Program Structures and Algorithms

Name: Chakradhar Grandhi Durga. **NEU ID**: 002933727

ASSIGNMENT-3

1.1 TIMER

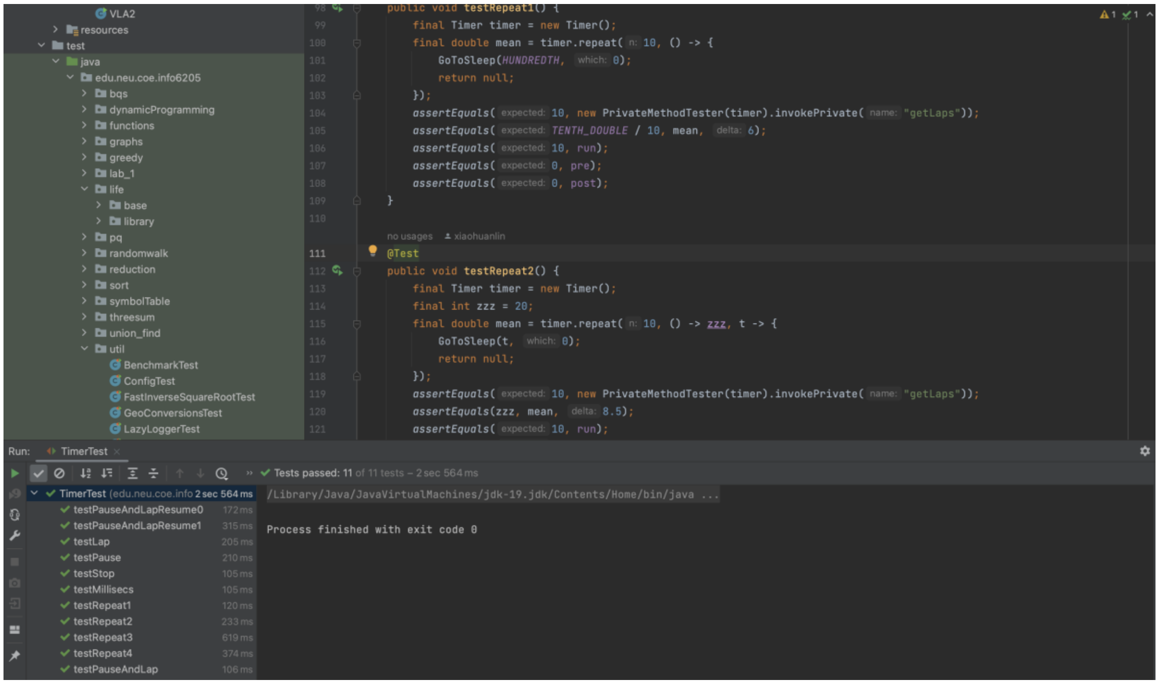
Here is the code for the benchmark timer class. I changed the repeat function in the timer class along with 2 other classes and here is the code.

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U>
function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: with " + n + " runs");
    T t = supplier.get();
    for (int i = 0; i < n; i++) {
        if (preFunction != null) {
            pause();
            t = preFunction.apply(t);
            resume();
        }

        U u = function.apply(t);
        lap();

        if (postFunction != null) {
            pause();
            postFunction.accept(u);
            resume();
        }

    }
    pause();
    double meantime = meanLapTime();
    resume();
    return meantime;
    // END
}
```

```
private static long getClock() {
    return System.nanoTime();

