Program Structures and Algorithms
Spring 2023(SEC 03)

NAME: Vipul Rajderkar
NUID: 002700991

Task:
Solve 3-SUM using the Quadrithmic, Quadratic, and (bonus point) quadraticWithCalipers approaches, as shown in skeleton code in the repository.

Relationship Conclusion:
After solving 3-Sum problem using three different approaches, it can be conferred that Quadratic Approach (quadratic as well as quadratic with calipers) performs better than rest 2 viz. Quadrithimic and cubic.

Time complexities of all approaches are as below:
Quadratic: $O(n^2)$
Quadrithimic: $O(n^2 \log n)$
Cubic: $O(n^3)$

Evidence to support that conclusion:

Approach 1: 3-Sum Quadratic

Step 1: Select a number as a middle number (say x) from an array (complexity: O(n))
Step 2: Use 2 pointers pointing number less than and greater than middle number (x)
Step 3: While both pointers don't reach middle number's index -> check the sum and increment corresponding pointers depending on whether the sum is less than or equal to 0.

Code Snippet:

eeSumQuadratic › getTriples

Quadrithmic.java | ThreeSumQuadratic.java | Triple.java | ThreeSumCubic.java

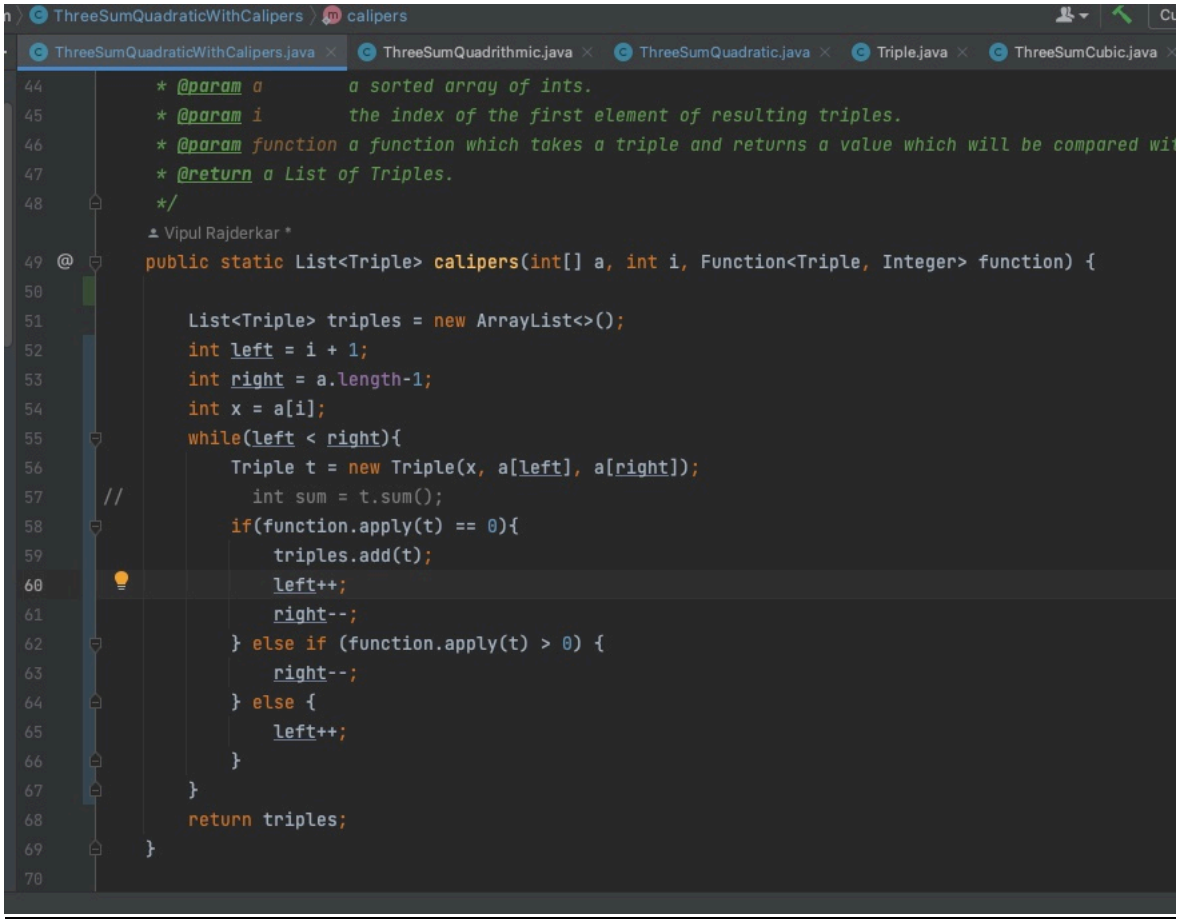
```
     * Get a list of Triples such that the middle index is the give
     *
     * @param j the index of the middle value.
     * @return a Triple such that
     */
    4 usages  Vipul Rajderkar *
    public List<Triple> getTriples(int j) {
        List<Triple> triples = new ArrayList<>();
        int left = 0;
        int right = length - 1;
        while (left < j && right>j) {
            Triple t = new Triple(a[left], a[j], a[right]);
            if (a[left]+a[j]+a[right] == 0) {
                triples.add(t);
                left++;
                right--;
            } else if (a[left]+a[j]+a[right] < 0)
                left++;
            else
                right--;
        }
        return triples;
    }
```

