

Zixuan.Zhu (002196924)

Program Structures & Algorithms

Fall 2021

Assignment No. 2

◉ Task

Your task for this assignment is in three parts.

(Part 1) You are to implement three methods of a class called Timer. Please see the skeleton class that I created in the repository. Timer is invoked from a class called Benchmark_Timer which implements the Benchmark interface. The APIs of these class are as follows:

```
public interface Benchmark<T> {
    default double run(T t, int m) {
        return runFromSupplier(() -> t, m);
    }

    double runFromSupplier(Supplier<T> supplier, int m);
}

public class Benchmark_Timer<T> implements Benchmark<T> {
    public Benchmark_Timer(String description, UnaryOperator<T> fPre, Consumer<T> fRun, Consumer<T> fPost)

    public Benchmark_Timer(String description, UnaryOperator<T> fPre, Consumer<T> fRun)

    public Benchmark_Timer(String description, Consumer<T> fRun, Consumer<T> fPost)

    public Benchmark_Timer(String description, Consumer<T> f)

    public class Timer {
        ... // see below for methods to be implemented...
    }
}
```

```

public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> func
tion, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}

private static long getClock() {
// TO BE IMPLEMENTED
}

private static double toMillisecs(long ticks) {
// TO BE IMPLEMENTED
}

```

The function to be timed, hereinafter the "target" function, is the Consumer function fRun (or just f) passed in to one or other of the constructors. For example, you might create a function which sorts an array with n elements.

The generic type T is that of the input to the target function.

The first parameter to the first run method signature is the parameter that will, in turn, be passed to target function. In the second signature, supplier will be invoked each time to get a t which is passed to the other run method.

The second parameter to the run function (m) is the number of times the target function will be called.

The return value from run is the average number of milliseconds taken for each run of the target function.

Don't forget to check your implementation by running the unit tests in BenchmarkTest and TimerTest. If you have trouble with the exact timings in the unit tests, it's quite OK (in this assignment only) to change parameters until the tests run. Different machine architectures will result in different behavior.

- (Part 2) Implement InsertionSort (in the InsertionSort class) by simply looking up the insertion code used by Arrays.sort. If you have the instrument = true setting in test/resources/config.ini, then you will need to use the helper methods for comparing and swapping (so that they properly

count the number of swaps/compares). The easiest is to use the `helper.swapStableConditional` method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in `InsertionSortTest`.

- (Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to be sorted are of type `Integer`. Use the doubling method for choosing `n` and test for at least five values of `n`. Draw any conclusions from your observations regarding the order of growth.

As usual, the submission will be your entire project (clean, i.e. without the target and project folders). There are stubs and unit tests in the repository.

Report on your observations and show screenshots of the runs and also the unit tests. Please note that you may have to adjust the required execution time for the insertion sort unit test(s) because your computer may not run at the same speed as mine.

Further notes: you should use the `System.nanoTime` method to get the clock time. This isn't guaranteed to be accurate which is one of the reasons you should run the experiment several times for each value of `n`. Also, for each invocation of run, run the given target function ten times to get the system "warmed up" before you start the timing properly.

The `Sort` interface takes care of copying the array when the `sort(array)` signature is called. It returns a new array as a result. The original array is unchanged. Therefore, you do not need to worry about the insertion-based sorts getting quicker because of the arrays getting more sorted (they don't).

◉ Relationship Conclusion:

Comparison of the running time of four differently sorted but same length(n) arrays (random, ordered, partially-ordered and reverse-ordered) when using insertion sort:

reverse-ordered > random > partially-ordered > ordered

("partially-ordered" in this assignment means that array is ordered from "array.length/2" to "array.length")

◉ Evidence to support the conclusion:

Github address:

Timer:

<https://github.com/ZixuanZhu-faye/INFO-6205/blob/e60c0539e66af2048d8e425d4d26703827c0c691/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/util/Timer.java>

InsertionSort:

<https://github.com/ZixuanZhu-faye/INFO-6205/blob/e60c0539e66af2048d8e425d4d26703827c0c691/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/elementary/InsertionSort.java>

Benchmark_Timer:

https://github.com/ZixuanZhu-faye/INFO-6205/blob/e60c0539e66af2048d8e425d4d26703827c0c691/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/util/Benchmark_Timer.java

