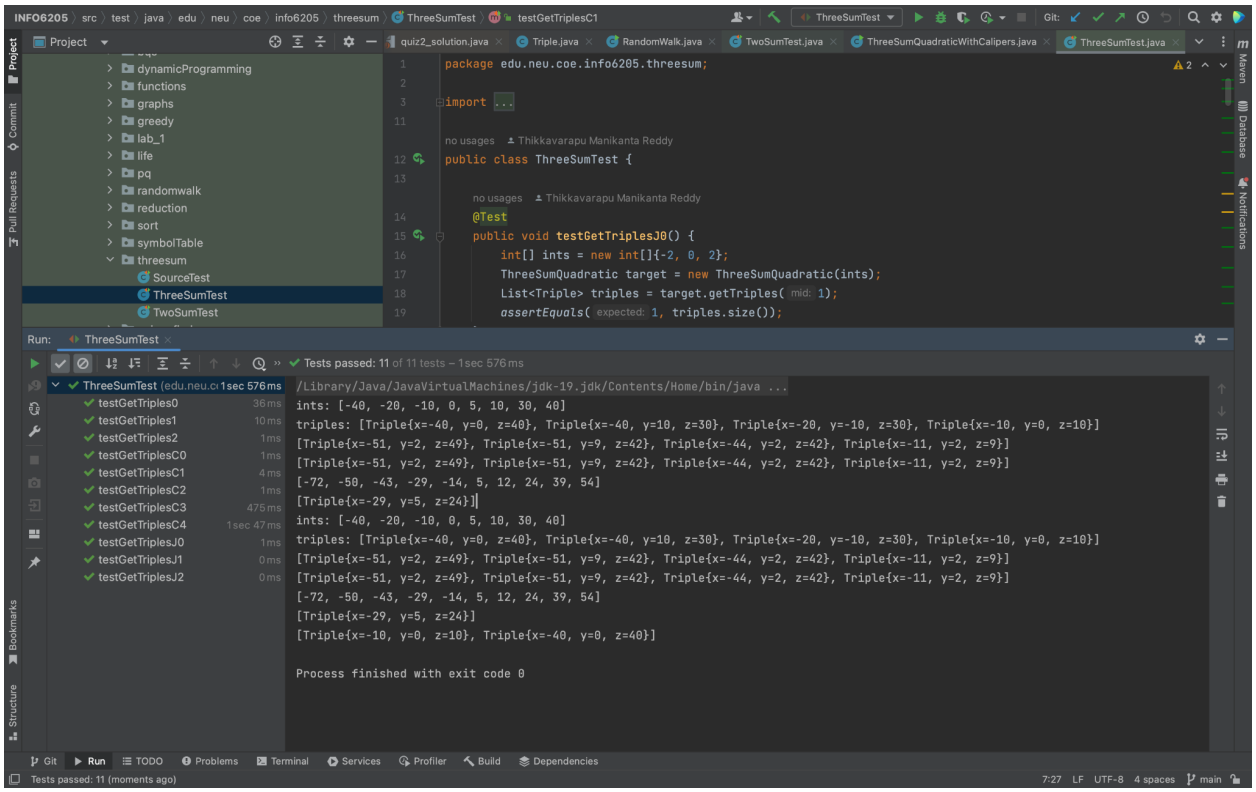
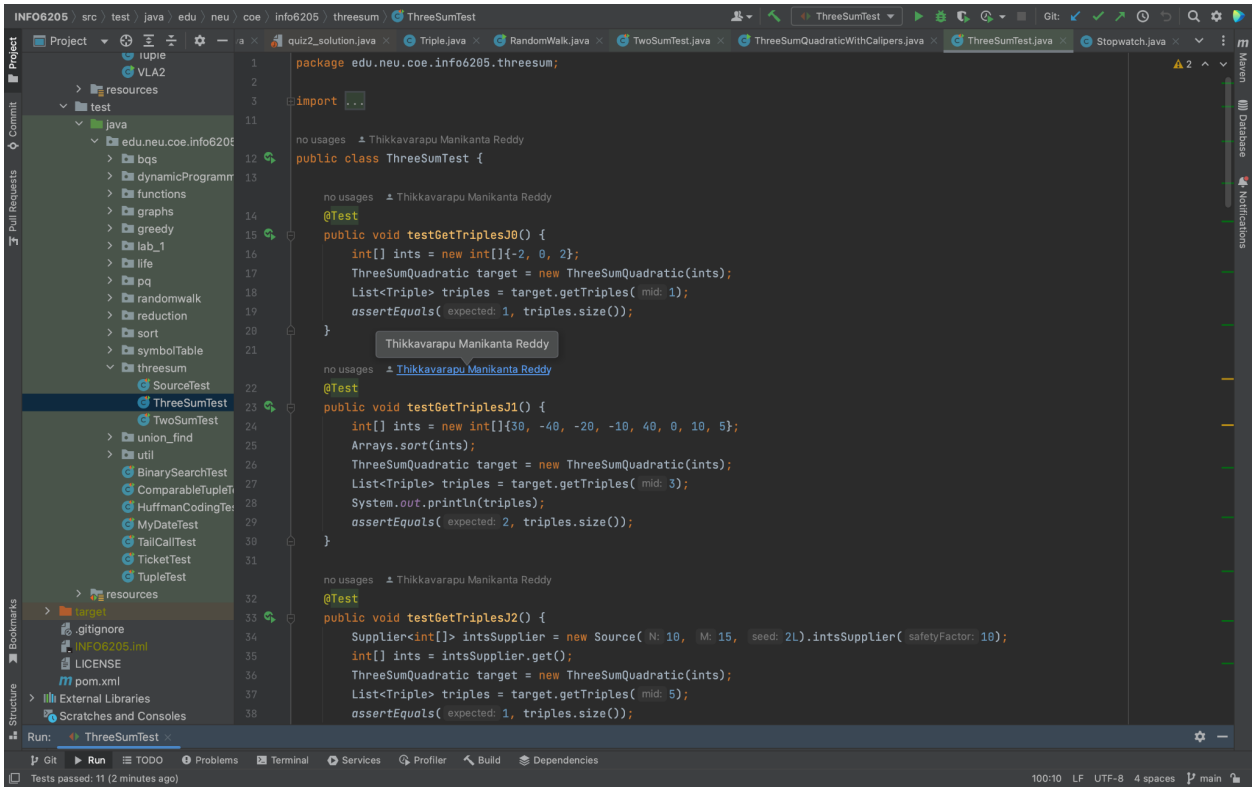