Approach 2: 3-Sum Quadratic with Calipers

The computation is less heavy than approach 1 because of the invariant that remaining 2 numbers are going to be more than first number.

Step 1: Select a number as a first number (say x) from a sorted array (complexity: O(n))
Step 2: Use 2 pointers pointing numbers just greater than x and the largest number.
Step 3: While both pointers don't reach middle number's index -> check the sum and increment/decrement corresponding pointers depending on whether the sum is less than or equal to 0.

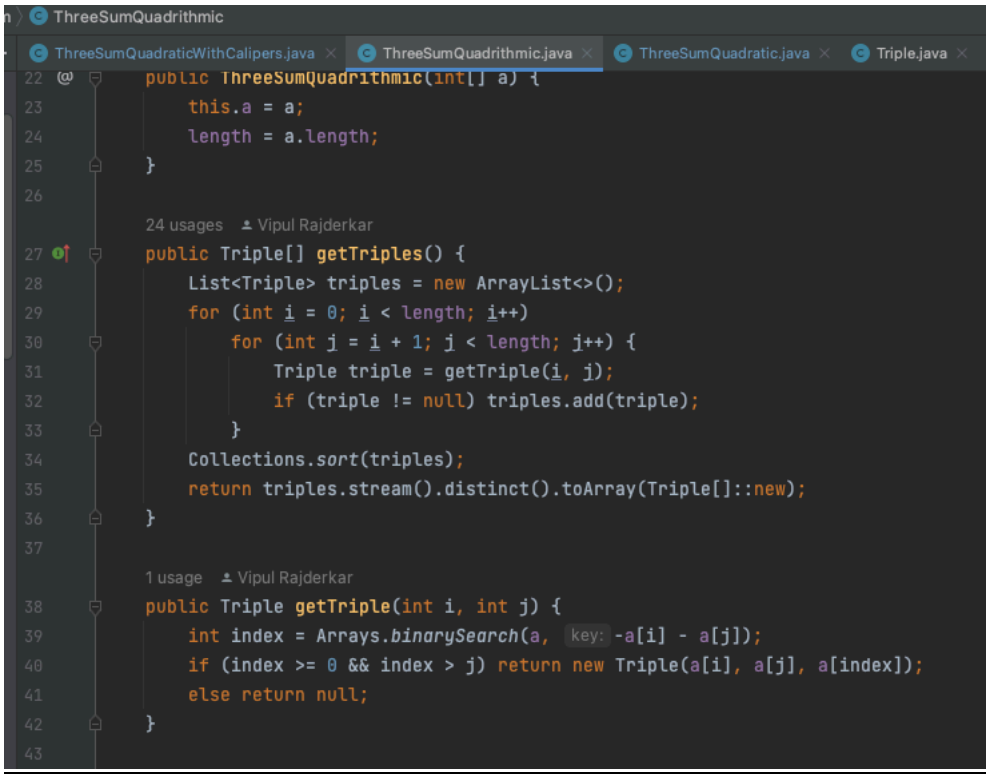
Code Snippet:



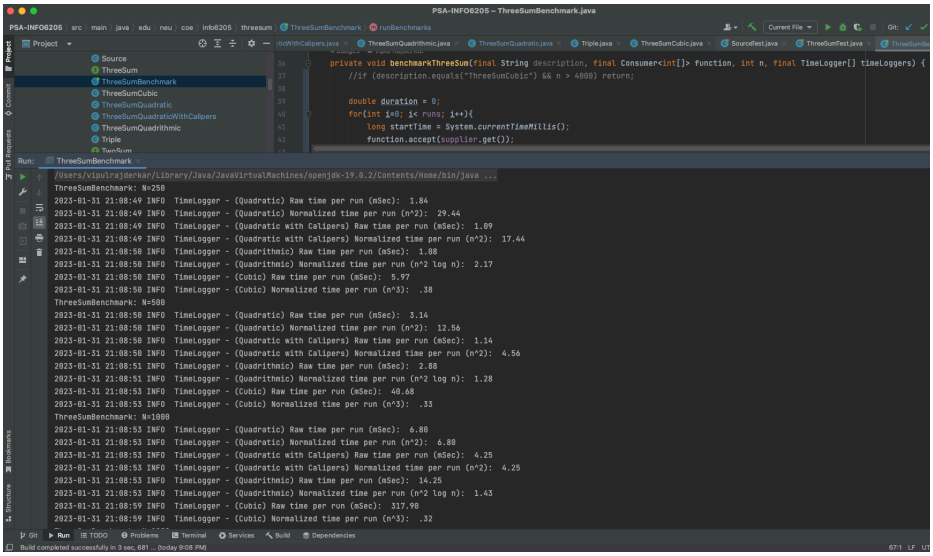
Approach 3: 3-Sum Quadrithimic

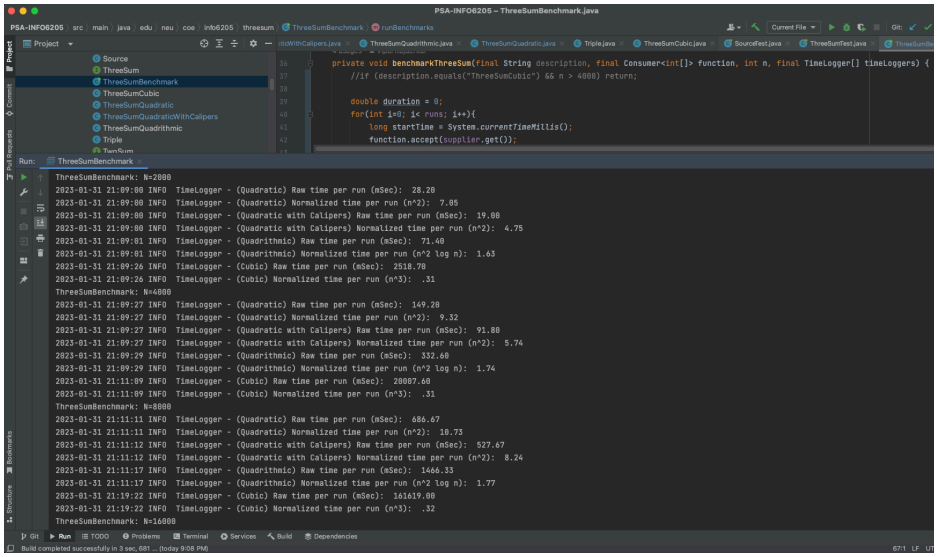
Step 1: Select a pair of number x and y (Complexity: $O(n^2)$)
Step 2: Find the third element(z = -(x+y)) using BinarySearch (Complexity: $O(\log n)$)