    // END
}
```

```
public double millisecs() {
    if (running) throw new TimerException();
    return toMillisecs(ticks);
}
```

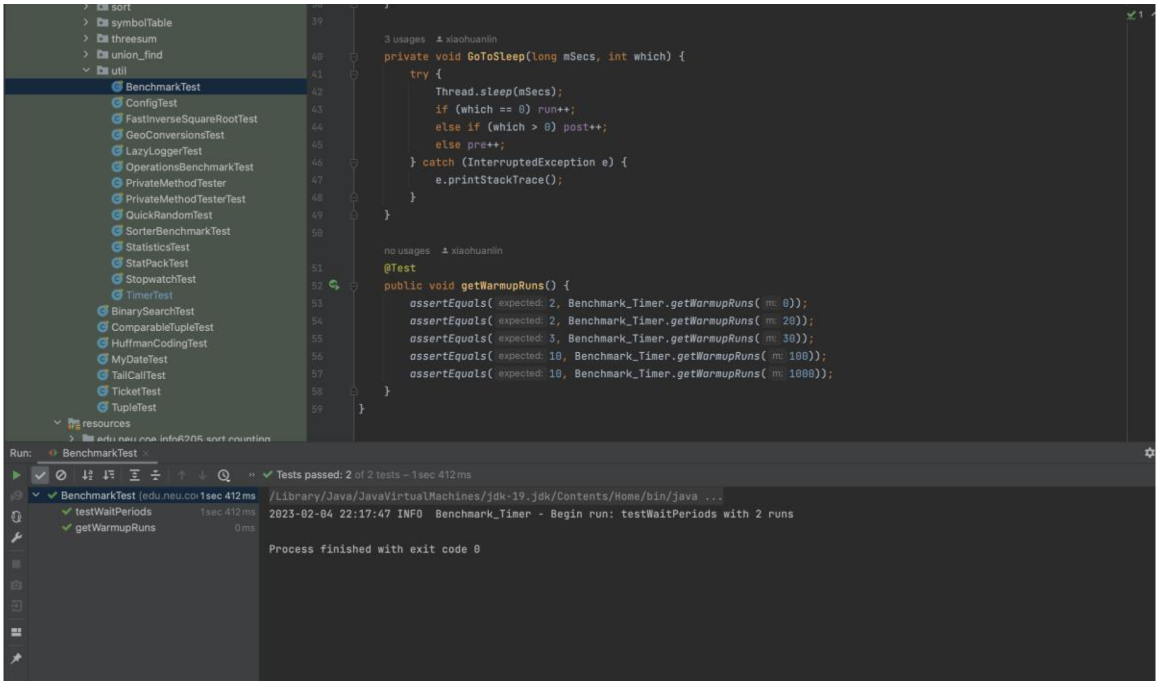
Here is the test cases for the timers. Here is the output for the Timertest



```
public void testRepeat1() {
    final Timer timer = new Timer();
    final double mean = timer.repeat( n: 10, () -> {
        GoToSleep(HUNDREDTH,  which: 0);
        return null;
    });
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals( expected: TENTH_DOUBLE / 10, mean,  delta: 6);
    assertEquals( expected: 10, run);
    assertEquals( expected: 0, pre);
    assertEquals( expected: 0, post);
}

no usages  xiaohuanlin
@Test
public void testRepeat2() {
    final Timer timer = new Timer();
    final int zzz = 20;
    final double mean = timer.repeat( n: 10, () -> zzz, t -> {
        GoToSleep(t,  which: 0);
        return null;
    });
    assertEquals( expected: 10, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    assertEquals(zzz, mean,  delta: 8.5);
    assertEquals( expected: 10, run);
```

```
✓ Tests passed: 11 of 11 tests – 2 sec 564 ms

✓ TimerTest (edu.neu.coe.info 2 sec 564 ms
    ✓ testPauseAndLapResume0  172 ms
    ✓ testPauseAndLapResume1  315 ms
    ✓ testLap                 205 ms
    ✓ testPause               210 ms
    ✓ testStop                105 ms
    ✓ testMillisecs           105 ms
    ✓ testRepeat1             120 ms
    ✓ testRepeat2             233 ms
    ✓ testRepeat3             619 ms
    ✓ testRepeat4             374 ms
    ✓ testPauseAndLap         106 ms

/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Here is the output for the BenchmarkTest test cases. You can see that the testcases passed successfully.



```
3 usages  xiaohuanlin
private void GoToSleep(long mSecs, int which) {
    try {
        Thread.sleep(mSecs);
        if (which == 0) run++;
        else if (which > 0) post++;
        else pre++;
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

no usages  xiaohuanlin
@Test
public void getWarmupRuns() {
    assertEquals( expected: 2, Benchmark_Timer.getWarmupRuns( m: 0));
    assertEquals( expected: 2, Benchmark_Timer.getWarmupRuns( m: 20));
    assertEquals( expected: 3, Benchmark_Timer.getWarmupRuns( m: 30));
    assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 100));
    assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 1000));
}
}
```