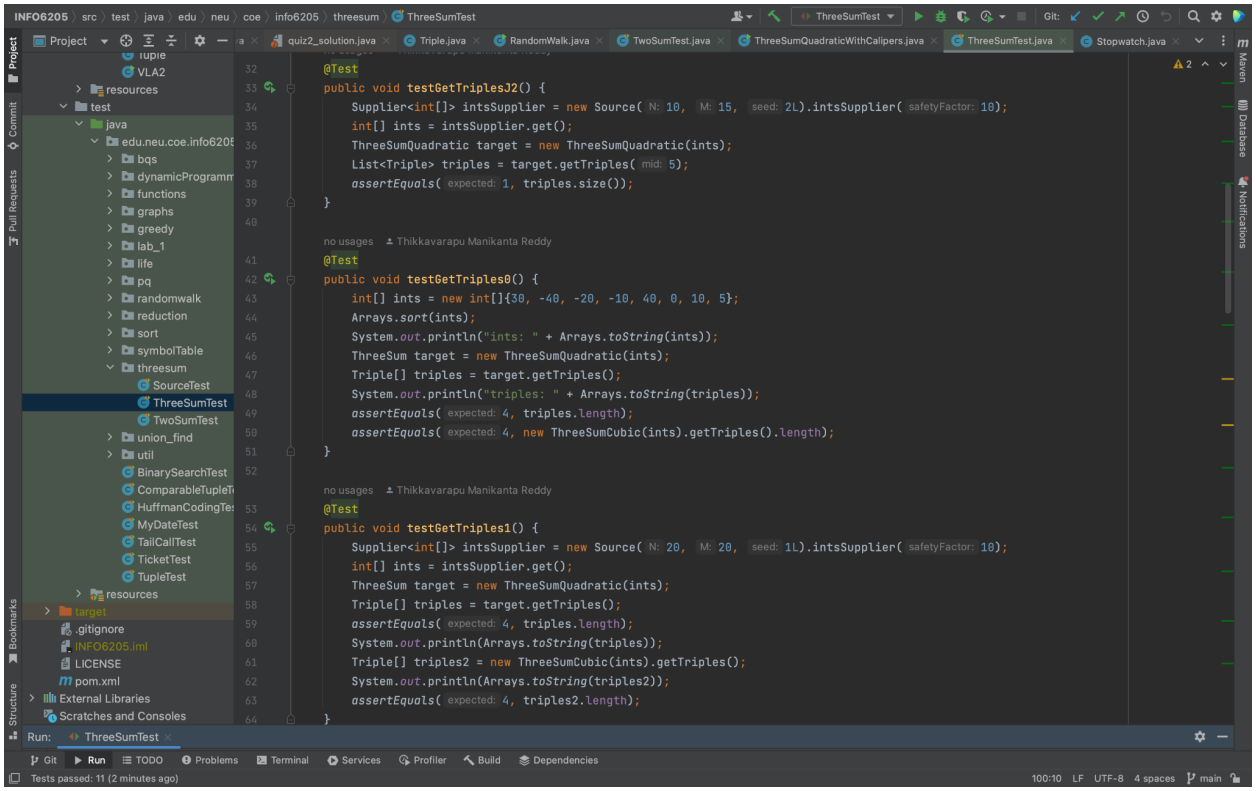
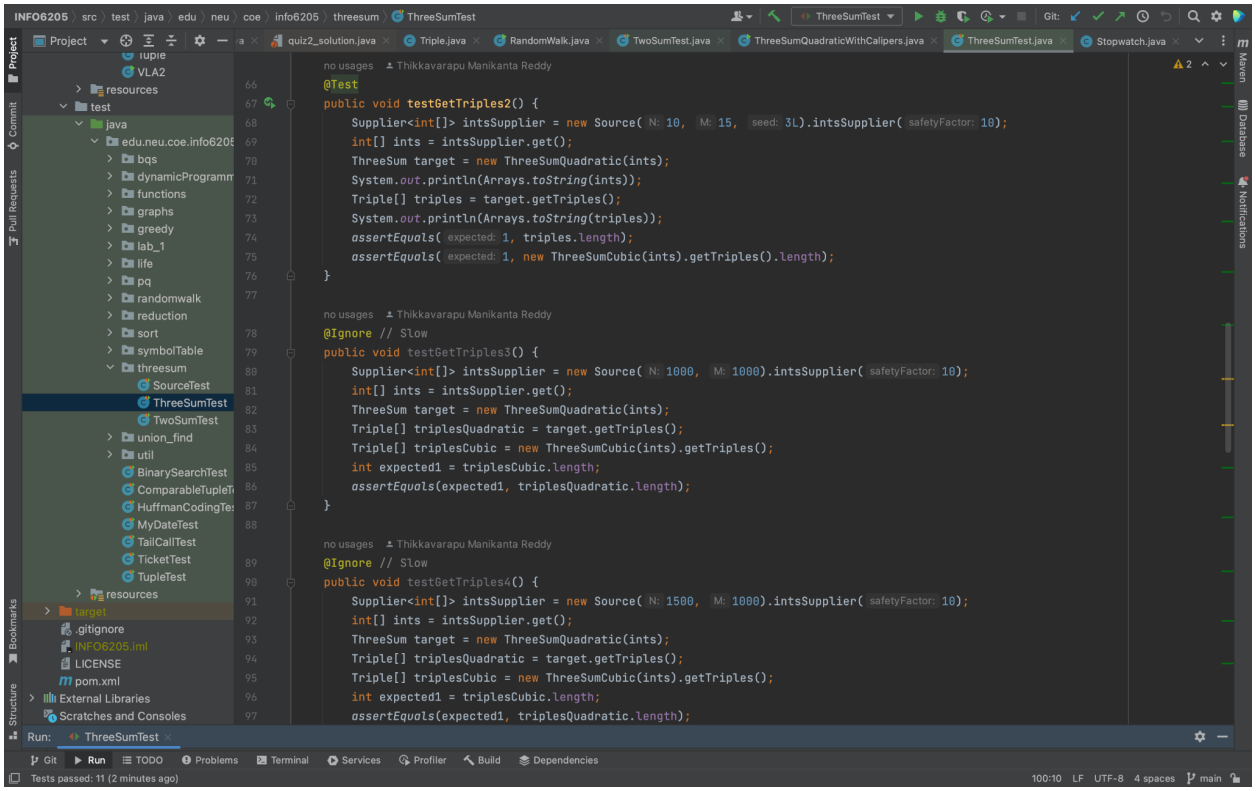
