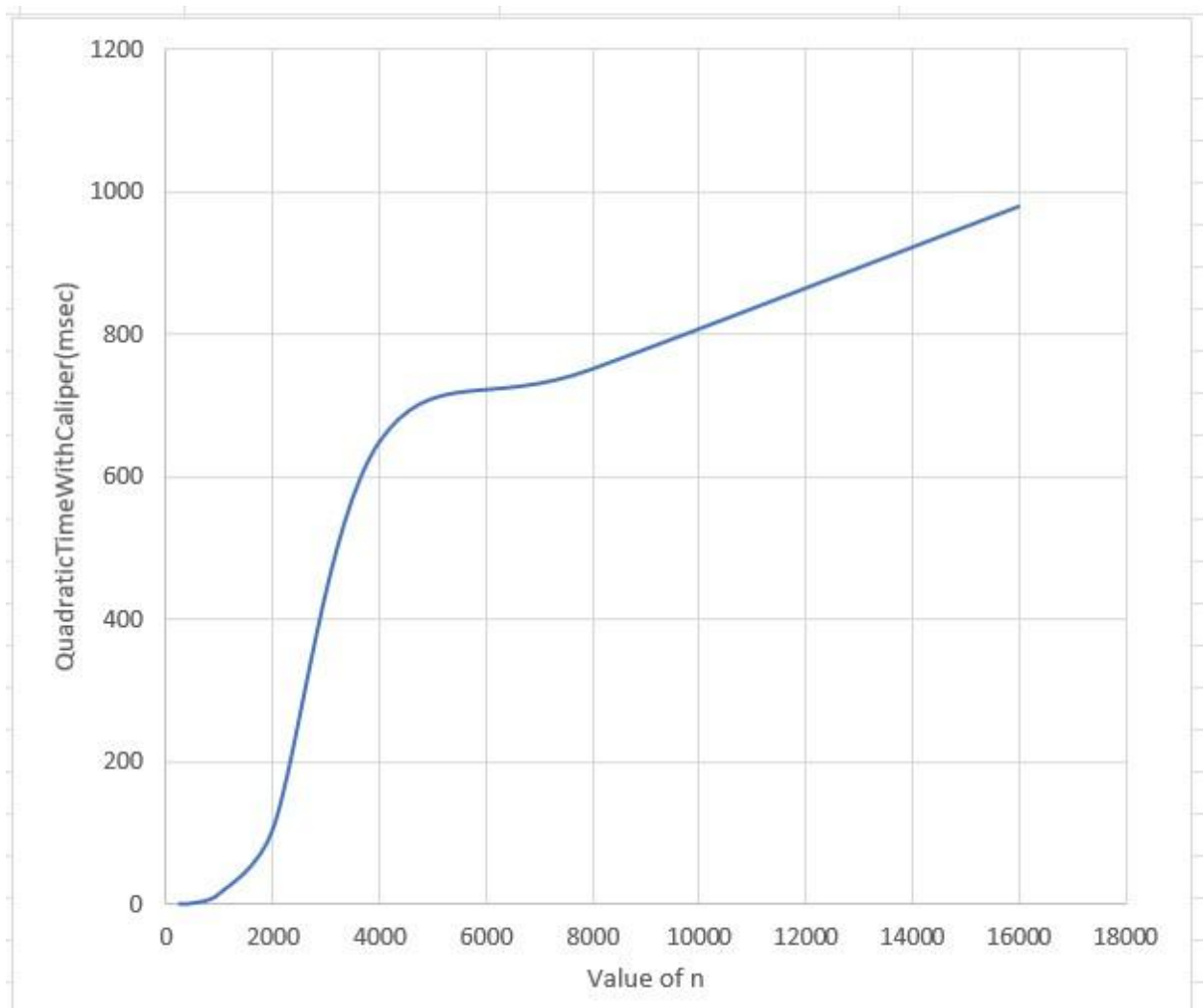
Value of n	No. of runs	Quadrithmic Raw Time(ms)	Quadrithmic Normalized Time(n^2logn)
250	100	1.46	4.45
500	50	6.82	3.04
1000	20	29.35	2.95
2000	10	131.8	1.11
4000	5	584.2	3.05
8000	3	3608.67	4.35
16000	2	16686	4.67

