Program Structures and Algorithms

ASSIGNMENT-2

1.1 SOLVING THREE-SUM

For this assignment, my approach for solving the three-sum is by using the 2-pointer method.

Quadratic Calliper Approach:

Considering three pointers **i**,**j**,**k** for the array **arr[]**, assuming one of the terms is already given(**j**), I have taken 2 pointers, one at the start(**i**) and the other at the end(**k**).

As per the question arr[i] + arr[j] + arr[k] = 0.

So my pseudo code involves

```
If(arr[i] + arr[j] + arr[k] == 0){
        Add triplet to result
}
Else if(arr[i] + arr[j] + arr[k] > 0){
        Decrease k by one      (as the array is already sorted arr[k-1] < arr[k])
}
Else{
        Increase i by 1.  (as arr[i+1]> arr[i] and it can bring some more comparisions)
}
```

The breaking case for this pseudo code is when (i!<j) && (j!<k).

Quadratic Approach:

Considering three pointers **i**,**j**,**k** for the array **arr[]**, assuming one of the terms is considered given(**i**), I have taken 2 pointers, one at the start(**j**) after index **i** and the other at the end(**k**).

As per the question arr[i] + arr[j] + arr[k] = 0.

So my pseudo code involves

```
If(arr[i] + arr[j] + arr[k] == 0){
        Add triplet to result
```

}

Else if(arr[i] + arr[j] + arr[k] > 0){

Decrease k by one (as the array is already sorted arr[k-1] < arr[k])

}

Else{

Increase j by 1. (as arr[j+1]> arr[j] and it can bring some more comparisons)

}

The breaking case for this pseudo code is when (j!<k).

1.2 CODE

Here is the code for the quadratic calliper approach.

```
public static List<Triple> calipers(int[] a, int i, Function<Triple,
Integer> function) {
    List<Triple> triples = new ArrayList<>();
    int j=i+1,k=a.length-1;
    while(j<k){
        if(a[j] + a[k] +a[i] ==0){
            triples.add(new Triple(a[i],a[j],a[k]));
            j++;
            k--;
        }
        else if(a[j] + a[k] +a[i] >0){
            k--;
        }
        else
            j++;
    }
    return triples;
}
```

Here is the code for the quadratic approach

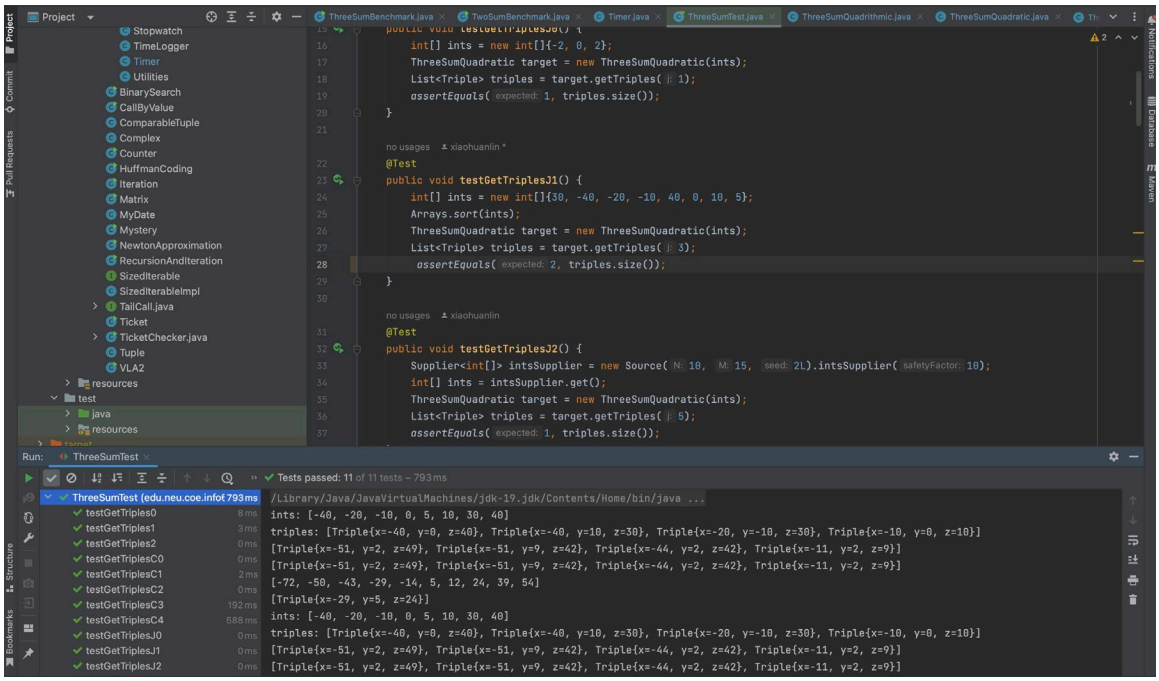
```
public List<Triple> getTriples(int j) {
    List<Triple> triples = new ArrayList<>();
    // FIXME : for each candidate, test if a[i] + a[j] + a[k] = 0.
    // END
    int i=0,k=a.length-1;
    while((i<j) && (j<k)){
```

```
        if(a[j] + a[k] +a[i] ==0){
            triples.add(new Triple(a[i],a[j],a[k]));
            i++;
            k--;


        }
        else if(a[j] + a[k] +a[i] >0){
            k--;
        }
        else
            i++;
    }
    return triples;
}
```