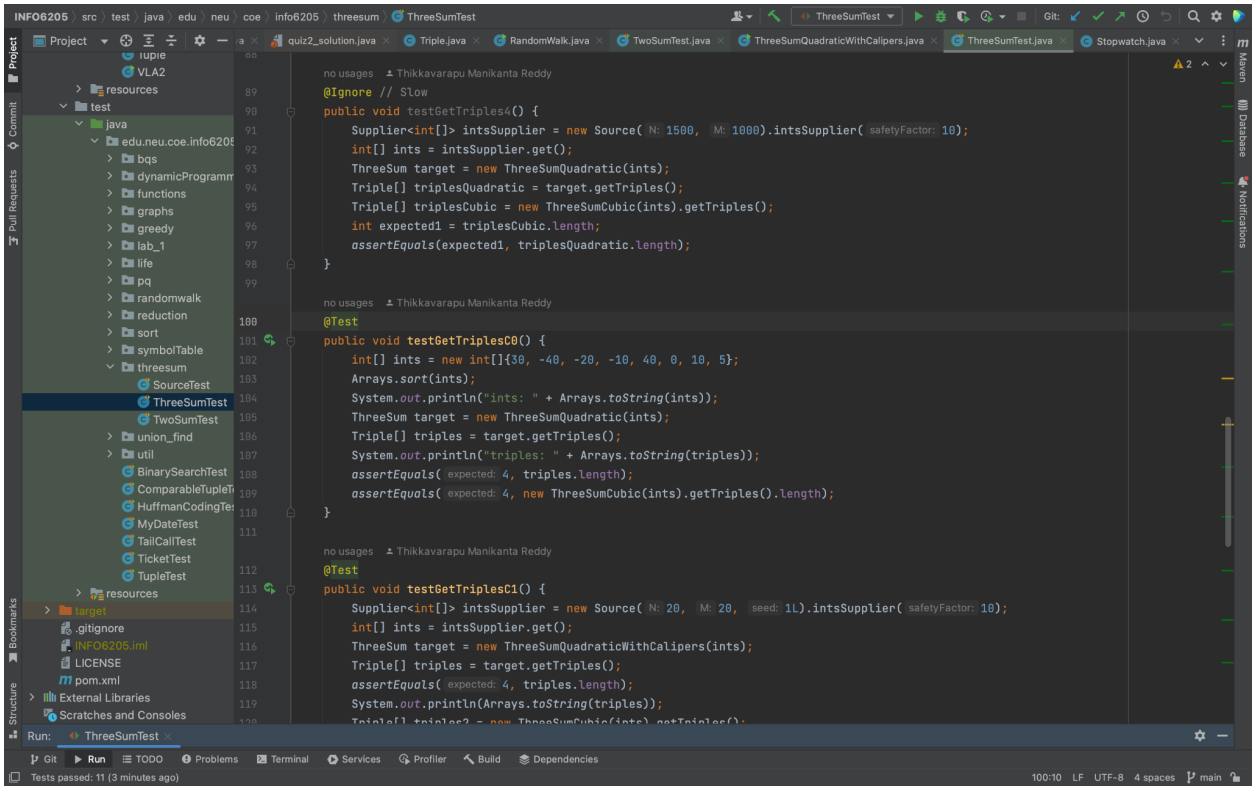
Screenshots:



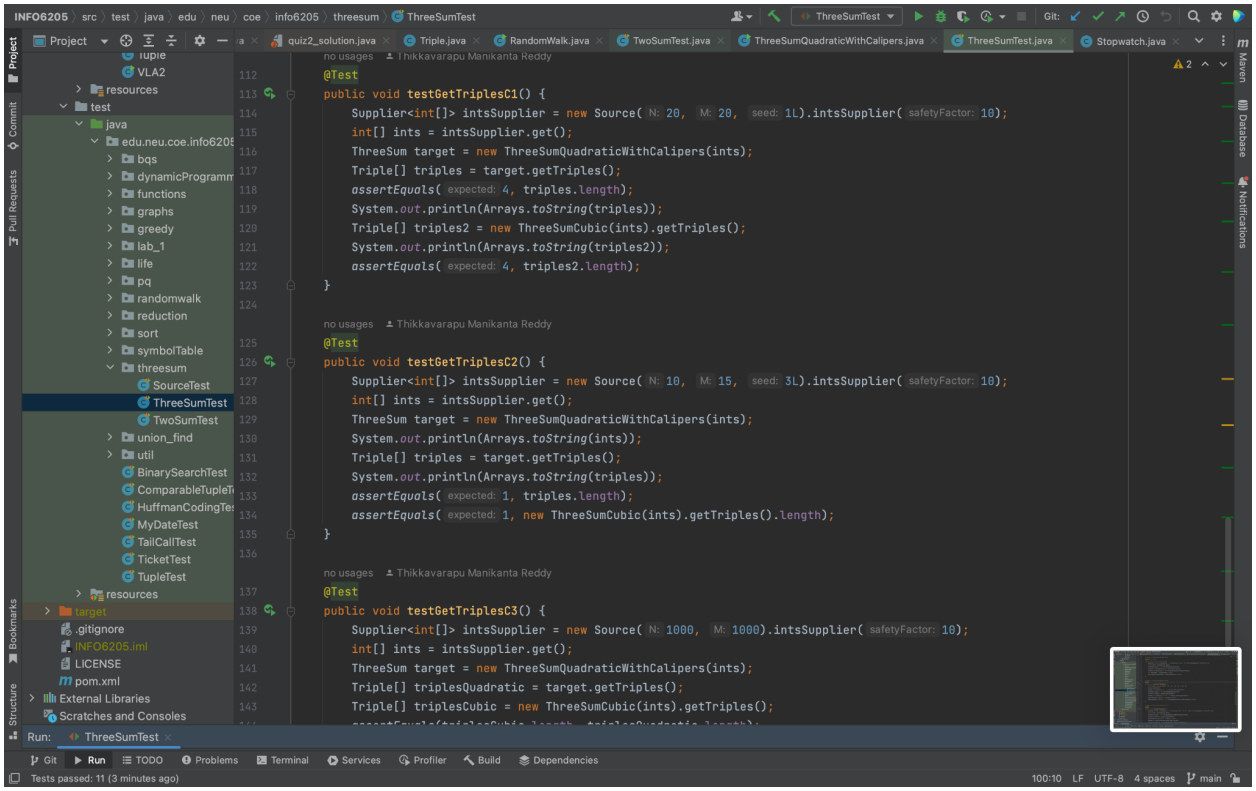



INFO6205 src test java edu neu coe info6205 threesum ThreeSumTest
ThreeSumTest
Git:
Project
quiz2_solution.java
Triple.java
RandomWalk.java
TwoSumTest.java
ThreeSumQuadraticWithCalipers.java
ThreeSumTest.java
Stopwatch.java
Maven
Database
Notifications
Commit
Pull Requests
Bookmarks
Structure
VLA2
resources
test
java
edu.neu.coe.info6205
bqs
dynamicProgramm
functions
graphs
greedy
lab_1
life
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTe
HuffmanCodingTes
MyDateTest
TailCallTest
TicketTest
TupleTest
resources
target
.gitignore
INFO6205.iml
LICENSE
pom.xml
External Libraries
Scratches and Consoles
@Test
public void testGetTriplesJ2() {
Supplier<int[]> intsSupplier = new Source(N: 10, M: 15, seed: 2L).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
List<Triple> triples = target.getTriples(mid: 5);
assertEquals(expected: 1, triples.size());
}
no usages Thikkavarapu Manikanta Reddy
@Test
public void testGetTriples0() {
int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
Arrays.sort(ints);
System.out.println("ints: " + Arrays.toString(ints));
ThreeSum target = new ThreeSumQuadratic(ints);
Triple[] triples = target.getTriples();
System.out.println("triples: " + Arrays.toString(triples));
assertEquals(expected: 4, triples.length);
assertEquals(expected: 4, new ThreeSumCubic(ints).getTriples().length);
}
no usages Thikkavarapu Manikanta Reddy
@Test
public void testGetTriples1() {
Supplier<int[]> intsSupplier = new Source(N: 20, M: 20, seed: 1L).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSum target = new ThreeSumQuadratic(ints);
Triple[] triples = target.getTriples();
assertEquals(expected: 4, triples.length);
System.out.println(Arrays.toString(triples));
Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
System.out.println(Arrays.toString(triples2));
assertEquals(expected: 4, triples2.length);
}
Run: ThreeSumTest
Git Run TODO Problems Terminal Services Profiler Build Dependencies
Tests passed: 11 (2 minutes ago)
100:10 LF UTF-8 4 spaces main