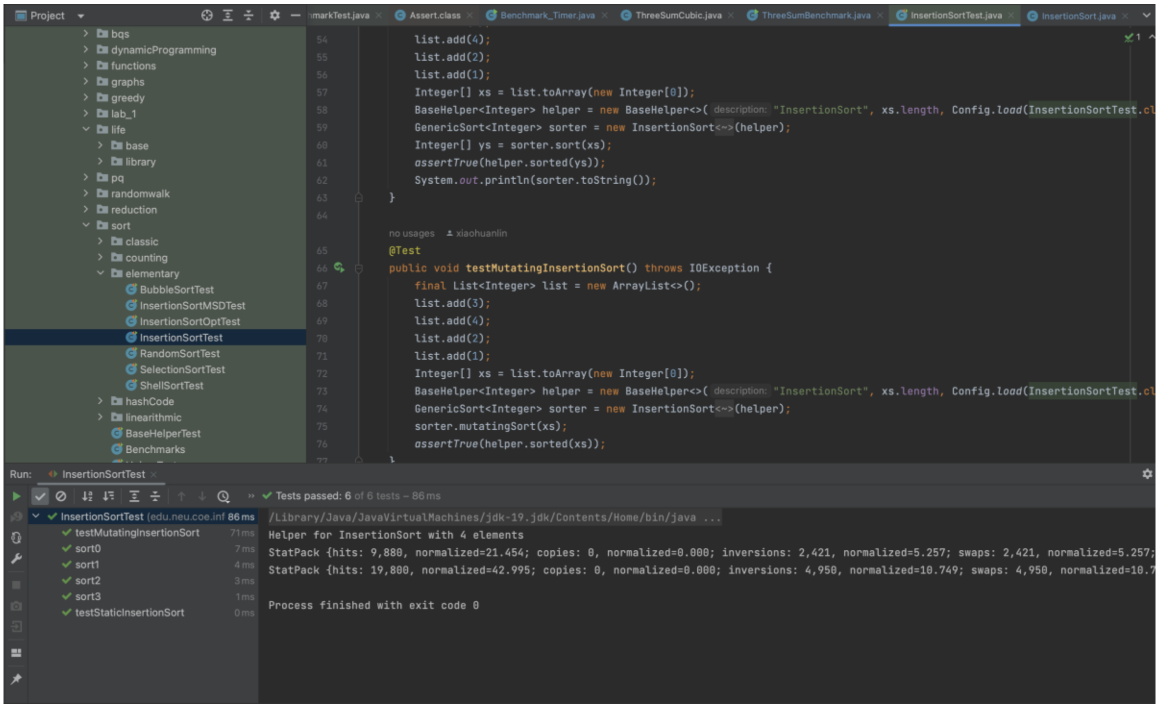
```
✓ Tests passed: 2 of 2 tests – 1 sec 412 ms

✓ BenchmarkTest (edu.neu.coe 1 sec 412 ms
    ✓ testWaitPeriods  1 sec 412 ms
    ✓ getWarmupRuns    0 ms

/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
2023-02-04 22:17:47 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

1.2 INSERTION-SORT

Here is the code for the insertion sort.

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for(int i = from+1; i < to ; i++){
        int k =i;
        while(k > from && helper.swapStableConditional(xs, k) ){
            k--;
        }
    }
}
```

And here are is the screenshot for the insertion sort test. As you can see that all the test cases are passed in the below context.



