-------------------------------------------------------------------  
Name: Yash Pawar  
NU ID: 002747371  
Github: github.com/NeuYash/PSAAssignments  
Email: [pawar.ya@northeastern.edu](mailto:pawar.ya@northeastern.edu)-------------------------------------------------------------------  
Attachments :   
A]Code :  
 1. ThreeSumQuadratic.java  
 2. ThreeSumQuadraticwithCalipers.java  
 3. TwoSumQuadraticWithCalipers.java  
 4. ThreeSumBenchMark.java  
 5. TwoSumBenchMark.java   
 6. Timer.java  
  
B] TestCases :  
 1.ThreeSumTest.java  
 2. TwoSumTest.java  
  
C] Images:  
 TestCases running and benchmark results.  
 Graph for doubling N Values.   
  
  
  
  
  
  
  
  
Three Sum Problem:   
  
Find triplets from an array which sum to a given target in an efficient way.   
  
There are 3 ways of going about it.  
1] Cubic Solution (O(N^3))  
1] Quadrathmic Solution (O((N^2)logn))  
1] Quadratic Solution (O(N^2))  
  
  
Quadrathmic Solution :   
1] We use sorted array for this approach.  
2] Here we use 2 for loops to find first two elements and subtract its sum from zero to know the 3rd element in triplet.   
3] Now we use Binary Search(logn) to search 3rd element in the array.   
  
Quadratic Solution :   
1] We use Sorted Array for this approach.  
2] Here we use 2 pointers low and high after looping through the loop getting our first element and trying to find next 2 elements which sum to target.  
3] As we have a sorted array, it is very easy to control these two pointers based on if the sum is more or less than target.   
  
**Bonus**:   
  
QuadraticWithCalipers Solution :   
  
1] We use Sorted Array for This approach.  
2] here we use similar 2 pointers but less number of iterations as we are given a middle element and now we need to find the low and high which takes less number of comparisons as we are not trying to find mid and high but low and high pair.   
3] It is more efficient than Quadratic Solution.  
4] Please see BenchMark Scores for better Understanding.  
  
Here is a Table that. Will better explain this point.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N |  | Quadratic WCaliper | Quadratic | Quadrathmic | Cubic |
|  |  | (millisecs) | (millisecs) | (millisecs) | (millisecs) |
| 125 | Raw Time | 0.12 | 0.06 | 0.15 | 0.38 |
|  | Nrm Time | 7.52 | 4.02 | 1.37 | 0.20 |
| 250 | Raw Time | 0.08 | 0.17 | 0.5 | 2.75 |
|  | Nrm Time | 1.33 | 2.75 | 1.0 | 0.18 |
| 500 | Raw Time | 0.49 | 0.69 | 2.29 | 21.40 |
|  | Nrm Time | 1.95 | 2.77 | 1.02 | 0.17 |
| 1000 | Raw Time | 2.09 | 2.66 | 11.89 | 167.87 |
|  | Nrm Time | 2.09 | 2.66 | 1.19 | 0.17 |
| 2000 | Raw Time | 8.99 | 11.39 | 55.67 | 1336.6 |
|  | Nrm Time | 2.22 | 2.85 | 1.27 | 0.17 |
| 4000 | Raw Time | 64.33 | 68.07 | 298.53 | 10589 |
|  | Nrm Time | 4.02 | 4.25 | 1.56 | 0.17 |
| 8000 | Raw Time | 396.47 | 405.24 | 1453 | - |
|  | Nrm Time | 6.66 | 6.33 | 1.75 | - |
| 16000 | Raw Time | 1187.83 | 2221.08 | 5675.63 | - |
|  | Nrm Time | 7.37 | 8.68 | 1.59 | - |

Similar results for TwoSum with QuadraticCaliper and Quadratic Approach.  
  
Proofs for TwoSum will be found in Attached Benchmark images.  
  
  
Why does Quadratic Solution Work?  
We employ two pointers to iterate through the loop and aid in the discovery of potential triples that add up to the target. Additionally, finding the third member in the array that will form a triplet doesn't require logn Binary search, therefore our technique can solve any three-sum problem in O(n2) time.  
int low=0, high=a.length-1;

while(low<j && high>j) {

int currentsum=a[low]+a[j]+a[high];

if(currentsum==0) {

triples.add(new Triple(a[low], a[j], a[high]));

low++;

high--;

}else if(currentsum<0) {

low++;

}else {

high--;

}

}//while

Since we already have the center element, the quadratic solution utilizing Calipers' method is superior. We need now look for the first and third elements to complete the triplet. Since there are fewer array accesses and comparisons in this case, the solution runs faster than a typical quadratic one.  
  