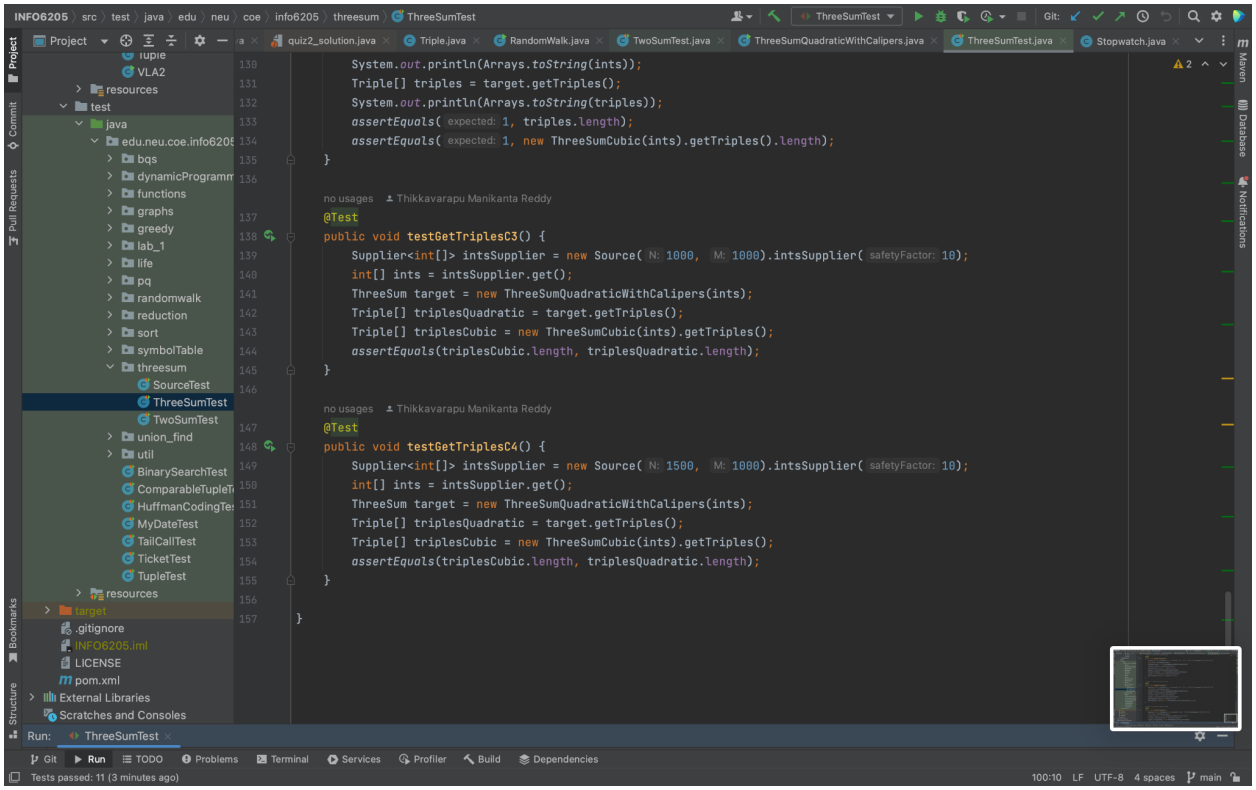

INFO6205 src test java edu neu coe info6205 threesum ThreeSumTest
ThreeSumTest
Git:
Project
quiz2_solution.java
Triple.java
RandomWalk.java
TwoSumTest.java
ThreeSumQuadraticWithCalipers.java
ThreeSumTest.java
Stopwatch.java
Maven
Database
Notifications
Commit
Pull Requests
Bookmarks
Structure
VLA2
resources
test
java
edu.neu.coe.info6205
bqs
dynamicProgramm
functions
graphs
greedy
lab_1
life
pq
randomwalk
reduction
sort
symbolTable
threesum
SourceTest
ThreeSumTest
TwoSumTest
union_find
util
BinarySearchTest
ComparableTupleTe
HuffmanCodingTes
MyDateTest
TailCallTest
TicketTest
TupleTest
resources
target
.gitignore
INFO6205.iml
LICENSE
pom.xml
External Libraries
Scratches and Consoles
no usages Thikkavarapu Manikanta Reddy
@Test
public void testGetTriples2() {
Supplier<int[]> intsSupplier = new Source(N: 10, M: 15, seed: 3L).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSum target = new ThreeSumQuadratic(ints);
System.out.println(Arrays.toString(ints));
Triple[] triples = target.getTriples();
System.out.println(Arrays.toString(triples));
assertEquals(expected: 1, triples.length);
assertEquals(expected: 1, new ThreeSumCubic(ints).getTriples().length);
}
no usages Thikkavarapu Manikanta Reddy
@Ignore // Slow
public void testGetTriples3() {
Supplier<int[]> intsSupplier = new Source(N: 1000, M: 1000).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSum target = new ThreeSumQuadratic(ints);
Triple[] triplesQuadratic = target.getTriples();
Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
int expected1 = triplesCubic.length;
assertEquals(expected1, triplesQuadratic.length);
}
no usages Thikkavarapu Manikanta Reddy
@Ignore // Slow
public void testGetTriples4() {
Supplier<int[]> intsSupplier = new Source(N: 1500, M: 1000).intsSupplier(safetyFactor: 10);
int[] ints = intsSupplier.get();
ThreeSum target = new ThreeSumQuadratic(ints);
Triple[] triplesQuadratic = target.getTriples();
Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
int expected1 = triplesCubic.length;
assertEquals(expected1, triplesQuadratic.length);
Run: ThreeSumTest
Git Run TODO Problems Terminal Services Profiler Build Dependencies
Tests passed: 11 (2 minutes ago)
100:10 LF UTF-8 4 spaces main



```
no usages  Thikkavarapu Manikanta Reddy
@Ignore // Slow
public void testGetTriples4() {
    Supplier<int[]> intsSupplier = new Source( N: 1500,  M: 1000).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triplesQuadratic = target.getTriples();
    Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
    int expected1 = triplesCubic.length;
    assertEquals(expected1, triplesQuadratic.length);
}

no usages  Thikkavarapu Manikanta Reddy
@Test
public void testGetTriplesC0() {
    int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
    Arrays.sort(ints);
    System.out.println("ints: " + Arrays.toString(ints));
    ThreeSum target = new ThreeSumQuadratic(ints);
    Triple[] triples = target.getTriples();
    System.out.println("triples: " + Arrays.toString(triples));
    assertEquals( expected: 4, triples.length);
    assertEquals( expected: 4, new ThreeSumCubic(ints).getTriples().length);
}

no usages  Thikkavarapu Manikanta Reddy
@Test
public void testGetTriplesC1() {
    Supplier<int[]> intsSupplier = new Source( N: 20,  M: 20,  seed: 1L).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
    Triple[] triples = target.getTriples();
    assertEquals( expected: 4, triples.length);
    System.out.println(Arrays.toString(triples));
```



```
no usages  Thikkavarapu Manikanta Reddy
@Test
public void testGetTriplesC1() {
    Supplier<int[]> intsSupplier = new Source( N: 20,  M: 20,  seed: 1L).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
    Triple[] triples = target.getTriples();
    assertEquals( expected: 4, triples.length);
    System.out.println(Arrays.toString(triples));
    Triple[] triples2 = new ThreeSumCubic(ints).getTriples();
    System.out.println(Arrays.toString(triples2));
    assertEquals( expected: 4, triples2.length);
}

no usages  Thikkavarapu Manikanta Reddy
@Test
public void testGetTriplesC2() {
    Supplier<int[]> intsSupplier = new Source( N: 10,  M: 15,  seed: 3L).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
    System.out.println(Arrays.toString(ints));
    Triple[] triples = target.getTriples();
    System.out.println(Arrays.toString(triples));
    assertEquals( expected: 1, triples.length);
    assertEquals( expected: 1, new ThreeSumCubic(ints).getTriples().length);
}

no usages  Thikkavarapu Manikanta Reddy
@Test
public void testGetTriplesC3() {
    Supplier<int[]> intsSupplier = new Source( N: 1000,  M: 1000).intsSupplier( safetyFactor: 10);
    int[] ints = intsSupplier.get();
    ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
    Triple[] triplesQuadratic = target.getTriples();
    Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
```



