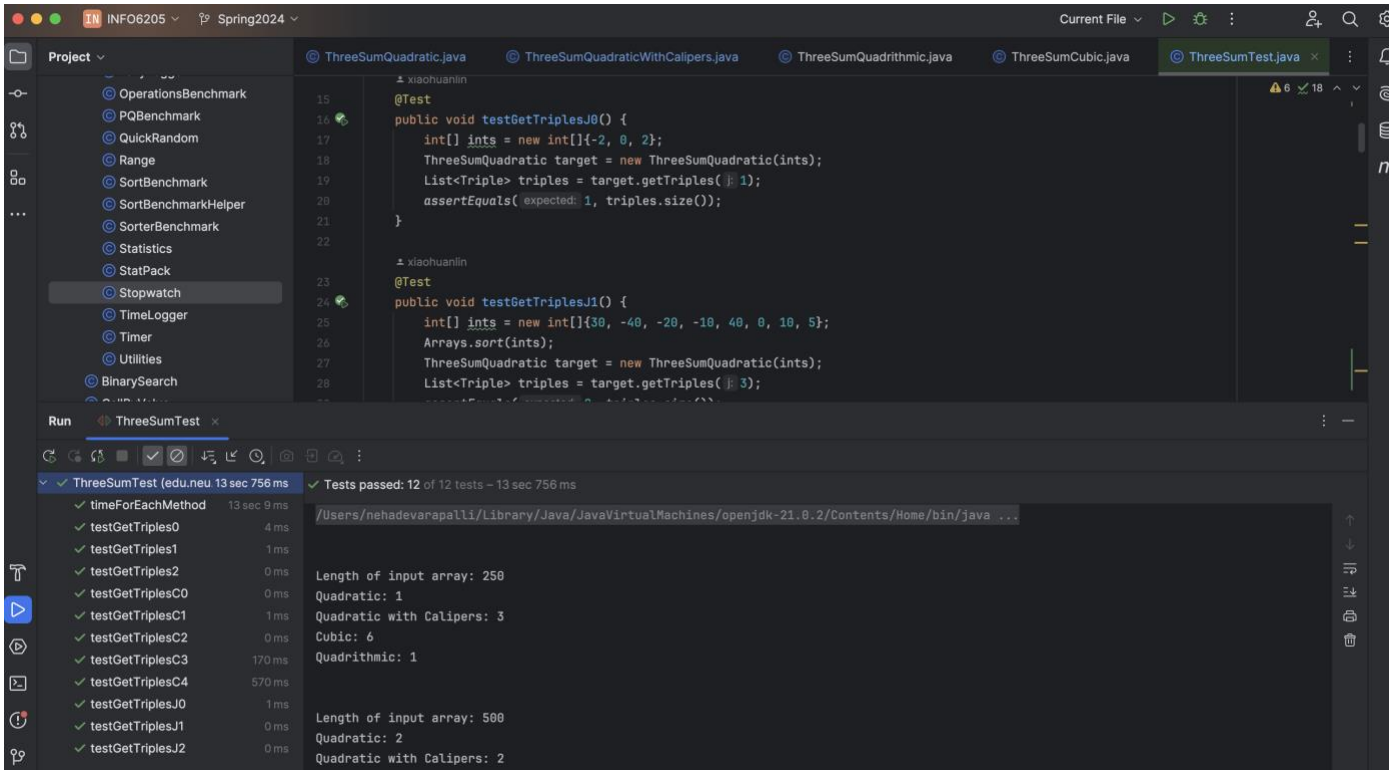
INFO 6205 - Program Structures and Algorithms
Spring 2024

NAME: Neha Devarapalli
NUID: 002883137
GITHUB LINK: https://github.com/nehadevarapalli/INFO6205

Task: Assignment 2 (3-SUM)