Value of n	No. of runs	Quadratic Raw Time(ms)	Quadratic Normalized Time(n^2)
250	100	1.17	18.72
500	50	1.68	6.72
1000	20	6.2	6.2
2000	10	44.2	11.05
4000	5	283	17.69
8000	3	1223	19.11
16000	2	6137.5	23.97

Value of n	No. of runs	QuadraticWithCalipers Raw Time(msec)	QuadraticWithCalipers Normalized Time(n^2)
250	100	1.22	19.52
500	50	2.48	9.92
1000	20	15.95	15.95
2000	10	105.6	26.4
4000	5	648.5	3.49
8000	3	750.65	4.45
16000	2	978.2	9.78

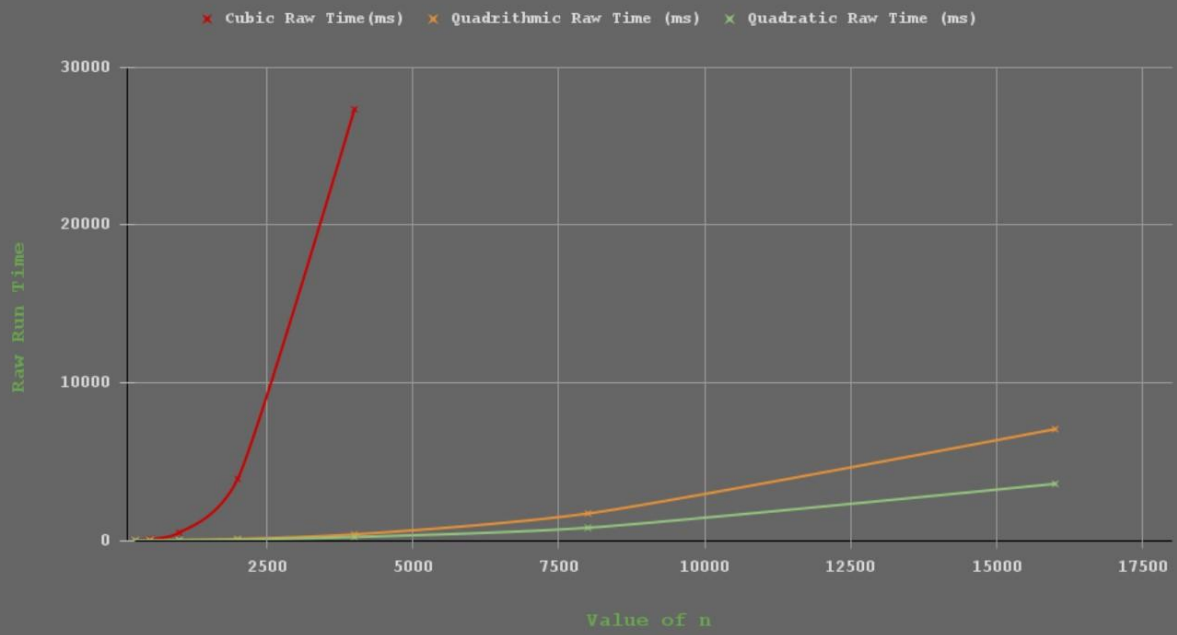






ThreeSum

Cubic, Quadrithmic & Quadratic Raw Time



Passed Unit Test Cases:

The screenshot shows an IDE with the following components:

- Top Left:** Test runner status bar showing "Finished after 2.369 seconds", "Runs: 11/11", "Errors: 0", and "Failures: 0".
- Top Center:** Source code for `TimerException` and `LazyLogger`. The `TimerException` class is highlighted, showing its constructor and methods.
- Top Right:** A list of fields and methods for the `TimerException` class, including `milliseconds`, `toString`, `ticks`, `laps`, `running`, `getTicks`, `getLaps`, `isRunning`, `getClock`, `toMilliseconds`, and `logger`.
- Bottom:** A console window showing the output of the unit test. The output indicates that the test passed and displays the results of the `Triple` and `ints` arrays.

```
<terminated> ThreeSumTest [JUnit] C:\Program Files\Java\jdk-19\bin\javaw.exe (28-Jan-2023, 9:31:08 am - 9:31:11 am) [pid: 12636]
[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)]
```