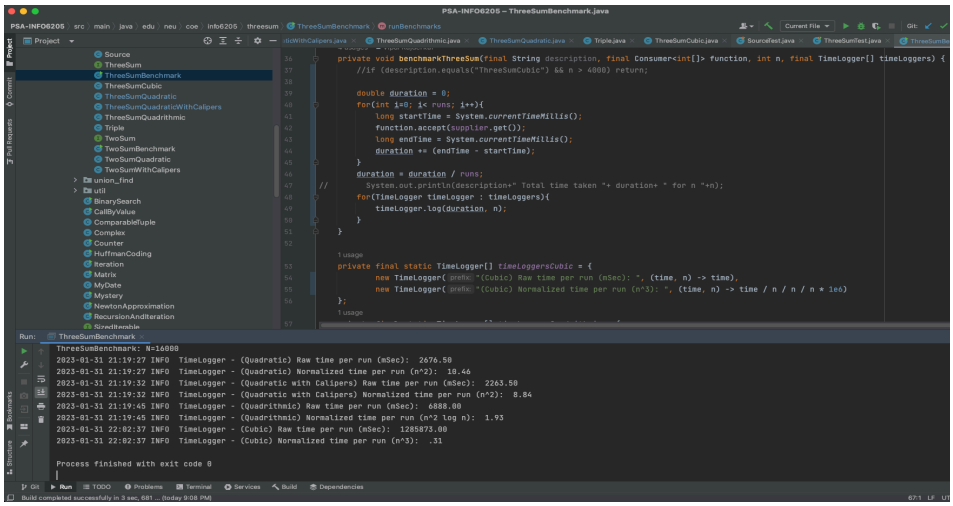
Code Snippet:



Benchmarking Code Snippet and Output:



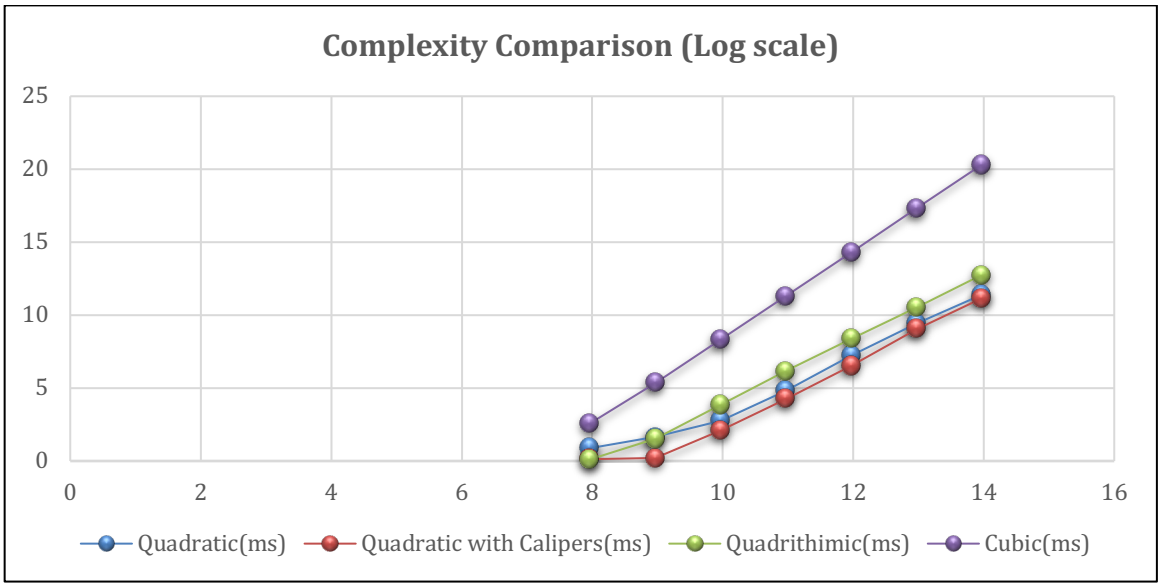

PSA-INFO6205 – ThreeSumBenchmark.java
private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLogger[] timeLoggers) {
//if (description.equals("ThreeSumCubic") && n > 4000) return;
double duration = 0;
for(int i=0; i< runs; i++){
long startTime = System.currentTimeMillis();
function.accept(supplier.get());
Run: ThreeSumBenchmark
ThreeSumBenchmark: N=2000
2023-01-31 21:09:00 INFO TimeLogger - (Quadratic) Raw time per run (mSec): 28.20
2023-01-31 21:09:00 INFO TimeLogger - (Quadratic) Normalized time per run (n^2): 7.05
2023-01-31 21:09:00 INFO TimeLogger - (Quadratic with Calipers) Raw time per run (mSec): 19.00
2023-01-31 21:09:00 INFO TimeLogger - (Quadratic with Calipers) Normalized time per run (n^2): 4.75
2023-01-31 21:09:01 INFO TimeLogger - (Quadrithmic) Raw time per run (mSec): 71.40
2023-01-31 21:09:01 INFO TimeLogger - (Quadrithmic) Normalized time per run (n^2 log n): 1.63
2023-01-31 21:09:26 INFO TimeLogger - (Cubic) Raw time per run (mSec): 2518.70
2023-01-31 21:09:26 INFO TimeLogger - (Cubic) Normalized time per run (n^3): .31
ThreeSumBenchmark: N=4000
2023-01-31 21:09:27 INFO TimeLogger - (Quadratic) Raw time per run (mSec): 149.20
2023-01-31 21:09:27 INFO TimeLogger - (Quadratic) Normalized time per run (n^2): 9.32
2023-01-31 21:09:27 INFO TimeLogger - (Quadratic with Calipers) Raw time per run (mSec): 91.80
2023-01-31 21:09:27 INFO TimeLogger - (Quadratic with Calipers) Normalized time per run (n^2): 5.74
2023-01-31 21:09:29 INFO TimeLogger - (Quadrithmic) Raw time per run (mSec): 332.60
2023-01-31 21:09:29 INFO TimeLogger - (Quadrithmic) Normalized time per run (n^2 log n): 1.74
2023-01-31 21:11:09 INFO TimeLogger - (Cubic) Raw time per run (mSec): 20087.60
2023-01-31 21:11:09 INFO TimeLogger - (Cubic) Normalized time per run (n^3): .31
ThreeSumBenchmark: N=8000
2023-01-31 21:11:11 INFO TimeLogger - (Quadratic) Raw time per run (mSec): 686.67
2023-01-31 21:11:11 INFO TimeLogger - (Quadratic) Normalized time per run (n^2): 10.73
2023-01-31 21:11:12 INFO TimeLogger - (Quadratic with Calipers) Raw time per run (mSec): 527.67
2023-01-31 21:11:12 INFO TimeLogger - (Quadratic with Calipers) Normalized time per run (n^2): 8.24
2023-01-31 21:11:17 INFO TimeLogger - (Quadrithmic) Raw time per run (mSec): 1466.33
2023-01-31 21:11:17 INFO TimeLogger - (Quadrithmic) Normalized time per run (n^2 log n): 1.77
2023-01-31 21:19:22 INFO TimeLogger - (Cubic) Raw time per run (mSec): 161619.00
2023-01-31 21:19:22 INFO TimeLogger - (Cubic) Normalized time per run (n^3): .32
ThreeSumBenchmark: N=16000
Build completed successfully in 3 sec, 681 ... (today 9:08 PM)