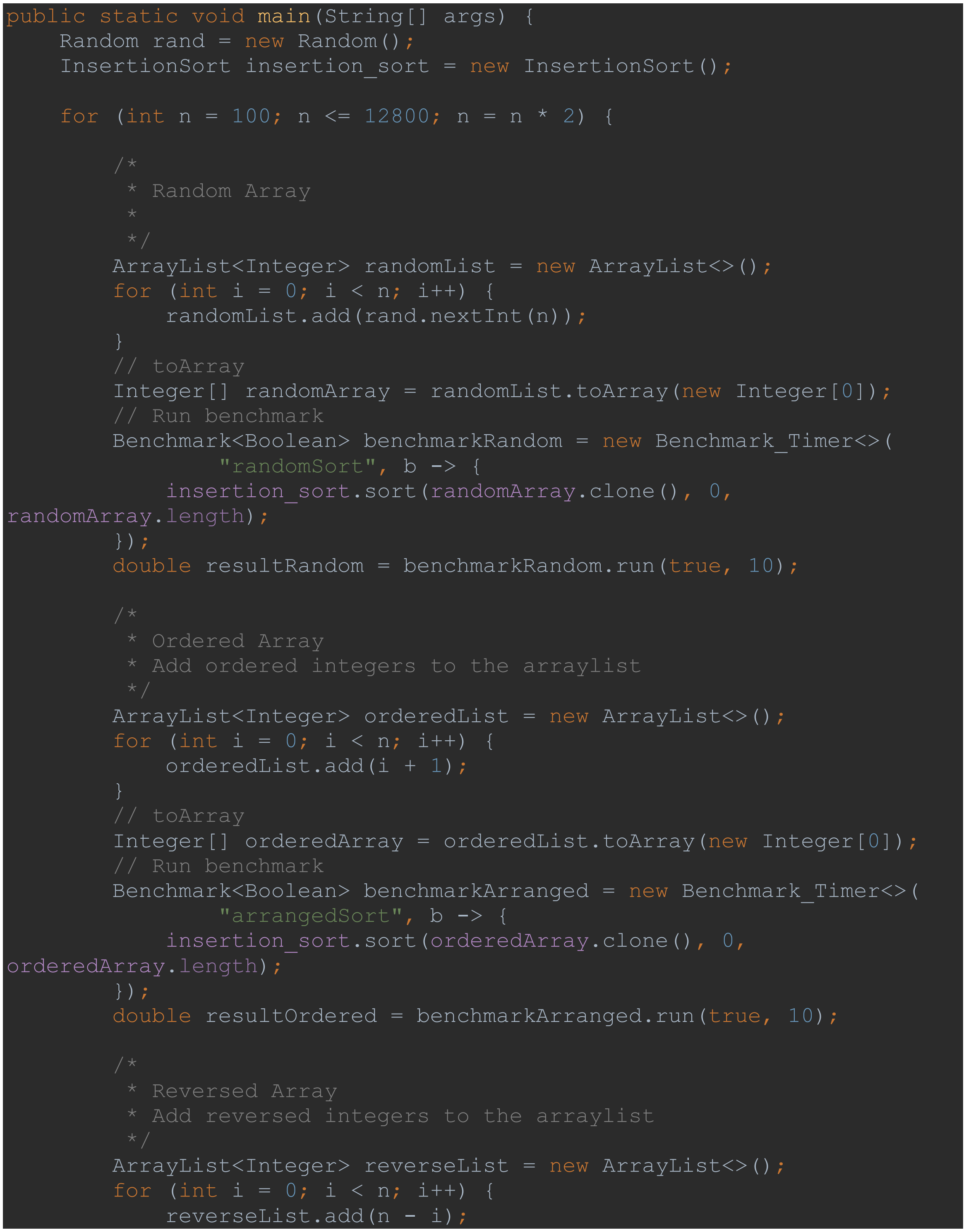
```
        list.add(4);
        list.add(2);
        list.add(1);
        Integer[] xs = list.toArray(new Integer[0]);
        BaseHelper<Integer> helper = new BaseHelper<>( description: "InsertionSort", xs.length, Config.load(InsertionSortTest.cl
        GenericSort<Integer> sorter = new InsertionSort<~>(helper);
        Integer[] ys = sorter.sort(xs);
        assertTrue(helper.sorted(ys));
        System.out.println(sorter.toString());
    }

    no usages   xiaohuanlin
    @Test
    public void testMutatingInsertionSort() throws IOException {
        final List<Integer> list = new ArrayList<>();
        list.add(3);
        list.add(4);
        list.add(2);
        list.add(1);
        Integer[] xs = list.toArray(new Integer[0]);
        BaseHelper<Integer> helper = new BaseHelper<>( description: "InsertionSort", xs.length, Config.load(InsertionSortTest.cl
        GenericSort<Integer> sorter = new InsertionSort<~>(helper);
        sorter.mutatingSort(xs);
        assertTrue(helper.sorted(xs));
```

```
✓ Tests passed: 6 of 6 tests – 86 ms

✓ InsertionSortTest (edu.neu.coe.inf 86 ms
    ✓ testMutatingInsertionSort     71 ms
    ✓ sort0                          7 ms
    ✓ sort1                          4 ms
    ✓ sort2                          3 ms
    ✓ sort3                          1 ms
    ✓ testStaticInsertionSort        0 ms
```

