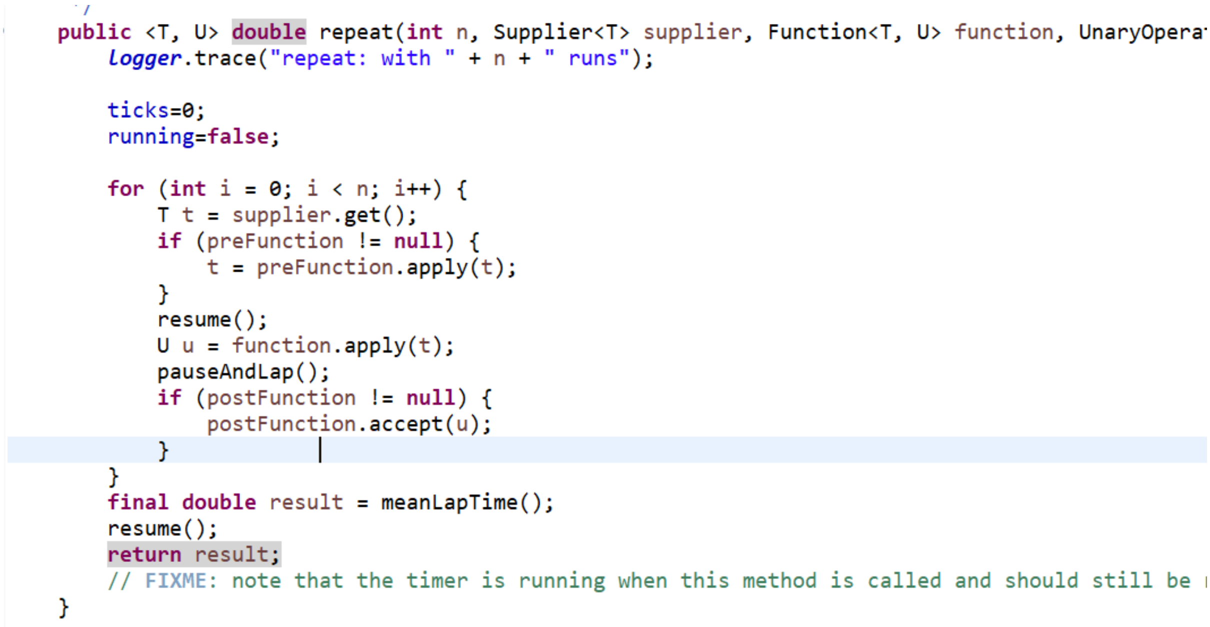
Program Structures and Algorithms
Spring 2023(Section - 1)

Name: Mani Charan Reddy Loka
NUID: 002727403

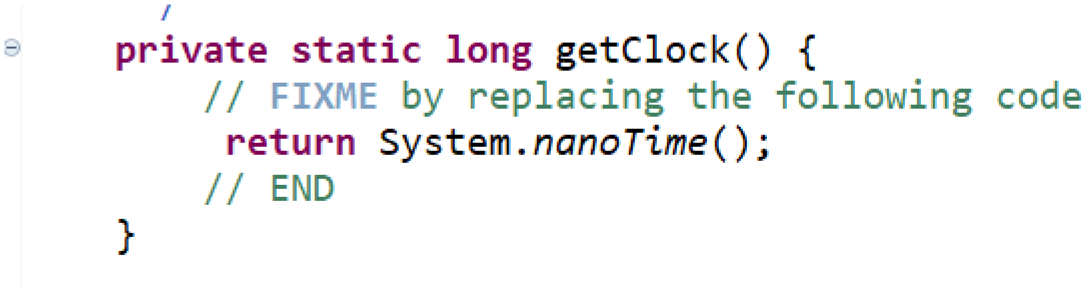
Task:
(Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*.

Repeat method:



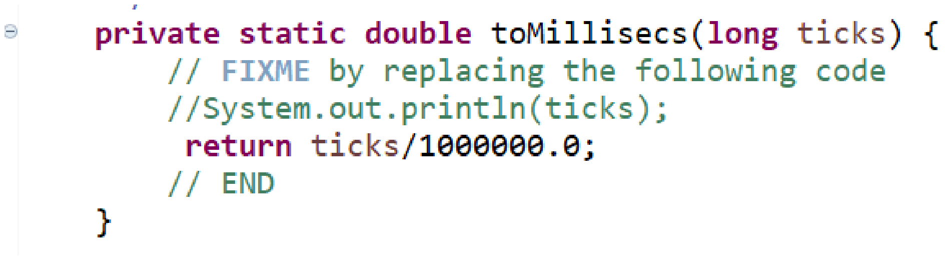
```
    public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOpera
        logger.trace("repeat: with " + n + " runs");

        ticks=0;
        running=false;

        for (int i = 0; i < n; i++) {
            T t = supplier.get();
            if (preFunction != null) {
                t = preFunction.apply(t);
            }
            resume();
            U u = function.apply(t);
            pauseAndLap();
            if (postFunction != null) {
                postFunction.accept(u);
            }
        }
        final double result = meanLapTime();
        resume();
        return result;
        // FIXME: note that the timer is running when this method is called and should still be
    }
```

getClock method:



```
    private static long getClock() {
        // FIXME by replacing the following code
         return System.nanoTime();
        // END
    }
```

toMillisecs method:

