

## Program Structures and Algorithms Spring 2023(SEC -03)

NAME: SAHITHI BOMMINENI  
NUID: 002768024

### Task: Assignment 2 (3-SUM)

### Unit Test Screenshots:

```
48  */
49  @
50  public static List<Triple> calipers(int[] a, int i, Function<Triple, Integer> function) {
51      List<Triple> triples = new ArrayList<>();
52      // FIXME : use function to qualify triples and to navigate otherwise.
53      int j = i + 1;
54      int k = a.length - 1;
55      while (j < k) {
56          int sum = a[i] + a[j] + a[k];
57          Triple triple = new Triple(a[i], a[j], a[k]);
58          int result = function.apply(triple);
59          if (result == 0) {
60              triples.add(triple);
61              j++;
62              k--;
63          } else if (result < 0) j++;
64          else k--;
65      }
66      return triples;
67  }
```

Run: ThreeSumTest x Tests passed: 11 of 11 tests - 2 sec 137ms

Test	Duration
testGetTriples0	52ms
testGetTriples1	9ms
testGetTriples2	7ms
testGetTriplesC0	3ms
testGetTriplesC1	7ms
testGetTriplesC2	3ms
testGetTriplesC3	625ms
testGetTriplesC4	1sec 428ms
testGetTriplesJ0	1ms
testGetTriplesJ1	1ms
testGetTriplesJ2	1ms

```
131  System.out.println(Arrays.toString(triples));
132  assertEquals( expected: 1, triples.length);
133  assertEquals( expected: 1, new ThreeSumCubic(ints).getTriples().length);
134  }
135
136  no usages
137  @Test
138  public void testGetTriplesC3() {
139      Supplier<int[]> intsSupplier = new Source( N: 1000, M: 1000).intsSupplier( safetyFactor: 10);
140      int[] ints = intsSupplier.get();
141      ThreeSum target = new ThreeSumQuadraticWithCalipers(ints);
142      Triple[] triplesQuadratic = target.getTriples();
143      Triple[] triplesCubic = new ThreeSumCubic(ints).getTriples();
144      assertEquals(triplesCubic.length, triplesQuadratic.length);
145  }
```

Run: ThreeSumTest x Tests passed: 11 of 11 tests - 4 sec 950ms

Test	Duration
testGetTriples0	51ms
testGetTriples1	11ms
testGetTriples2	5ms
testGetTriplesC0	2ms
testGetTriplesC1	7ms
testGetTriplesC2	2ms
testGetTriplesC3	2 sec 598ms
testGetTriplesC4	2 sec 271ms
testGetTriplesJ0	2ms
testGetTriplesJ1	0ms
testGetTriplesJ2	1ms

The screenshot shows an IDE with the following components:

- Project Explorer:** Lists files including Pair, Source, ThreeSum, ThreeSumBenchmark, ThreeSumCubic, ThreeSumQuadratic, ThreeSumQuadraticW, ThreeSumQuadrithmik, Triple, TwoSum, TwoSumBenchmark, TwoSumQuadratic, TwoSumWithCalipers, union\_find, util, BinarySearch, and CallByValue.
- Editor:** Displays the code for `ThreeSumTest.java`. The code includes a `@Test` method `testGetTriplesC3()` that uses `Supplier<int[]>` to provide input arrays and `ThreeSumQuadraticWithCalipers` to find triplets. It asserts the length of the returned triplets.
- Run Console:** Shows the output of the tests. It indicates that 11 of 11 tests passed in 3 seconds and 623 milliseconds. The output lists the input arrays and the resulting triplets for each test case.

## Observation:

The quadratic algorithms are efficient because they divide the problem space into  $N$  subspaces, with each subspace associated with a specific value of the middle index among three values. Once the middle index is fixed, the only remaining step to solve the problem is to identify two elements in the remaining indices that add up to the negative of the middle element. This is a critical factor in solving the problem.

By fixing the middle index and solving the problem for each subspace, the need to compare every possible triplet in the input array, which would take cubic time, is avoided. Instead, each subspace is solved in linear time using two pointers to search for the remaining two elements whose sum is the negative of the middle element.

This approach reduces the time complexity of the three-sum problem from cubic to quadratic, resulting in a significant improvement in running time.