  
  
  
  
  
public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) {

List<Triple> triples = new ArrayList<>();

// **FIXME** : use function to qualify triples and to navigate otherwise.

int low = i + 1;

int high = a.length - 1;

while (low < high) {

Triple t1 = new Triple(a[i], a[low], a[high]);

int currentsum = function.apply(t1);

if (currentsum == 0) {

triples.add(new Triple(a[i], a[low], a[high]));

low++;

high--;

} else if (currentsum < 0) {

low++;

} else {

high--;

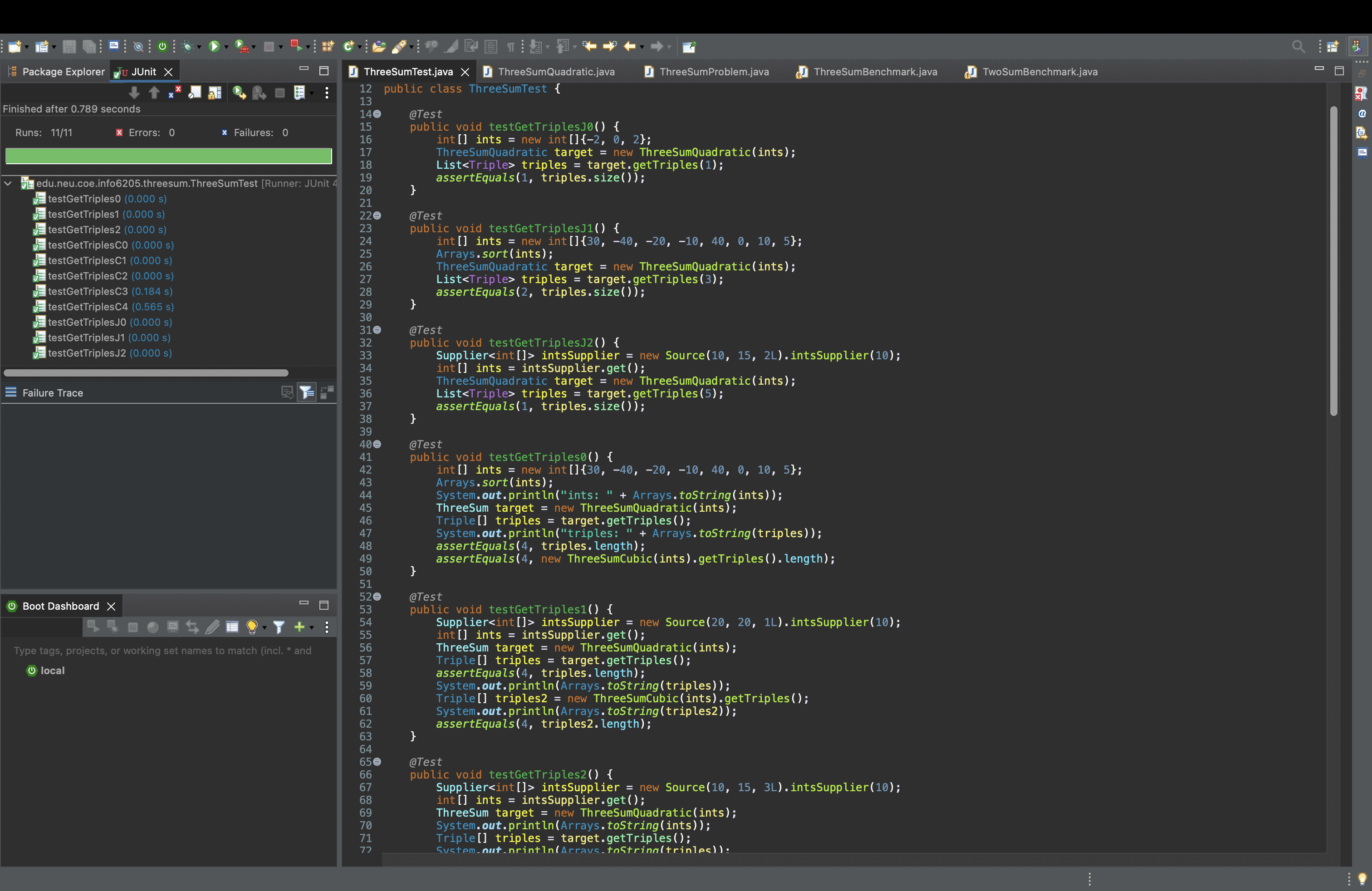
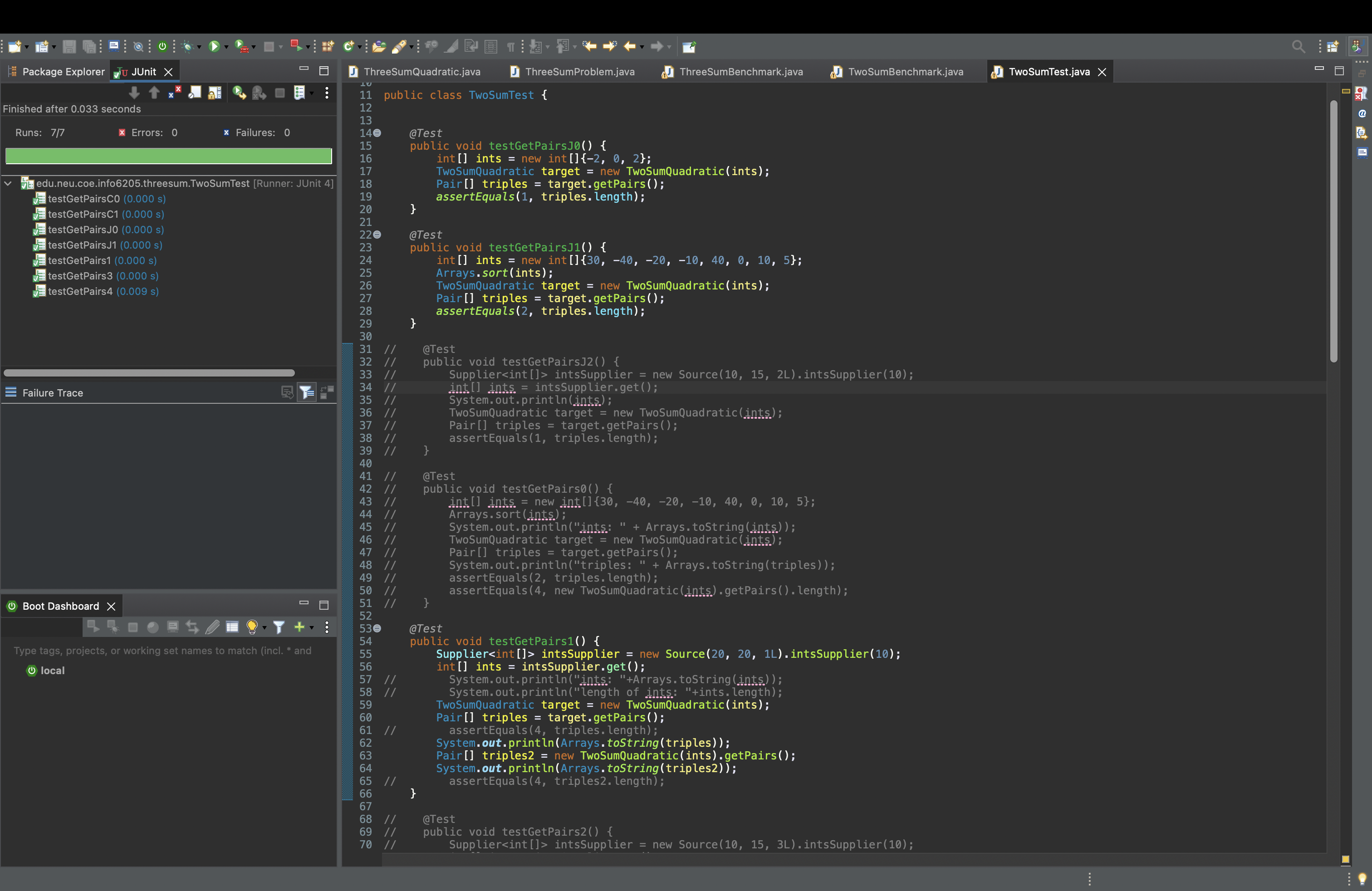
}

} // while

// END

return triples;

}

All the test cases for ThreeSum and Two sum problems are running.  
  
  
  
  
  
  
The below displayed graph will explain the mentioned benchmarking scenarios.

