

Program Structures and Algorithms Spring 2024

NAME: Mihir Ravindra Adedkar

NUID: 002810839

GITHUB LINK: <https://github.com/mihiradelkar/INFO6205-PSA>

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.

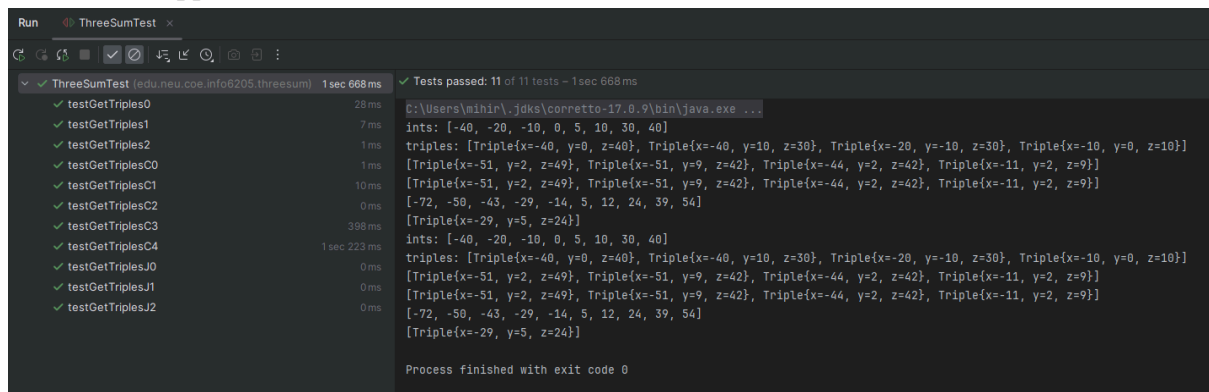
Submit (in your own repository--see instructions elsewhere--include the source code and the unit tests of course):

(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.

Evidence to support that conclusion:



```
Run ThreeSumTest x
C:\Users\mihir\jdk\corretto-17.0.9\bin\java.exe ...
Tests passed: 11 of 11 tests - 1 sec 668 ms
testGetTriples0 28 ms
testGetTriples1 7 ms
testGetTriples2 1 ms
testGetTriplesC0 1 ms
testGetTriplesC1 10 ms
testGetTriplesC2 0 ms
testGetTriplesC3 398 ms
testGetTriplesC4 1 sec 223 ms
testGetTriplesJ0 0 ms
testGetTriplesJ1 0 ms
testGetTriplesJ2 0 ms
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[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}]
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[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
```

Test Result Sheet:

	N	ThreeSumQuadratic Raw Time (ms)	ThreeSumQuadratic Normalized Time (n^2)	ThreeSumQuadrithmic Raw Time (ms)	ThreeSumQuadrithmic Normalized Time (n^2 $\log n$)	ThreeSumCubic Raw Time (ms)	ThreeSumCubic Normalized Time (n^3)
0	250	2.31	36.96	1.81	3.64	8.29	0.53
1	500	2.48	9.92	4.74	2.11	53.66	0.43
2	1000	6.10	6.10	20.95	2.10	420.05	0.42
3	2000	24.30	6.08	97.60	2.23	3535.3	0.44
4	4000	207.20	12.95	482.80	2.52	28132.8	0.44
5	8000	869.67	13.59	2375.33	2.86	N/A	N/A
6	16000	5847.50	22.84	9756.00	2.73	N/A	N/A

Explanation:

The quadratic method for solving the ThreeSum problem works based on a couple of key principles:

Sorting: The first step in the quadratic approach is to sort the input array. This ordering is crucial as it allows the algorithm to efficiently find pairs that sum to a specific value (in this case, the negative of a third value).

Two-pointer Technique: Once the array is sorted, the algorithm iteratively fixes one element and then uses two pointers to scan through the array for pairs that sum up to the negative of this fixed element. The pointers typically start at positions just after the fixed element and at the end of the array, moving towards each other.

If the sum of the elements at the two pointers is too large, the end pointer is moved backward.

If the sum is too small, the start pointer is moved forward.

If the sum is just right (equal to the negative of the fixed element), a valid triplet is found.

Time Complexity: This process is $O(N)$ for each fixed element (since each element is visited at most once by each pointer), and since there are N elements to fix in turn, the overall time complexity is $O(N^2)$.

Avoiding Duplicates: The sorted nature of the array also makes it easier to skip over duplicate values, which is important for finding unique triplets.

Efficiency: This method is significantly more efficient than the brute-force cubic approach ($O(N^3)$), which checks all possible triplets. By fixing one element and efficiently finding pairs that sum to its negative, the quadratic method reduces the number of necessary comparisons.