Code:

```
package edu.neu.coe.info6205.threesum;

import edu.neu.coe.info6205.util.Stopwatch;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
* Implementation of ThreeSum which follows the brute-force approach of
* testing every candidate in the solution-space.
* The array provided in the constructor may be randomly ordered.
* <p>
* This algorithm runs in O(N^3) time.
*/
class ThreeSumCubic implements ThreeSum {
   /**
    * Construct a ThreeSumCubic on a.
    * @param a an array.
    */
   public ThreeSumCubic(int[] a) {
       this.a = a;
       length = a.length;
```

```
    }

    public Triple[] getTriples() {
        List<Triple> triples = new ArrayList<>();
        for (int i = 0; i < length; i++)
            for (int j = i + 1; j < length; j++) {
                for (int k = j + 1; k < length; k++) {
                    if (a[i] + a[j] + a[k] == 0)
                        triples.add(new Triple(a[i], a[j], a[k]));
                }
            }
        Collections.sort(triples);
        return triples.stream().distinct().toArray(Triple[]::new);
    }

    private final int[] a;
    private final int length;

    public static void main(String[] args) {
        int n = 6400;
        int[] array = new int[n];
        for (int j = 0; j < n; j++) {
            int rnd = -50;
            array[j] = rnd;
            rnd += 6;
        }
        ThreeSumCubic threeSumCubic = new ThreeSumCubic(array);
        Stopwatch stopwatch = new Stopwatch();
        threeSumCubic.getTriples();
        System.out.println("Time taken to complete the program : " +
stopwatch.lap() + " milliseconds");
    }
}
```

```
package edu.neu.coe.info6205.threesum;

import edu.neu.coe.info6205.util.Stopwatch;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
* Implementation of ThreeSum which follows the simple optimization of
* requiring a sorted array, then using binary search to find an element x
where
* -x the sum of a pair of elements.
* <p>
```

```java
* The array provided in the constructor MUST be ordered.
* <p>
* This algorithm runs in O(N^2 log N) time.
*/
class ThreeSumQuadrithmic implements ThreeSum {
   /**
    * Construct a ThreeSumQuadrithmic on a.
    * @param a a sorted array.
    */
   public ThreeSumQuadrithmic(int[] a) {
       this.a = a;
       length = a.length;
   }

   public Triple[] getTriples() {
       List<Triple> triples = new ArrayList<>();
       for (int i = 0; i < length; i++)
           for (int j = i + 1; j < length; j++) {
               Triple triple = getTriple(i, j);
               if (triple != null) triples.add(triple);
           }
       Collections.sort(triples);
       return triples.stream().distinct().toArray(Triple[]::new);
   }

   public Triple getTriple(int i, int j) {
       int index = Arrays.binarySearch(a, -a[i] - a[j]);
       if (index >= 0 && index > j) return new Triple(a[i], a[j], a[index]);
       else return null;
   }

   private final int[] a;
   private final int length;

   public static void main(String[] args) {
       int n = 6400;
       int[] array = new int[n];
       for (int j = 0; j < n; j++) {
           int rnd = -50;
           array[j] = rnd;
           rnd += 6;
       }
       ThreeSumQuadrithmic threeSumQuadrithmic = new
ThreeSumQuadrithmic(array);
       Stopwatch stopwatch = new Stopwatch();
       threeSumQuadrithmic.getTriples();
       System.out.println("Time taken to complete the program : " +
stopwatch.lap() + " milliseconds");
   }
```

```
}
```

```
package edu.neu.coe.info6205.threesum;

import edu.neu.coe.info6205.util.Stopwatch;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.Random;

/**
* Implementation of ThreeSum which follows the approach of dividing the
solution-space into
* N sub-spaces where each sub-space corresponds to a fixed value for the
middle index of the three values.
* Each sub-space is then solved by expanding the scope of the other two
indices outwards from the starting point.
* Since each sub-space can be solved in O(N) time, the overall complexity is
O(N^2).
* <p>
* NOTE: The array provided in the constructor MUST be ordered.
*/
public class ThreeSumQuadratic implements ThreeSum {
    /**
     * Construct a ThreeSumQuadratic on a.
     * @param a a sorted array.
     */
    public ThreeSumQuadratic(int[] a) {
        this.a = a;
        length = a.length;
    }

    public Triple[] getTriples() {
        List<Triple> triples = new ArrayList<>();
        for (int i = 0; i < length; i++) triples.addAll(getTriples(i));
        Collections.sort(triples);
        return triples.stream().distinct().toArray(Triple[]::new);
    }

    /**
     * Get a list of Triples such that the middle index is the given value j.
     *
     * @param j the index of the middle value.
     * @return a Triple such that
     */
    public List<Triple> getTriples(int mid) {
```

```
        List<Triple> triples = new ArrayList<>();
        // FIXME : for each candidate, test if a[i] + a[j] + a[k] = 0.
        // END
        int low = mid-1, high = mid+1;
        while(low >= 0 && high < length) {
            int sum = a[low] + a[mid] + a[high];
            if (sum == 0) {
                triples.add(new Triple(a[low], a[mid], a[high]));
                low--; high++;
            }
            else if (sum < 0) {
                high++;
            }
            else if(sum > 0) {
                low--;
            }
        }
        return triples;
    }

    private final int[] a;
    private final int length;

    public static void main(String[] args) {
        int n = 6400;
        int[] array = new int[n];
        for (int j = 0; j < n; j++) {
            int rnd = -50;
            array[j] = rnd;
            rnd += 6;
        }
        ThreeSumQuadratic threeSumQuadratic = new ThreeSumQuadratic(array);
        Stopwatch stopwatch = new Stopwatch();
        threeSumQuadratic.getTriples();
        System.out.println("Time taken to complete the program : " +
stopwatch.lap() + " milliseconds");
    }

}
```

```
package edu.neu.coe.info6205.threesum;

import edu.neu.coe.info6205.util.Stopwatch;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.function.Function;
```

```java
/**
* Implementation of ThreeSum which follows the approach of dividing the
solution-space into
* N sub-spaces where each sub-space corresponds to a fixed value for the
middle index of the three values.
* Each sub-space is then solved by expanding the scope of the other two
indices outwards from the starting point.
* Since each sub-space can be solved in O(N) time, the overall complexity is
O(N^2).
* <p>
* The array provided in the constructor MUST be ordered.
*/
public class ThreeSumQuadraticWithCalipers implements ThreeSum {
   /**
    * Construct a ThreeSumQuadratic on a.
    *
    * @param a a sorted array.
    */
   public ThreeSumQuadraticWithCalipers(int[] a) {
       this.a = a;
       length = a.length;
   }

   /**
    * Get an array or Triple containing all of those triples for which sum is
zero.
    *
    * @return a Triple[].
    */
   public Triple[] getTriples() {
       List<Triple> triples = new ArrayList<>();
       Collections.sort(triples); // ???
       for (int i = 0; i < length - 2; i++)
           triples.addAll(calipers(a, i, Triple::sum));
       return triples.stream().distinct().toArray(Triple[]::new);
   }

   /**
    * Get a set of candidate Triples such that the first index is the given
value i.
    * Any candidate triple is added to the result if it yields zero when
passed into function.
    *
    * @param a        a sorted array of ints.
    * @param i        the index of the first element of resulting triples.
    * @param function a function which takes a triple and returns a value
which will be compared with zero.
    * @return a List of Triples.
    */
```