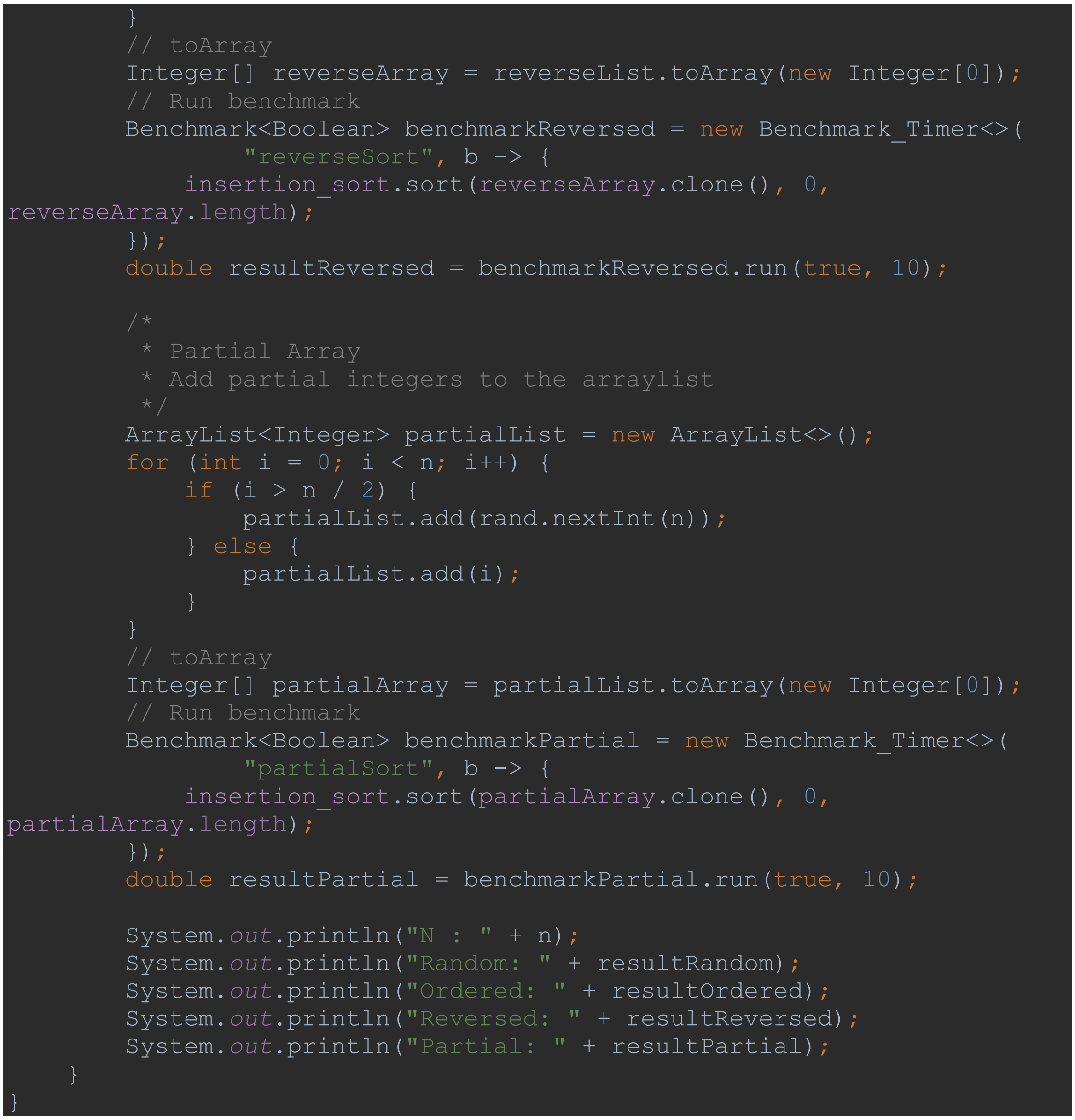
```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
Helper for InsertionSort with 4 elements
StatPack {hits: 9,880, normalized=21.454; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normalized=5.257;
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=10.7

Process finished with exit code 0
```