PSA-INFO6205 – ThreeSumBenchmark.java
private void benchmarkThreeSum(final String description, final Consumer<int[]> function, int n, final TimeLogger[] timeLoggers) {
//if (description.equals("ThreeSumCubic") && n > 4000) return;
double duration = 0;
for(int i=0; i< runs; i++){
long startTime = System.currentTimeMillis();
function.accept(supplier.get());
long endTime = System.currentTimeMillis();
duration += (endTime - startTime);
}
duration = duration / runs;
// System.out.println(description+" Total time taken "+ duration+ " for n "+n);
for(TimeLogger timeLogger : timeLoggers){
timeLogger.log(duration, n);
}
}
1 usage
private final static TimeLogger[] timeLoggersCubic = {
new TimeLogger(prefix: "(Cubic) Raw time per run (mSec): ", (time, n) -> time),
new TimeLogger(prefix: "(Cubic) Normalized time per run (n^3): ", (time, n) -> time / n / n / n * 1e6)
};
1 usage
Run: ThreeSumBenchmark
ThreeSumBenchmark: N=16000
2023-01-31 21:19:27 INFO TimeLogger - (Quadratic) Raw time per run (mSec): 2676.50
2023-01-31 21:19:27 INFO TimeLogger - (Quadratic) Normalized time per run (n^2): 10.46
2023-01-31 21:19:32 INFO TimeLogger - (Quadratic with Calipers) Raw time per run (mSec): 2263.50
2023-01-31 21:19:32 INFO TimeLogger - (Quadratic with Calipers) Normalized time per run (n^2): 8.84
2023-01-31 21:19:45 INFO TimeLogger - (Quadrithmic) Raw time per run (mSec): 6888.00
2023-01-31 21:19:45 INFO TimeLogger - (Quadrithmic) Normalized time per run (n^2 log n): 1.93
2023-01-31 22:02:37 INFO TimeLogger - (Cubic) Raw time per run (mSec): 1285873.00
2023-01-31 22:02:37 INFO TimeLogger - (Cubic) Normalized time per run (n^3): .31
Process finished with exit code 0
Build completed successfully in 3 sec, 681 ... (today 9:08 PM)

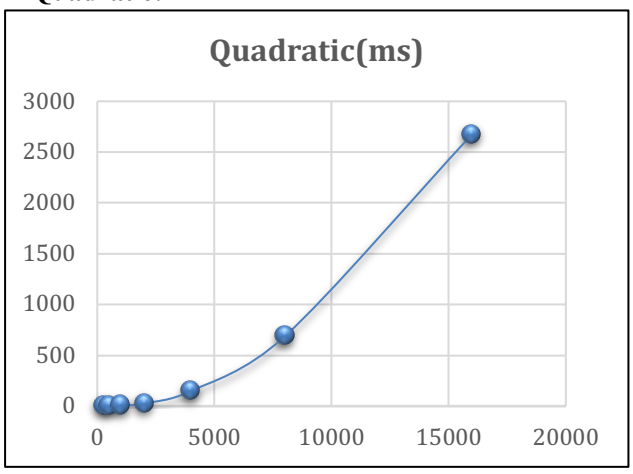
Graphical Representation:

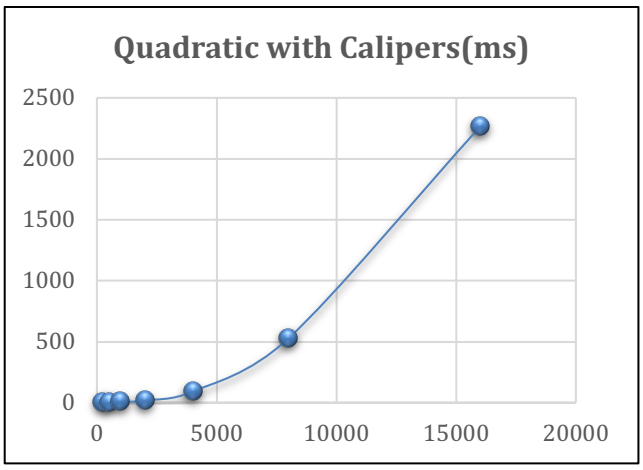
Timing Table:

N	Quadratic(ms)	Quadratic with Calipers(ms)	Quadrithimic(ms)	Cubic(ms)
250	1.84	1.09	1.08	5.97
500	3.14	1.14	2.88	40.68
1000	6.8	4.25	14.25	317.9
2000	28.2	19	71.4	2518.7
4000	149.2	91.8	332.6	20007.6
8000	686.67	527.67	1466.33	161619
16000	2676.5	2263.5	6888	1285873