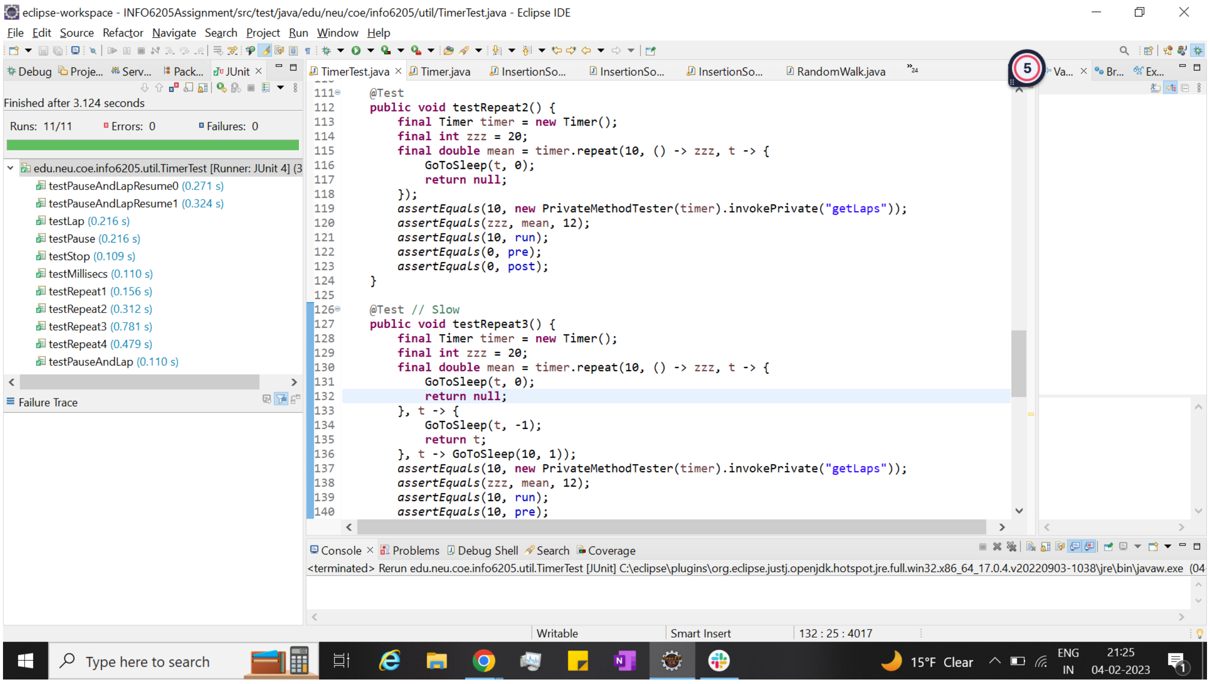


```
    private static double toMillisecs(long ticks) {
        // FIXME by replacing the following code
        //System.out.println(ticks);
         return ticks/1000000.0;
        // END
    }
```

Unit Test Screenshots:

1)TimerTest.java

Here, I updated the delta to a slightly higher value of 12 because I was always getting a deviation of around 10~11.



eclipse-workspace - INFO6205Assignment/src/test/java/edu/neu/coe/info6205/util/TimerTest.java - Eclipse IDE