1.3 MAIN-PROGRAM

Here is the following main program. The Main Function In BenchmarkTimer Class. This creates 4 different types of arrays and time it with Sort method in Insertion Sort java class.

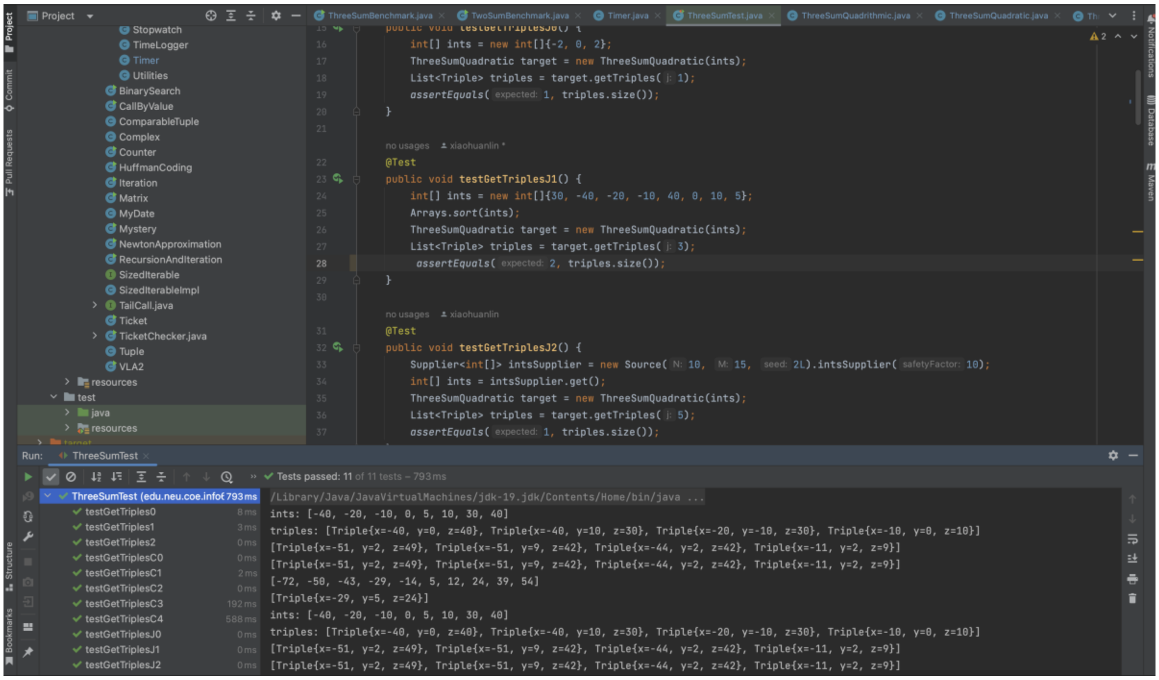


```
public static void main(String[] args) {
    Random rand = new Random();
    InsertionSort insertion_sort = new InsertionSort();

    for (int n = 100; n <= 12800; n = n * 2) {

        /*
         * Random Array
         *
         */
        ArrayList<Integer> randomList = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            randomList.add(rand.nextInt(n));
        }
        // toArray
        Integer[] randomArray = randomList.toArray(new Integer[0]);
        // Run benchmark
        Benchmark<Boolean> benchmarkRandom = new Benchmark_Timer<>(
                "randomSort", b -> {
            insertion_sort.sort(randomArray.clone(), 0,
randomArray.length);
        });
        double resultRandom = benchmarkRandom.run(true, 10);

        /*
         * Ordered Array
         * Add ordered integers to the arraylist
         */
        ArrayList<Integer> orderedList = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            orderedList.add(i + 1);
        }
        // toArray
        Integer[] orderedArray = orderedList.toArray(new Integer[0]);
        // Run benchmark
        Benchmark<Boolean> benchmarkArranged = new Benchmark_Timer<>(
                "arrangedSort", b -> {
            insertion_sort.sort(orderedArray.clone(), 0,
orderedArray.length);
        });
        double resultOrdered = benchmarkArranged.run(true, 10);

        /*
         * Reversed Array
         * Add reversed integers to the arraylist
         */
        ArrayList<Integer> reverseList = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            reverseList.add(n - i);
```



```
        }
        // toArray
        Integer[] reverseArray = reverseList.toArray(new Integer[0]);
        // Run benchmark
        Benchmark<Boolean> benchmarkReversed = new Benchmark_Timer<>(
                "reverseSort", b -> {
            insertion_sort.sort(reverseArray.clone(), 0,
reverseArray.length);
        });
        double resultReversed = benchmarkReversed.run(true, 10);

        /*
         * Partial Array
         * Add partial integers to the arraylist
         */
        ArrayList<Integer> partialList = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            if (i > n / 2) {
                partialList.add(rand.nextInt(n));
            } else {
                partialList.add(i);
            }
        }
        // toArray
        Integer[] partialArray = partialList.toArray(new Integer[0]);
        // Run benchmark
        Benchmark<Boolean> benchmarkPartial = new Benchmark_Timer<>(
                "partialSort", b -> {
            insertion_sort.sort(partialArray.clone(), 0,
partialArray.length);
        });
        double resultPartial = benchmarkPartial.run(true, 10);

        System.out.println("N : " + n);
        System.out.println("Random: " + resultRandom);
        System.out.println("Ordered: " + resultOrdered);
        System.out.println("Reversed: " + resultReversed);
        System.out.println("Partial: " + resultPartial);
    }
}
```

