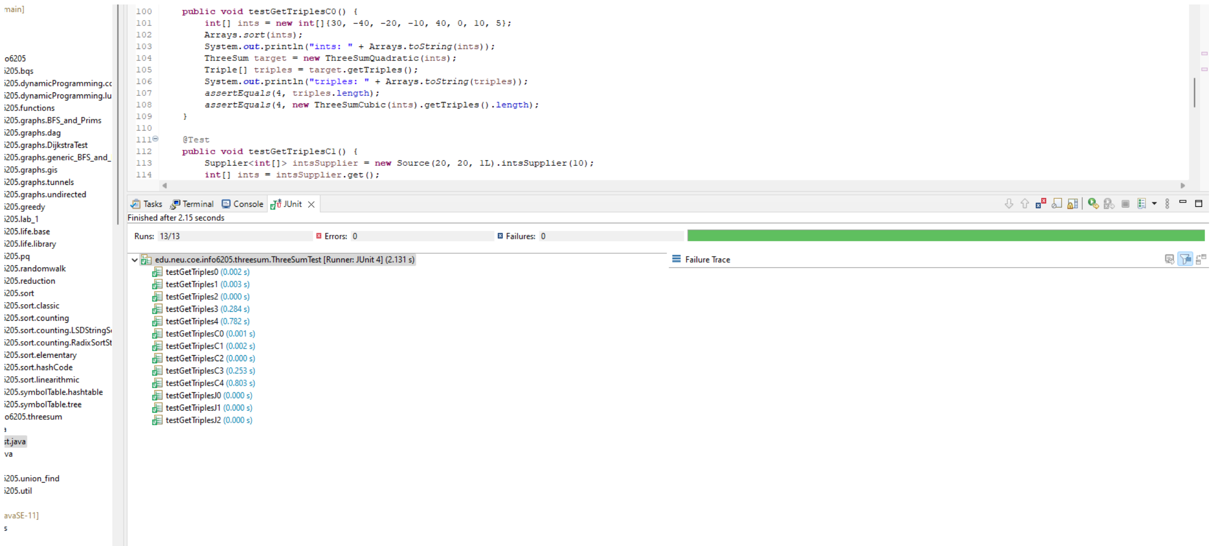
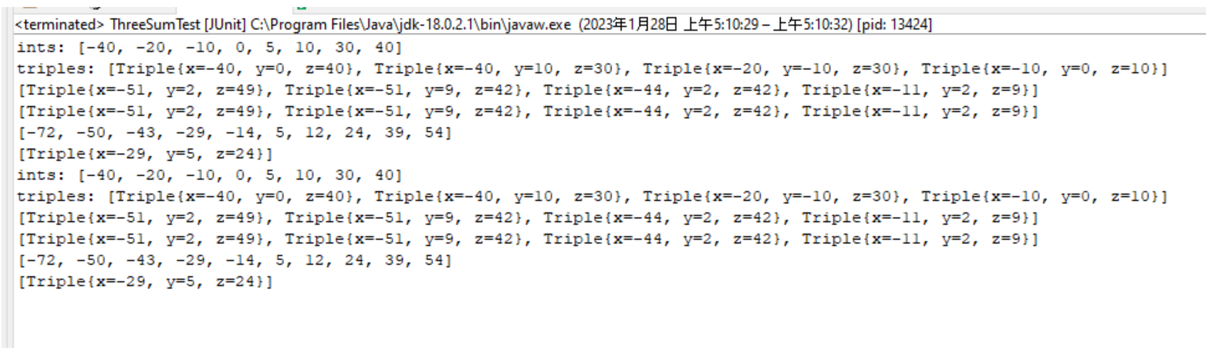
INFO6205 Assignment 2 threesum questions

Ruizhe Zeng

1. Unit Test Result with Calipers method





```
<terminated> ThreeSumTest [JUnit] C:\Program Files\Java\jdk-18.0.2.1\bin\javaw.exe (2023年1月28日 上午5:10:29 – 上午5:10:32) [pid: 13424]
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}]
```

2. Data observation

a. Data from program output (part of)
b. Spreadsheet for data
c. Graph

3. threesum question explanation

a. ThreeSumQuadratic

Imagine in an **SORTED** array, each time we pick one of the elements from the array and call it **Mid**. To get all the elements picked it requires O(n). Now we choose and element at its left side and call it **Left**, and then pick a element at Mid's right side and call it **Right.** Notice Left,Mid,Right should be in range of array all the time. After that we calculate the **Sum** = Mid+Left +Left+Right. If Sum equals 0. We find what we want, and move Left to the left for one index. If Sum < 0, then we know we cannot find any Sum = 0 if we continually move Left to left, it will only get smaller. So we only need to move

Right to the right in one index. And check for the Sum again. If Sum > 0, we know we cannot get any result if we move Right to the right, because it will only get larger. So we only need to move Left to the left for one index and check for Sum. Notice it may have more than one solution when we check for the Mid. That is why we need to move Left to the left in one index,when we find a Sum = 0. It will stop when Left or Right is no longer in the range of array. Therefore we ignore many invalid candidates, to do this for each Mid we need up to O(n) times. So it requires O(n^2) time.

b. **ThreeSumQuadraticWithCalipers**

This method is very similar to the previous one. This time we going to pick **Left** instead of Mid from array each time, so it also requires O(n) time to pick all the candidates. Then we are going to pick the element at Left's right side and call it **Mid**. After that we are going to pick the **Last** element in the array and call it **Right**. We want to make sure Left < Mid < Right and they are in the range of the array all the time. If Sum = Left + Mid + Right = 0 , we find the result and move Left to its right for one index. If Sum > 0, we know since Left is the smallest number we can get, we need to move Right to the left .If Sum < 0, since the Right is the largest number we can get, we need to move Mid to the right. This process several time until the statement Left< Mid<Right is no longer true or any of them is not in the range of array. To check all the Mid and Right it takes up to O(n) for each Mid. So it is also O(n^2), same as the previous method. However for each Mid it will only check at most (N - Mid) elements, but the previous method is going to check N elements if it cannot find any solution, so this method is more unlikely to get to the worst case, so technically it will be a little faster than the previous method.