Finished after 3.124 seconds

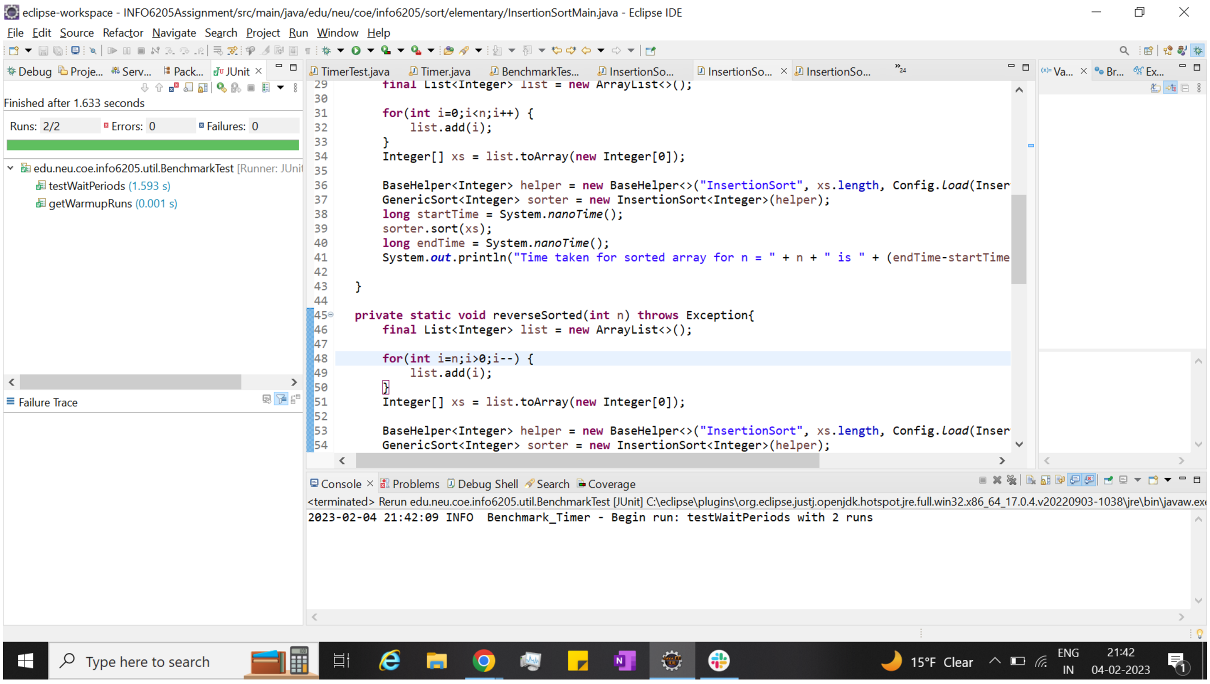
Runs: 11/11 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.util.TimerTest [Runner: JUnit 4] (3
    testPauseAndLapResume0 (0.271 s)
    testPauseAndLapResume1 (0.324 s)
    testLap (0.216 s)
    testPause (0.216 s)
    testStop (0.109 s)
    testMillisecs (0.110 s)
    testRepeat1 (0.156 s)
    testRepeat2 (0.312 s)
    testRepeat3 (0.781 s)
    testRepeat4 (0.479 s)
    testPauseAndLap (0.110 s)
```

```java
    @Test
    public void testRepeat2() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        });
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 12);
        assertEquals(10, run);
        assertEquals(0, pre);
        assertEquals(0, post);
    }

    @Test // Slow
    public void testRepeat3() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        }, t -> {
            GoToSleep(t, -1);
            return t;
        }, t -> GoToSleep(10, 1));
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 12);
        assertEquals(10, run);
        assertEquals(10, pre);
```

```
<terminated> Rerun edu.neu.coe.info6205.util.TimerTest [JUnit] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe (04
```

2) BenchmarkTest.java



eclipse-workspace - INFO6205Assignment/src/main/java/edu/neu/coe/info6205/sort/elementary/InsertionSortMain.java - Eclipse IDE

Finished after 1.633 seconds

Runs: 2/2 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.util.BenchmarkTest [Runner: JUnit
    testWaitPeriods (1.593 s)
    getWarmupRuns (0.001 s)
```

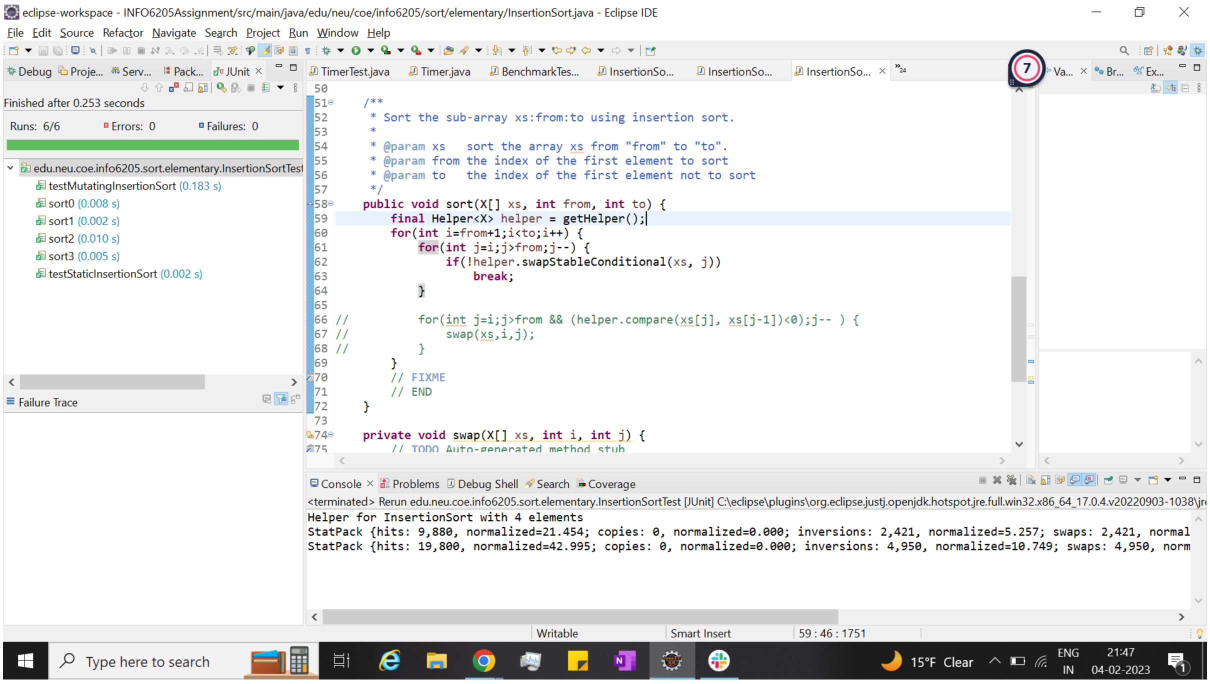
```java
        final List<Integer> list = new ArrayList<>();

        for(int i=0;i<n;i++) {
            list.add(i);
        }
        Integer[] xs = list.toArray(new Integer[0]);

        BaseHelper<Integer> helper = new BaseHelper<>("InsertionSort", xs.length, Config.load(Inser
        GenericSort<Integer> sorter = new InsertionSort<Integer>(helper);
        long startTime = System.nanoTime();
        sorter.sort(xs);
        long endTime = System.nanoTime();
        System.out.println("Time taken for sorted array for n = " + n + " is " + (endTime-startTime

    }

    private static void reverseSorted(int n) throws Exception{
        final List<Integer> list = new ArrayList<>();

        for(int i=n;i>0;i--) {
            list.add(i);
        }
        Integer[] xs = list.toArray(new Integer[0]);

        BaseHelper<Integer> helper = new BaseHelper<>("InsertionSort", xs.length, Config.load(Inser
        GenericSort<Integer> sorter = new InsertionSort<Integer>(helper);
```

```
<terminated> Rerun edu.neu.coe.info6205.util.BenchmarkTest [JUnit] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.ex
2023-02-04 21:42:09 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
```