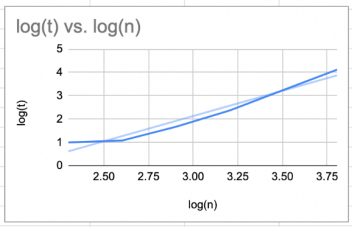
```
   public static List<Triple> calipers(int[] a, int i, Function<Triple,
Integer> function) {
       List<Triple> triples = new ArrayList<>();
       // FIXME : use function to qualify triples and to navigate otherwise.
       // END
       if (i > 0 && a[i] == a[i - 1]) {

       }
       else {
       int p1 = i+1, p2 = a.length-1;
       while(p1 < p2) {
           if (p2 < a.length - 1 && a[p2] == a[p2 + 1]) {
               p2--;
               continue;
           }
           int sum = a[i] + a[p1] + a[p2];
           if(sum == 0) {
               triples.add(new Triple(a[i], a[p1], a[p2]));
               p1++;p2--;
           }
           else if(sum < 0) {
               p1++;
           }
           else if(sum > 0) {
               p2--;
           }
       }
   }
       return triples;
   }

   private final int[] a;
   private final int length;

   public static void main(String[] args) {
       int n = 6400;
       int[] array = new int[n];
       for (int j = 0; j < n; j++) {
           int rnd = -50;
           array[j] = rnd;
           rnd += 6;
       }
       ThreeSumQuadraticWithCalipers threeSumQuadraticWithCalipers = new
ThreeSumQuadraticWithCalipers(array);
       Stopwatch stopwatch = new Stopwatch();
       threeSumQuadraticWithCalipers.getTriples();
       System.out.println("Time taken to complete the program : " +
stopwatch.lap() + " milliseconds");
   }
```

```
}
```

```
package edu.neu.coe.info6205.util;

/**
* Simple benchmarking tool: Stopwatch.
* There is no warm-up here, no pause/resume and no repetition (for these use
Timer).
* It is simply a convenient way to time an execution with results in
milliseconds.
* <p>
* Once the Stopwatch is started (i.e. constructed), you "read" the stopwatch
by calling lap(),
* which returns the number of milliseconds since the previous lap (or since
the start).
*/
public class Stopwatch implements AutoCloseable {

   /**
    * Construct and start a Stopwatch.
    */
   public Stopwatch() {
       start = System.nanoTime();
   }

   /**
    * @return the number of milliseconds elapsed since the last lap (or
start).
    * @throws AssertionError if this Stopwatch has been closed already.
    */
   public long lap() {
       assert start != null : "Stopwatch is closed";
       final long lapStart = start;
       start = System.nanoTime();
       return (start - lapStart) / 1000000;
   }

   /**
    * Close this Stopwatch.
    * NOTE that you should run the Stopwatch within a try-with-resource
statement. See unit tests.
    */
   public void close() {
       start = null;
   }

   private Long start;
}
```

