PSA Spring 2023 (SEC – 8)

NAME: ANURAG NANDRE NUID:

00278573

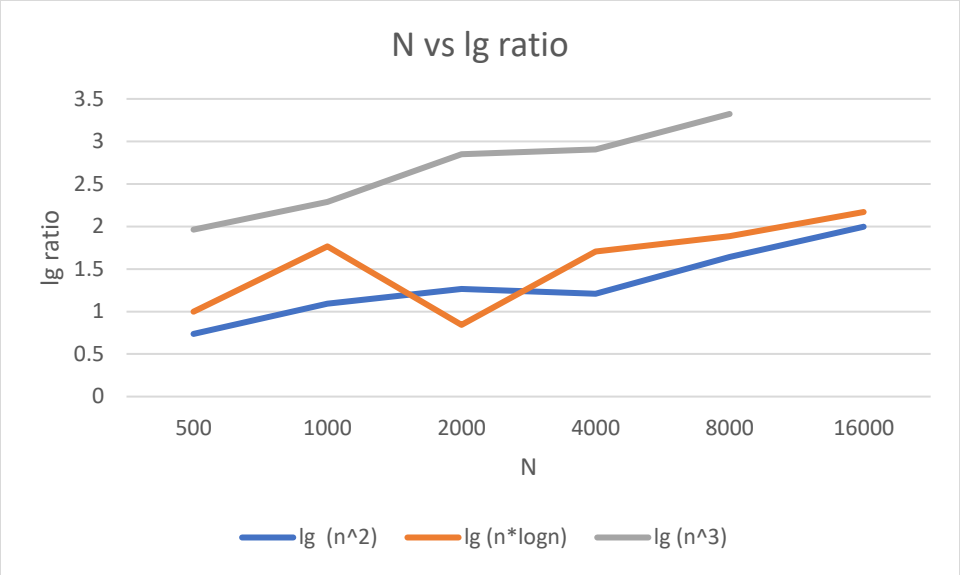
1. Task

Solve 3-SUM using the *Quadrithmic*, *Quadratic*, and (bonus point) *quadraticWithCalipers* approaches

2. Evidence :

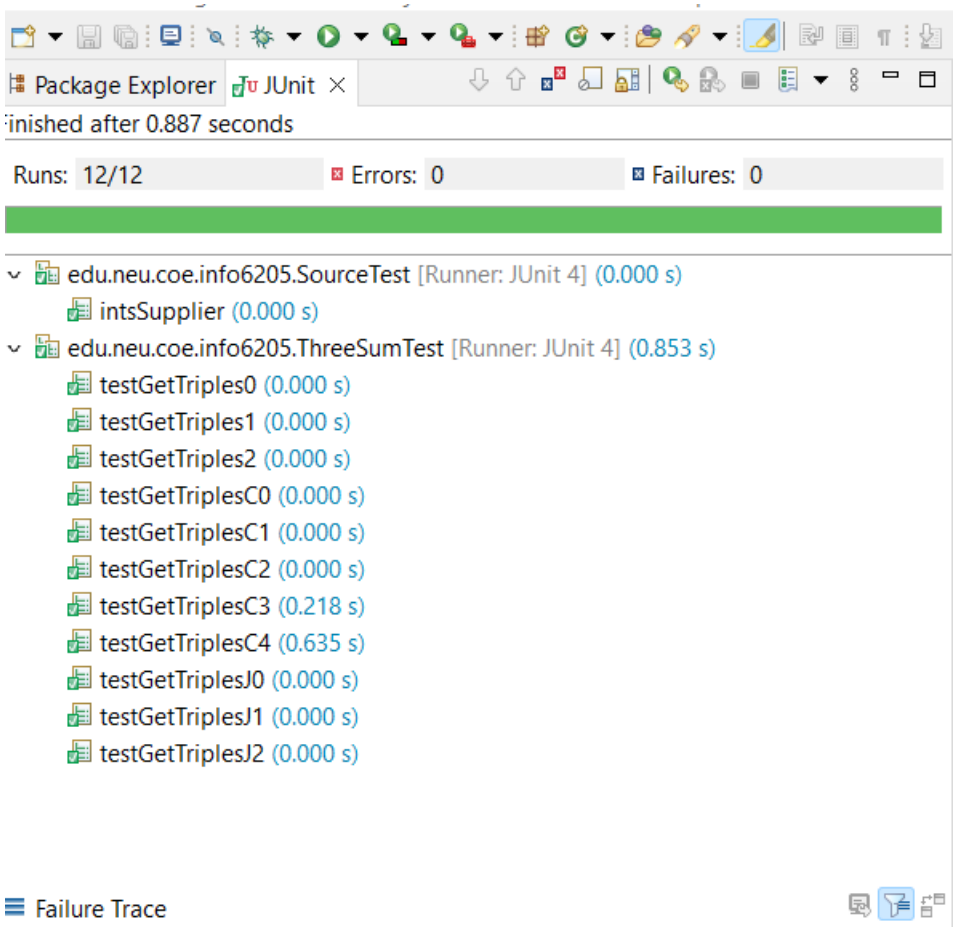
After the experiments are done the following results are observed