Program Structures and Algorithms
Spring 2023(Section - 1)

(Part 2) Implement *InsertionSort* (in the *InsertionSort* class)

Code:



eclipse-workspace - INFO6205Assignment/src/main/java/edu/neu/coe/info6205/sort/elementary/InsertionSort.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help
Debug Proje... Serv... Pack... JUnit
TimerTest.java Timer.java BenchmarkTes... InsertionSo... InsertionSo... InsertionSo...
Failure Trace
Console Problems Debug Shell Search Coverage
Writable Smart Insert 59 : 46 : 1751
Type here to search
15°F Clear ENG IN 21:47 04-02-2023

Finished after 0.253 seconds

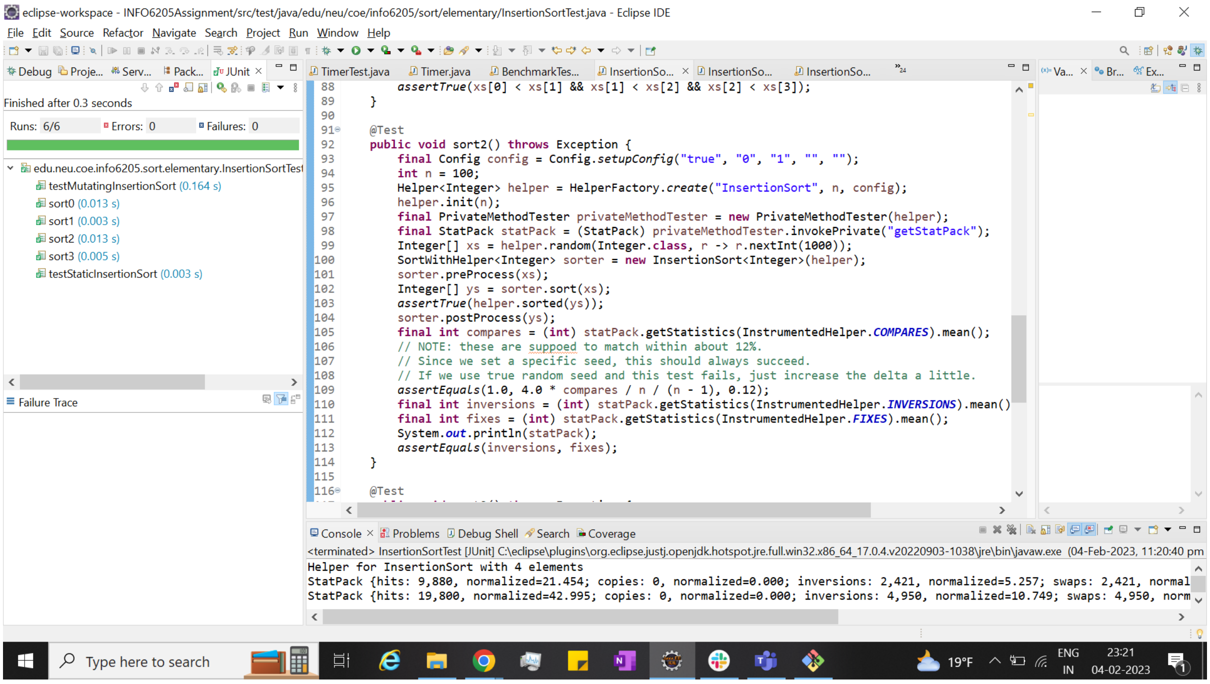
Runs: 6/6 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.sort.elementary.InsertionSortTest
    testMutatingInsertionSort (0.183 s)
    sort0 (0.008 s)
    sort1 (0.002 s)
    sort2 (0.010 s)
    sort3 (0.005 s)
    testStaticInsertionSort (0.002 s)
```