And here are the timing results for the following class.



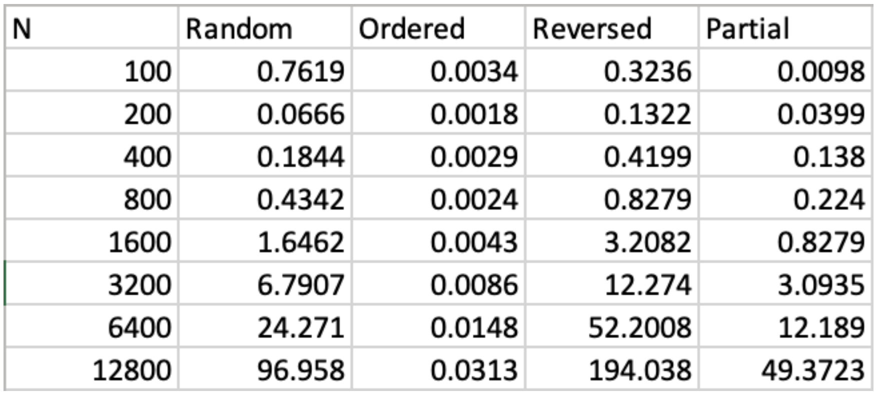
```
        int[] ints = new int[]{-2, 0, 2};
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples( j: 1);
        assertEquals( expected: 1, triples.size());
    }

    no usages  xiaohuanlin *
    @Test
    public void testGetTriplesJ1() {
        int[] ints = new int[]{30, -40, -20, -10, 40, 0, 10, 5};
        Arrays.sort(ints);
        ThreeSumQuadratic target = new ThreeSumQuadratic(ints);
        List<Triple> triples = target.getTriples( j: 3);
         assertEquals( expected: 2, triples.size());
    }

    no usages  xiaohuanlin
    @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( j: 5);
        assertEquals( expected: 1, triples.size());
```

```
✓ Tests passed: 11 of 11 tests – 793 ms

✓ ThreeSumTest (edu.neu.coe.infot 793 ms
  ✓ testGetTriples0      8 ms
  ✓ testGetTriples1      3 ms
  ✓ testGetTriples2      0 ms
  ✓ testGetTriplesC0     0 ms
  ✓ testGetTriplesC1     2 ms
  ✓ testGetTriplesC2     0 ms
  ✓ testGetTriplesC3   192 ms
  ✓ testGetTriplesC4   588 ms
  ✓ testGetTriplesJ0     0 ms
  ✓ testGetTriplesJ1     0 ms
  ✓ testGetTriplesJ2     0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[-72, -50, -43, -29, -14, 5, 12, 24, 39, 54]
[Triple{x=-29, y=5, z=24}]
ints: [-40, -20, -10, 0, 5, 10, 30, 40]
triples: [Triple{x=-40, y=0, z=40}, Triple{x=-40, y=10, z=30}, Triple{x=-20, y=-10, z=30}, Triple{x=-10, y=0, z=10}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
[Triple{x=-51, y=2, z=49}, Triple{x=-51, y=9, z=42}, Triple{x=-44, y=2, z=42}, Triple{x=-11, y=2, z=9}]
```