N		Quadratic		Quadrithmic		Cubic	
			lg ratio		*lg ratio*		*lg ratio*
250	Raw Time(millisecs)	9		5		10	
	Normalized	143.99		10.042		0.64	
500	Raw Time(millisecs)	15	0.7369656	10	1	39	1.963474
	Normalized	59.99		4.4614		0.312	
1000	Raw Time(millisecs)	32	1.0931094	34	1.765535	191	2.292027
	Normalized	32		3.411		0.191	
2000	Raw Time(millisecs)	77	1.2667865	61	0.843274	1382	2.855113
	Normalized	19.25		1.39		0.1727	
4000	Raw Time(millisecs)	178	1.2089469	199	1.705887	10358	2.905916
	Normalized	11.125		1.0394		0.16184	
8000	Raw Time(millisecs)	555	1.6406105	737	1.888896	103676	3.323265
	Normalized	8.6718		0.8881		0.202492188	
16000	Raw Time(millisecs)	2216	1.9973982	3323	2.17275		
	Normalized	8.65625		0.929447891			


N vs lg ratio
3.5
3
2.5
2
1.5
1
0.5
0
lg ratio
500
1000
2000
4000
8000
16000
N
lg (n^2)
lg (n*logn)
lg (n^3)

3. Unit Tests:

All test cases have been passed. The experiment is performed for 7 different N values. The results are as follows:



Problems @ Javadoc Declaration Console × Terminal

ThreeSumBenchmark (1) [Java Application] C:\Users\Anurag\Downloads\sts-4.15.3.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wir

```
Raw time per run (mSec): 250 is 9
Normalized time per run (n^2): 250 is 143.99999999999997
*****************************************************************************************
Raw time per run (mSec): 250 is 6
Normalized time per run (n^2 log n): 250 is 12.051544024967576
*****************************************************************************************
Raw time per run (mSec): 250 is 9
Normalized time per run (n^3): 250 is 0.5759999999999998
*****************************************************************************************
Raw time per run (mSec): 500 is 5
Normalized time per run (n^2): 500 is 20.0
*****************************************************************************************
Raw time per run (mSec): 500 is 11
Normalized time per run (n^2 log n): 500 is 4.907546133501284
*****************************************************************************************
Raw time per run (mSec): 500 is 32
Normalized time per run (n^3): 500 is 0.25599999999999995
*****************************************************************************************
Raw time per run (mSec): 1000 is 21
Normalized time per run (n^2): 1000 is 21.000000000000004
*****************************************************************************************
Raw time per run (mSec): 1000 is 9
Normalized time per run (n^2 log n): 1000 is 0.9030899869919434
*****************************************************************************************
Raw time per run (mSec): 1000 is 179
Normalized time per run (n^3): 1000 is 0.17899999999999996
*****************************************************************************************
Raw time per run (mSec): 2000 is 24
Normalized time per run (n^2): 2000 is 6.0
*****************************************************************************************
Raw time per run (mSec): 2000 is 34
Normalized time per run (n^2 log n): 2000 is 0.7751383557570983
*****************************************************************************************
Raw time per run (mSec): 2000 is 1387
Normalized time per run (n^3): 2000 is 0.173375
*****************************************************************************************
Raw time per run (mSec): 4000 is 108
Normalized time per run (n^2): 4000 is 6.75
*****************************************************************************************
Raw time per run (mSec): 4000 is 141
Normalized time per run (n^2 log n): 4000 is 0.7364749180123519
*****************************************************************************************
```

4. Explanation how Quadratic Methods work:

The basic idea of the algorithm is to sort the input array and then for each element, use two pointers (one starting from the element after the current element and another starting from the end of the array) to find the pairs of elements in the subarray that add up to the target value.

The outer loop iterates through all elements of the array and the inner loop starts from the next element and ends at the end of the array. The two pointers move towards each other depending on the sum of the current elements. If the sum is less than the target value, the left pointer is incremented, otherwise if the sum is greater than the target value, the right pointer is decremented.

This method is efficient because the array is sorted in the beginning, which allows for efficient traversal and eliminates the need for additional data structures such as hash maps.

In summary, 3sum method is solving problem of finding all unique triplets in an array of integers that add up to a given target value, it uses sorting and nested looping approach which results time complexity of O(n^2)