```java
    /**
     * Sort the sub-array xs:from:to using insertion sort.
     *
     * @param xs   sort the array xs from "from" to "to".
     * @param from the index of the first element to sort
     * @param to   the index of the first element not to sort
     */
    public void sort(X[] xs, int from, int to) {
        final Helper<X> helper = getHelper();
        for(int i=from+1;i<to;i++) {
            for(int j=i;j>from;j--) {
                if(!helper.swapStableConditional(xs, j))
                    break;
            }

//          for(int j=i;j>from && (helper.compare(xs[j], xs[j-1])<0);j-- ) {
//              swap(xs,i,j);
//          }
        }
        // FIXME
        // END
    }

    private void swap(X[] xs, int i, int j) {
        // TODO Auto-generated method stub
```

```
<terminated> Rerun edu.neu.coe.info6205.sort.elementary.InsertionSortTest [JUnit] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jr
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, normal
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, norm
```

Unit Test Screenshot:



eclipse-workspace - INFO6205Assignment/src/test/java/edu/neu/coe/info6205/sort/elementary/InsertionSortTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help
Debug Proje... Serv... Pack... JUnit
TimerTest.java Timer.java BenchmarkTes... InsertionSo... InsertionSo... InsertionSo...
Failure Trace
Console Problems Debug Shell Search Coverage
Type here to search
19°F ENG IN 23:21 04-02-2023

Finished after 0.3 seconds

Runs: 6/6 Errors: 0 Failures: 0

```
edu.neu.coe.info6205.sort.elementary.InsertionSortTest
    testMutatingInsertionSort (0.164 s)
    sort0 (0.013 s)
    sort1 (0.003 s)
    sort2 (0.013 s)
    sort3 (0.005 s)
    testStaticInsertionSort (0.003 s)
```

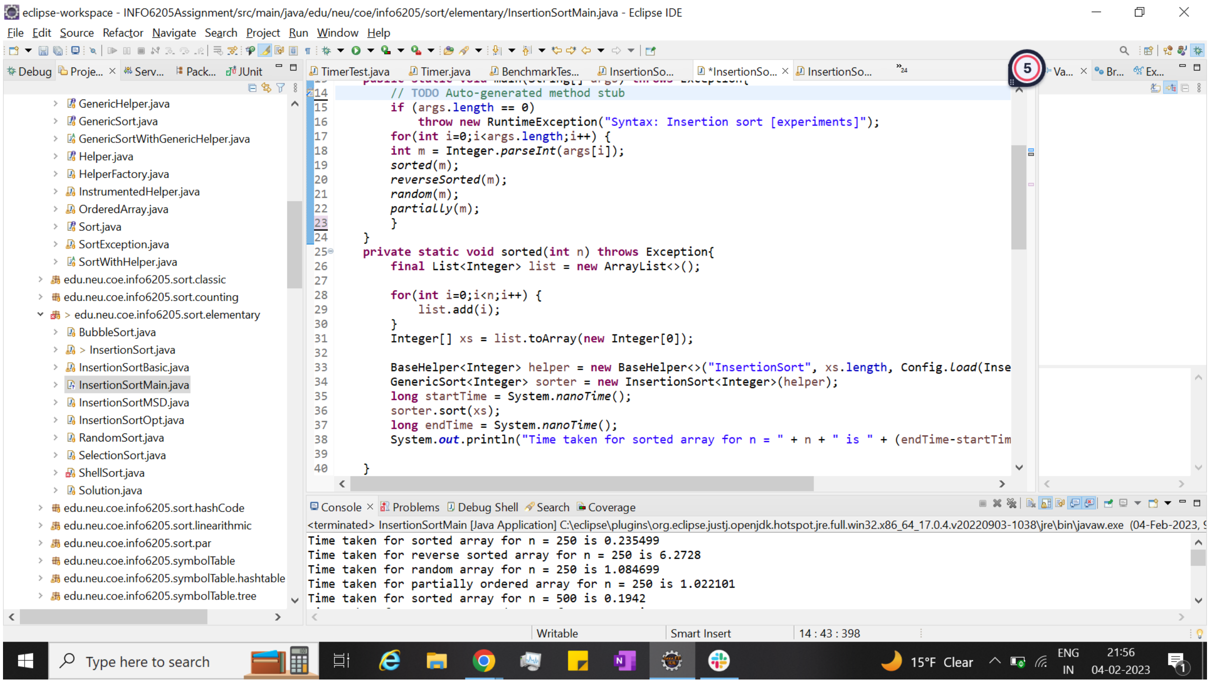
```java
        assertTrue(xs[0] < xs[1] && xs[1] < xs[2] && xs[2] < xs[3]);
    }