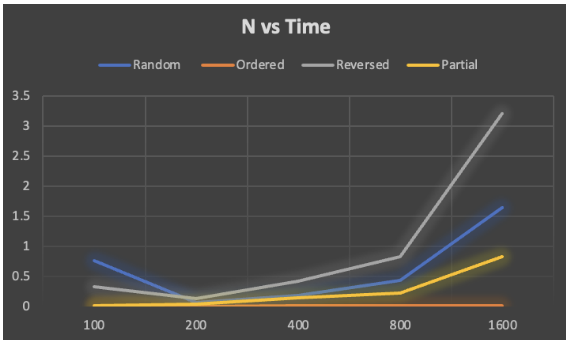
1.4 STATISTICS

I run the main code multiple times so that I can get the most accurate results and here is table of the results.

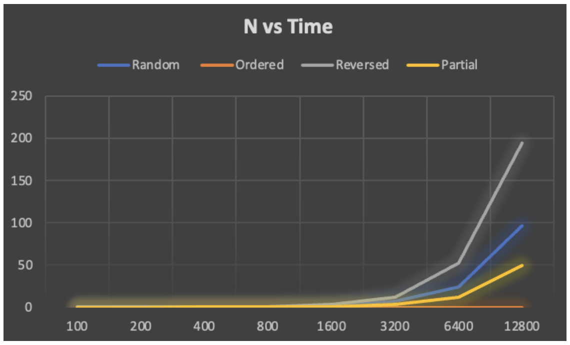


N	Random	Ordered	Reversed	Partial
100	0.7619	0.0034	0.3236	0.0098
200	0.0666	0.0018	0.1322	0.0399
400	0.1844	0.0029	0.4199	0.138
800	0.4342	0.0024	0.8279	0.224
1600	1.6462	0.0043	3.2082	0.8279
3200	6.7907	0.0086	12.274	3.0935
6400	24.271	0.0148	52.2008	12.189
12800	96.958	0.0313	194.038	49.3723

Here are the results of the initial progression of the graph and the final graph.



Initial progression



Final Progression

1.5 Conclusion

From the above graph and the data table, we can clearly see that insertion sort works best only for the ordered arrays. And for the worst case i.e the reverse order, the amount of time taken is hugely differing to that of a random case or the partial cases.

The number of comparisions taken by insertion sort is

$$\frac{N(N-1)}{4}$$

The graph shows a clear description of how insertion sort can take **O(N)** for the bestcase scenario (when the array is sorted/ has minimal number of swaps) and how it takes $\boldsymbol{O}(\boldsymbol{N}^{\mathbf{2}})$ for the worst case.