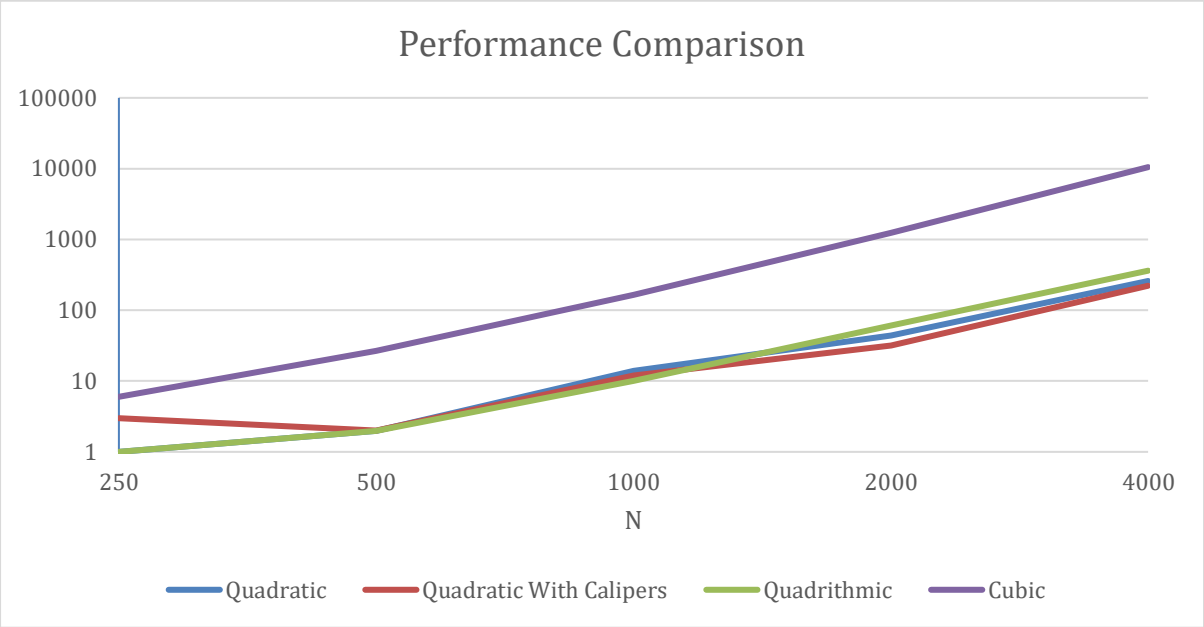
Unit Test Screenshots:
Passed all the unit tests successfully and added a custom test to measure the time taken by the Quadratic, Quadratic with calipers, Quadrithmic and Cubic methods and compared them while using the doubling method to double the length of the input arrays.



Following were the timing observations found after measuring the run-time for each approach in milliseconds using the Stopwatch class in the repository:

N	RAW TIME (milliseconds)			
	Quadratic	Quadratic With Calipers	Quadrithmic	Cubic
250	1	3	1	6
500	2	2	2	27
1000	14	12	10	165
2000	44	32	61	1246
4000	259	222	362	10502

Also, plotted a graph to better visualize the data.



As we can see, the cubic ($O(n^3)$) approach to solving the 3-SUM problem is taking a significantly much greater amount of time than any of the other approaches.

Why the quadratic method(s) work?

1. **Quadratic Method ($O(n^2)$)**
   The primary idea behind this algorithm is to iterate through each possible middle index ‘j’ (middle element in three sum) and then use the two pointers ‘i’ and ‘k’ (first and the last elements in three sum) to explore the solution space by expanding outwards from the middle.
   - The outer loop iterates over each index ‘j’ in the input array ‘a’.
   - The inner loop uses ‘i’ and ‘k’ to traverse the array from both ends starting from the middle.
   - Then we check the sum of the elements at the indices ‘i’, ‘j’ and ‘k’ using the ‘Triple’ class.
   - If the sum is zero, we add it to the ‘triples’ list.
   - If the sum is less than zero then, the pointer ‘i’ is moved to the right to make the sum greater. (since it is a sorted array)
   - If the sum is greater than zero, then the pointer ‘k’ is moved to the left to make the sum smaller. (since it is a sorted array)
   - Once the loop ends, the method returns a list (‘triples’) of ‘Triple’ objects representing unique triplets that sum to zero.
   - After we reach the outer loop, the duplicates are removed before being converted to an array and returned.

The time complexity of this algorithm is $O(N^2)$ because for each 'j' the inner loop runs in linear time.

2. **Quadratic with Calipers Method ($O(N^2)$)**

   This method implements the use of calipers, which are used to optimize the process of finding valid triplets. Calipers are used to navigate through the solution space more efficiently.

   - The outer loop iterates over each index 'i' in the sorted input array up to the second last element (i = 0 to i = length-2) because there must be at least two elements following a[i] to form a triplet.
   - In the inner loop, the calipers method is used to find triples with the fixed index 'i'.
   - The 'calipers' method takes three parameters: the input array 'a', the index 'i' and a function that computes a value to be compared with zero. In this case, the function is 'Triple::sum'.
   - The calipers, 'left' and 'right' navigate through the array, exploring potential triples and adding them to the result ('triples' list) if the sum is zero.
   - If the sum is less than zero, left caliper is moved ahead. Else, the right caliper is moved behind.
   - After exiting the inner loop, the list is sorted to facilitate duplicate removal and then the distinct operation is applied to ensure only unique triplets are included.
   - At last, an array of unique 'Triple' objects is returned representing triplets that sum to zero.

   The time complexity of this algorithms is $O(N^2)$.

3. **Quadrithmic Method ($O(N^2 \log N)$)**

   This method works by using the binary search algorithm to efficiently find the third element in a triplet for a given pair of elements.

   - The 'getTriples' method iterates through each pair of elements in the array ('i' and 'j').
   - For each pair, it calls the 'getTriple' method to find the third element using binary search.
   - The target element for the third position is '-a[i] – a[j]' (negative sum of the first two elements).
   - If the index is found and is greater than 'j' (to ensure that we are finding 'k') , a 'Triple' object is created and returned.
   - The list of triples is sorted, and duplicates are removed to ensure that the final result contains unique triplets.

   The binary search operation takes logarithmic time, and it is executed for each pair of elements in a nested loop. So, the overall time complexity is $O(N^2 \log N)$.