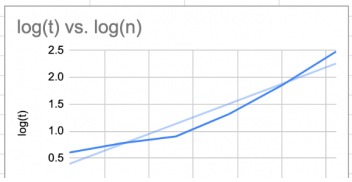
ThreeSumCubic

n	t (milliseconds)	log(n)	log(t)
200	10	2.301029996	1
400	12	2.602059991	1.079181246
800	47	2.903089987	1.672097858
1600	234	3.204119983	2.369215857
3200	1767	3.505149978	3.24723655
6400	13222	3.806179974	4.121297153



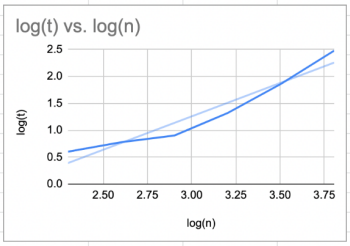
ThreeSumQuadrithmic

n	t (milliseconds)	log(n)	log(t)
200	4	2.301029996	0.6020599913
400	6	2.602059991	0.7781512504
800	8	2.903089987	0.903089987
1600	21	3.204119983	1.322219295
3200	73	3.505149978	1.86332286



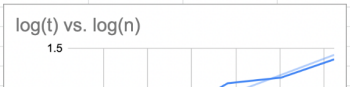
ThreeSumQuadrithmic

n	t (milliseconds)	log(n)	log(t)
200	4	2.301029996	0.6020599913
400	6	2.602059991	0.7781512504
800	8	2.903089987	0.903089987
1600	21	3.204119983	1.322219295
3200	73	3.505149978	1.86332286
6400	302	3.806179974	2.480006943



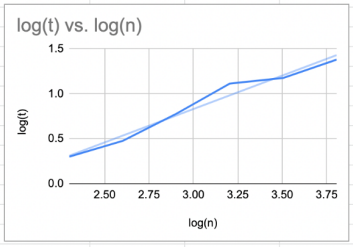
ThreeSumQuadratic

n	t (milliseconds)	log(n)	log(t)
200	2	2.301029996	0.3010299957



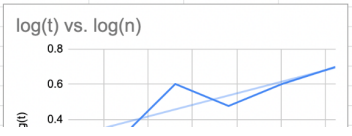
ThreeSumQuadratic

n	t (milliseconds)	log(n)	log(t)
200	2	2.301029996	0.3010299957
400	3	2.602059991	0.4771212547
800	6	2.903089987	0.7781512504
1600	13	3.204119983	1.113943352
3200	15	3.505149978	1.176091259
6400	24	3.806179974	1.380211242

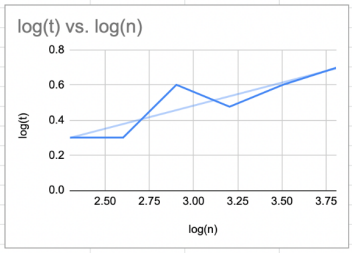


ThreeSumQuadraticWithCalipers

n	t (milliseconds)	log(n)	log(t)
200	2	2.301029996	0.3010299957
400	2	2.602059991	0.3010299957
800	4	2.903089987	0.6020599913
1600	3	3.204119983	0.4771212547
3200	4	3.505149978	0.6020599913



ThreeSumQuadraticWithCalipers			
n	t (milliseconds)	log(n)	log(t)
200	2	2.301029996	0.3010299957
400	2	2.602059991	0.3010299957
800	4	2.903089987	0.6020599913
1600	3	3.204119983	0.4771212547
3200	4	3.505149978	0.6020599913
6400	5	3.806179974	0.6989700043



Observations:

Time complexity:

ThreeSumCubic -> O(n3)
ThreeSumQuadrithmic -> O(n2logn)
ThreeSumQuadratic & ThreeSumQuadraticWithCalipers -> O(n2)

As we ran the 4 different programs each with different n (array length) values which resulted in different code run times. [Used StopWatch class to calculate the program run times]

After plotting these points in excel sheet where log(n) as x-axis and log(t) as y-axis we are getting a linear graph and by comparing the linear graphs of these programs, it is clearly evident that the ThreeSumQuadratic & ThreeSumQuadraticWithCalipers are taking lesser times to complete the program for different n values [array lengths]. ThreeSumCubic is the slower one among these programs.

If we club all 4 linear graphs, will observe ThreeSumCubic will be at the topmost and ThreeSumQuadratic & ThreeSumQuadraticWithCalipers will be at the bottom.

Why quadratic method works with O(n2) ?

A ThreeSum implementation that uses the strategy of partitioning the solution space into N subspaces, each of which corresponds to a fixed value for the middle index of the three values. Then, for each sub-space, the problem is addressed by extending the range of the first two indices. The overall complexity is O(N2) because it is possible to solve each subspace in O(N) time.

-> Initially mid = 0, low = mid - 1, high = mid + 1.
-> Use a while loop with a condition low>=0 && high<length
-> Use three if conditions, firstly check the sum of a[low]+a[mid]+a[high] is equal to zero
-> Secondly, check the sum of a[low]+a[mid]+a[high] is less than zero
-> Finally, check the sum of a[low]+a[mid]+a[high] is greater than zero
-> If first condition passes then store the triple and do low– & high++
-> If second condition passes then do high++;
-> If third condition passes then do low–;
-> Perform the above steps repetitively till u come across the while loop terminating condition.
-> finally return the triples arraylist.