1.3 UNIT-TESTING

Here is the unit testing for the all the conditions provided. As you can see that all the conditions have passed successfully.

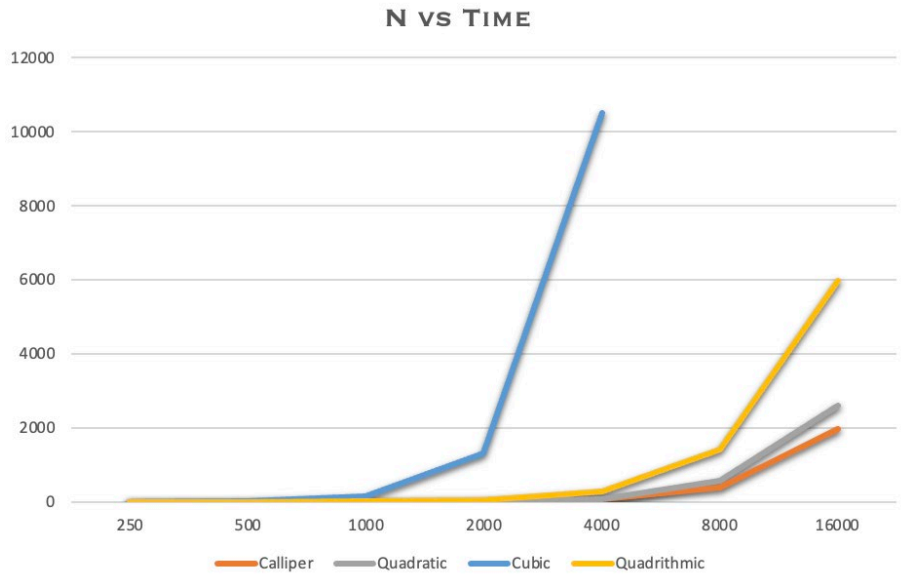


1.3 STATISTICS

Here are the staticts for the N and time graph. The time is calculated from completing the time function from the code.

N	Calliper	Quadratic	Quadrithmic	Cubic
250	0.27	0.24	0.53	2.75
500	0.54	0.77	1.99	20.9
1000	2.32	2.97	10.92	162.63
2000	9.73	12.31	59.86	1313.83
4000	84.09	73.17	295.85	10507.89
8000	399.52	576.76	1417.06	
16000	1971.05	2601.26	5970.75	

Here is the graph that shows exactly how the time differs for all the equations.



1.4 CONCLUSION

The complexity for quadratic is $\mathbf{O}(n^2)$ as the 2 pointer approach only has a complexity of $\mathbf{O}(n)$ as the array is traversed only once by the pointers. And for n traversals for the selected mid-point the total complexity now becomes

$$\mathbf{O}(n) \ . \mathbf{O}(n)$$

$$\therefore \ \textbf{Complexity} = \mathbf{O}(n^2)$$

If we even consider the case of quadratimic the complexity becomes

$$\mathbf{O}(n^2) \log n$$

Here the logn extra complexity becomes when we implement binary search. It might look simple in the early case but from the statics in the above graph, the complexity changes if the value of n increases and the changes become even more significant wrt. time.