    @Test
    public void sort2() throws Exception {
        final Config config = Config.setupConfig("true", "0", "1", "", "");
        int n = 100;
        Helper<Integer> helper = HelperFactory.create("InsertionSort", n, config);
        helper.init(n);
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        final StatPack statPack = (StatPack) privateMethodTester.invokePrivate("getStatPack");
        Integer[] xs = helper.random(Integer.class, r -> r.nextInt(1000));
        SortWithHelper<Integer> sorter = new InsertionSort<Integer>(helper);
        sorter.preProcess(xs);
        Integer[] ys = sorter.sort(xs);
        assertTrue(helper.sorted(ys));
        sorter.postProcess(ys);
        final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
        // NOTE: these are suppoed to match within about 12%.
        // Since we set a specific seed, this should always succeed.
        // If we use true random seed and this test fails, just increase the delta a little.
        assertEquals(1.0, 4.0 * compares / n / (n - 1), 0.12);
        final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean()
        final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
        System.out.println(statPack);
        assertEquals(inversions, fixes);
    }

    @Test
```

```
<terminated> InsertionSortTest [JUnit] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe  (04-Feb-2023, 11:20:40 pm
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, normal
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, norm
```

Program Structures and Algorithms
Spring 2023(Section - 1)

(Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks:

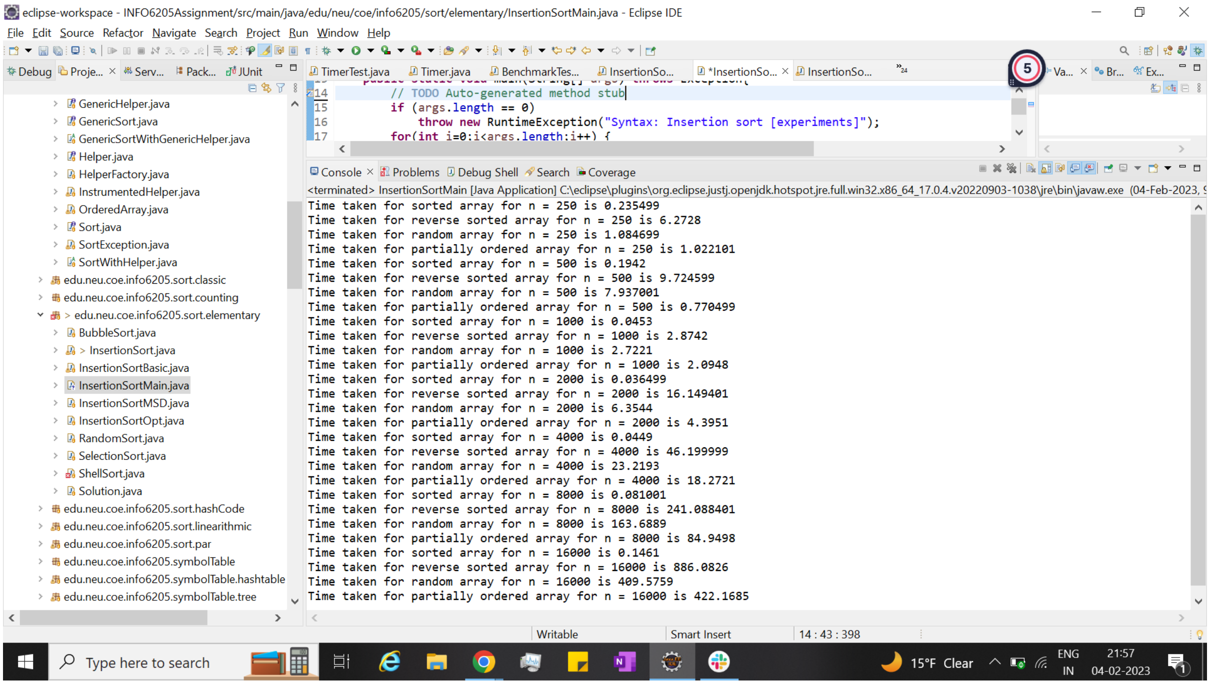
Main method:



```
        // TODO Auto-generated method stub
        if (args.length == 0)
            throw new RuntimeException("Syntax: Insertion sort [experiments]");
        for(int i=0;i<args.length;i++) {
        int m = Integer.parseInt(args[i]);
        sorted(m);
        reverseSorted(m);
        random(m);
        partially(m);
        }
    }
    private static void sorted(int n) throws Exception{
        final List<Integer> list = new ArrayList<>();

        for(int i=0;i<n;i++) {
            list.add(i);
        }
        Integer[] xs = list.toArray(new Integer[0]);

        BaseHelper<Integer> helper = new BaseHelper<>("InsertionSort", xs.length, Config.load(Inse
        GenericSort<Integer> sorter = new InsertionSort<Integer>(helper);
        long startTime = System.nanoTime();
        sorter.sort(xs);
        long endTime = System.nanoTime();
        System.out.println("Time taken for sorted array for n = " + n + " is " + (endTime-startTim