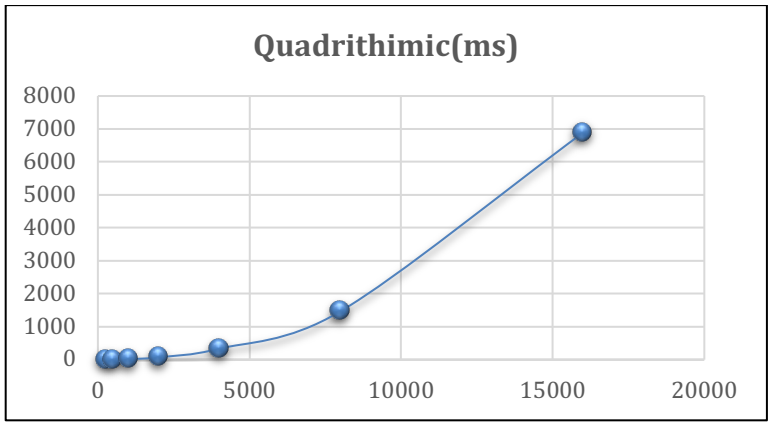


3-Sum Quadratic:

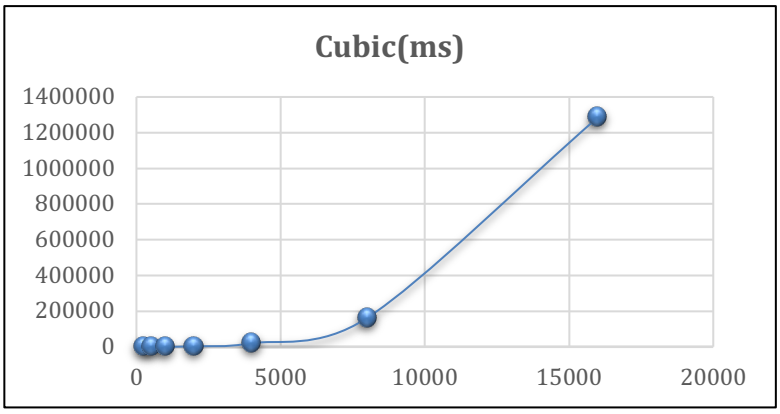




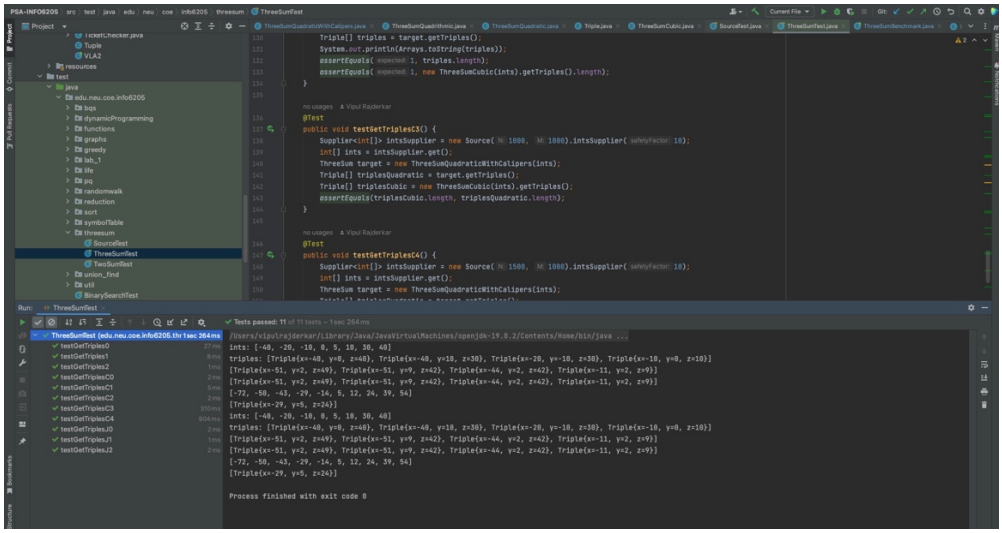
3-Sum Quadrithimic:



Cubic:



Unit Test Screenshots:



Explanation of why Quadratics work:

Quadratic approach performs definitely better than the rest as it is most efficient approach with a time complexity of $O(N^2)$. Since we use a two-pointer approach, the computation time is reduced as depending on the difference between expected and actual sum only required pointers can be moved.