    }
```

```
<terminated> InsertionSortMain [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe (04-Feb-2023, 
Time taken for sorted array for n = 250 is 0.235499
Time taken for reverse sorted array for n = 250 is 6.2728
Time taken for random array for n = 250 is 1.084699
Time taken for partially ordered array for n = 250 is 1.022101
Time taken for sorted array for n = 500 is 0.1942
```

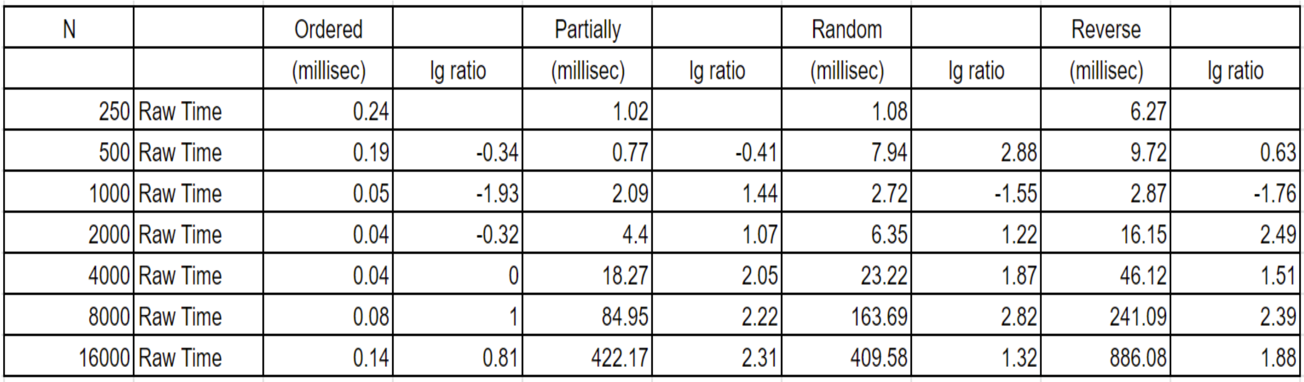
Output:



```
        // TODO Auto-generated method stub
        if (args.length == 0)
            throw new RuntimeException("Syntax: Insertion sort [experiments]");
```

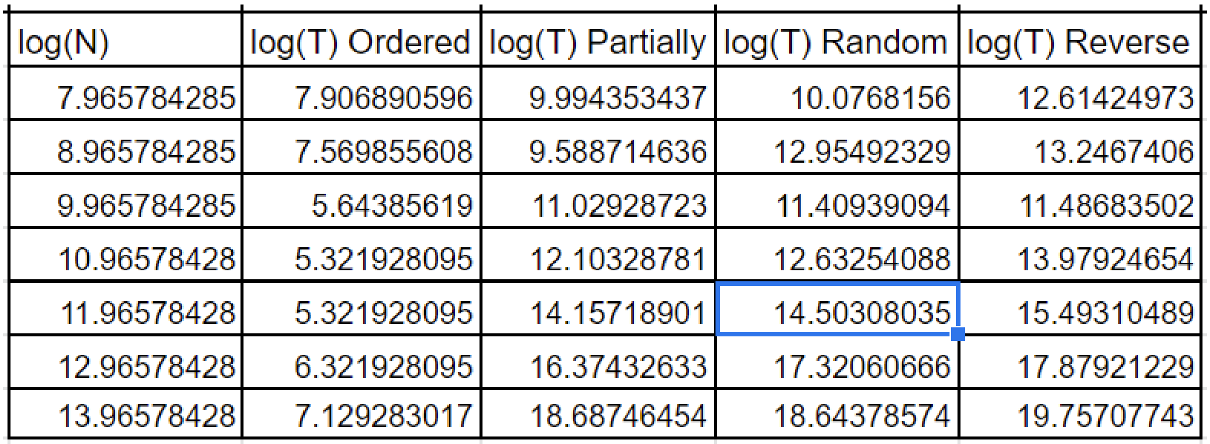
```
<terminated> InsertionSortMain [Java Application] C:\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038\jre\bin\javaw.exe (04-Feb-2023, 
Time taken for sorted array for n = 250 is 0.235499
Time taken for reverse sorted array for n = 250 is 6.2728
Time taken for random array for n = 250 is 1.084699
Time taken for partially ordered array for n = 250 is 1.022101
Time taken for sorted array for n = 500 is 0.1942
Time taken for reverse sorted array for n = 500 is 9.724599
Time taken for random array for n = 500 is 7.937001
Time taken for partially ordered array for n = 500 is 0.770499
Time taken for sorted array for n = 1000 is 0.0453
Time taken for reverse sorted array for n = 1000 is 2.8742
Time taken for random array for n = 1000 is 2.7221
Time taken for partially ordered array for n = 1000 is 2.0948
Time taken for sorted array for n = 2000 is 0.036499
Time taken for reverse sorted array for n = 2000 is 16.149401
Time taken for random array for n = 2000 is 6.3544
Time taken for partially ordered array for n = 2000 is 4.3951
Time taken for sorted array for n = 4000 is 0.0449
Time taken for reverse sorted array for n = 4000 is 46.199999
Time taken for random array for n = 4000 is 23.2193
Time taken for partially ordered array for n = 4000 is 18.2721
Time taken for sorted array for n = 8000 is 0.081001
Time taken for reverse sorted array for n = 8000 is 241.088401
Time taken for random array for n = 8000 is 163.6889
Time taken for partially ordered array for n = 8000 is 84.9498
Time taken for sorted array for n = 16000 is 0.1461
Time taken for reverse sorted array for n = 16000 is 886.0826
Time taken for random array for n = 16000 is 409.5759
Time taken for partially ordered array for n = 16000 is 422.1685
```

(b)Validating by using doubling benchmark:
Here, I timed the observations using the doubling method from n=250 until n=16000



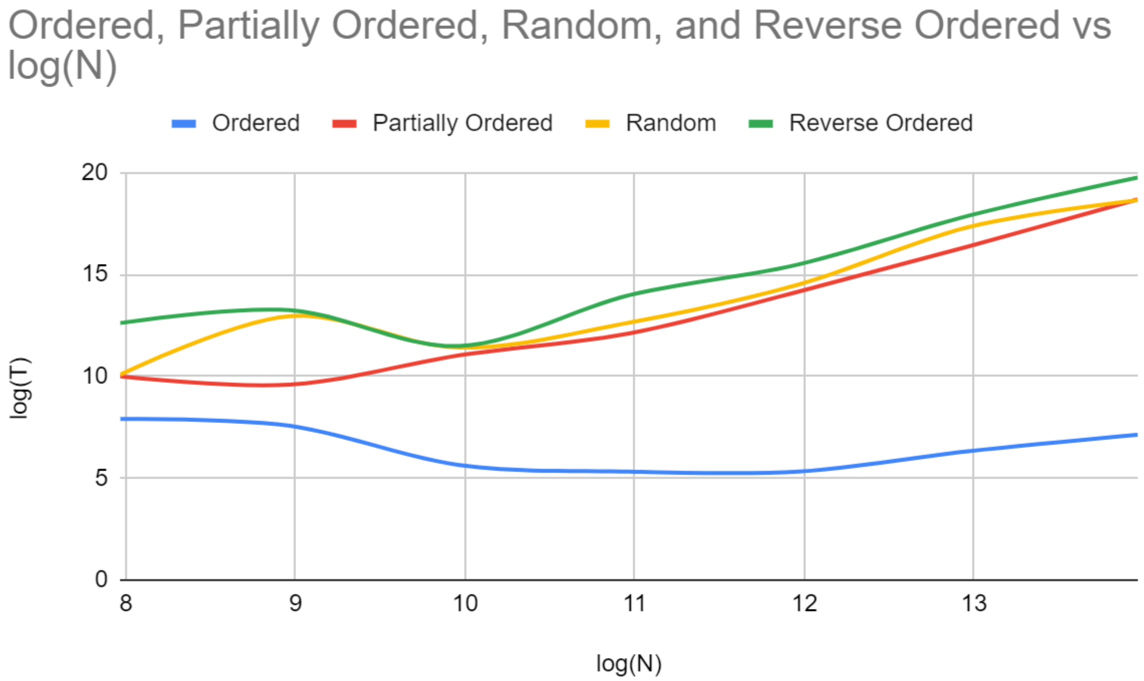
N		Ordered		Partially		Random		Reverse	
		(millisec)	lg ratio	(millisec)	lg ratio	(millisec)	lg ratio	(millisec)	lg ratio
250	Raw Time	0.24		1.02		1.08		6.27	
500	Raw Time	0.19	-0.34	0.77	-0.41	7.94	2.88	9.72	0.63
1000	Raw Time	0.05	-1.93	2.09	1.44	2.72	-1.55	2.87	-1.76
2000	Raw Time	0.04	-0.32	4.4	1.07	6.35	1.22	16.15	2.49
4000	Raw Time	0.04	0	18.27	2.05	23.22	1.87	46.12	1.51
8000	Raw Time	0.08	1	84.95	2.22	163.69	2.82	241.09	2.39
16000	Raw Time	0.14	0.81	422.17	2.31	409.58	1.32	886.08	1.88

log(N) vs log(T) values for the above observations. Here, before applying the log(T), I multiplied every value with 10^3 in order to avoid negative values, as the log of something less than 1 is negative.



log(N)	log(T) Ordered	log(T) Partially	log(T) Random	log(T) Reverse
7.965784285	7.906890596	9.994353437	10.0768156	12.61424973
8.965784285	7.569855608	9.588714636	12.95492329	13.2467406
9.965784285	5.64385619	11.02928723	11.40939094	11.48683502
10.96578428	5.321928095	12.10328781	12.63254088	13.97924654
11.96578428	5.321928095	14.15718901	14.50308035	15.49310489
12.96578428	6.321928095	16.37432633	17.32060666	17.87921229
13.96578428	7.129283017	18.68746454	18.64378574	19.75707743

Here’s the graph of log(N) vs log(T) for four different initial array ordering situations: random, ordered, partially ordered, and reverse ordered



Conclusion:

The order of growth is not very much impacted when the input array is sorted, while it increases with the size of the array in all other cases when the input array is partially sorted, random, or reverse ordered. It is always higher in the case of reverse ordered as it involves the highest number of swaps and a little lower for partially sorted and random array as it doesn’t involve as many swaps as required by reverse ordered array.