1. Output

B-E-G-I-N

Round 1

Random: $n = 40$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:1.3333333333333333

meantime:1.0333333333333334

Meantime: 1.0333333333333334

Partially-ordered: $n = 40$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.6666666666666666

meantime:0.8333333333333334

Meantime: 0.8333333333333334

Ordered: $n = 40$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.6666666666666666
meantime:0.7333333333333333
Meantime: 0.7333333333333333
Reverse-ordered n = 40
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.6666666666666666
meantime:0.7333333333333333
Mean lap time: 0.7333333333333333
Round 2
Random: n = 80
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.5666666666666667
Meantime: 0.5666666666666667
Partially-ordered: n = 80
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.7333333333333333
Meantime: 0.7333333333333333
Ordered: n = 80
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.3333333333333333
Meantime: 0.3333333333333333
Reverse-ordered n = 80
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.5
Mean lap time: 0.5
Round 3
Random: n = 160
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.5
Meantime: 0.5
Partially-ordered: n = 160
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.36666666666666664
Meantime: 0.36666666666666664
Ordered: n = 160
2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:0.3333333333333333
meantime:0.8

Meantime: 0.8

Reverse-ordered $n = 160$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.3333333333333333

meantime:0.4666666666666667

Mean lap time: 0.4666666666666667

Round 4

Random: $n = 320$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.3333333333333333

meantime:0.5333333333333333

Meantime: 0.5333333333333333

Partially-ordered: $n = 320$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.3333333333333333

meantime:0.3666666666666664

Meantime: 0.3666666666666664

Ordered: $n = 320$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.0

meantime:0.3333333333333333

Meantime: 0.3333333333333333

Reverse-ordered $n = 320$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.6666666666666666

meantime:0.9666666666666667

Mean lap time: 0.9666666666666667

Round 5

Random: $n = 640$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:1.3333333333333333

meantime:0.9666666666666667

Meantime: 0.9666666666666667

Partially-ordered: $n = 640$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.6666666666666666

meantime:0.6

Meantime: 0.6

Ordered: $n = 640$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.3333333333333333

meantime:0.6

Meantime: 0.6

Reverse-ordered $n = 640$

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs
meantime:1.0

meantime:1.2666666666666666

Mean lap time: 1.2666666666666666

Round 6

Random: n = 1280

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:1.6666666666666667

meantime:1.9666666666666666

Meantime: 1.9666666666666666

Partially-ordered: n = 1280

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:1.3333333333333333

meantime:2.2666666666666666

Meantime: 2.2666666666666666

Ordered: n = 1280

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.6666666666666666

meantime:0.23333333333333334

Meantime: 0.23333333333333334

Reverse-ordered n = 1280

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:3.6666666666666665

meantime:5.033333333333333

Mean lap time: 5.033333333333333

Round 7

Random: n = 2560

2021-09-26 16:59:17 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:6.666666666666667

meantime:7.633333333333334

Meantime: 7.633333333333334

Partially-ordered: n = 2560

2021-09-26 16:59:18 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:4.666666666666667

meantime:5.166666666666667

Meantime: 5.166666666666667

Ordered: n = 2560

2021-09-26 16:59:18 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.0

meantime:0.13333333333333333

Meantime: 0.13333333333333333

Reverse-ordered n = 2560

2021-09-26 16:59:18 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:13.0

meantime:12.7

Mean lap time: 12.7

Round 8

Random: n = 5120

2021-09-26 16:59:18 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:26.333333333333332

meantime:29.833333333333332

Meantime: 29.833333333333332

Partially-ordered: n = 5120

2021-09-26 16:59:19 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:20.666666666666668

meantime:22.766666666666666

Meantime: 22.766666666666666

Ordered: n = 5120

2021-09-26 16:59:20 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.0

meantime:0.13333333333333333

Meantime: 0.13333333333333333

Reverse-ordered n = 5120

2021-09-26 16:59:20 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:63.0

meantime:62.266666666666666

Mean lap time: 62.266666666666666

Round 9

Random: n = 10240

2021-09-26 16:59:22 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:154.0

meantime:123.2

Meantime: 123.2

Partially-ordered: n = 10240

2021-09-26 16:59:26 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:108.0

meantime:78.866666666666666

Meantime: 78.866666666666666

Ordered: n = 10240

2021-09-26 16:59:29 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.0

meantime:0.2

Meantime: 0.2

Reverse-ordered n = 10240

2021-09-26 16:59:29 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:216.0

meantime:206.666666666666666

Mean lap time: 206.666666666666666

Round 10

Random: n = 20480

2021-09-26 16:59:36 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:442.3333333333333

meantime:449.6666666666667

Meantime: 449.6666666666667

Partially-ordered: n = 20480

2021-09-26 16:59:51 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:336.0

meantime:324.73333333333335

Meantime: 324.73333333333335

Ordered: n = 20480

2021-09-26 17:00:02 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:0.3333333333333333

meantime:0.23333333333333334

Meantime: 0.23333333333333334

Reverse-ordered n = 20480

2021-09-26 17:00:02 INFO Benchmark_Timer - Begin run: InsertionSort Time with 30 runs

meantime:967.3333333333334

meantime:956.1

Mean lap time: 956.1

E-N-D

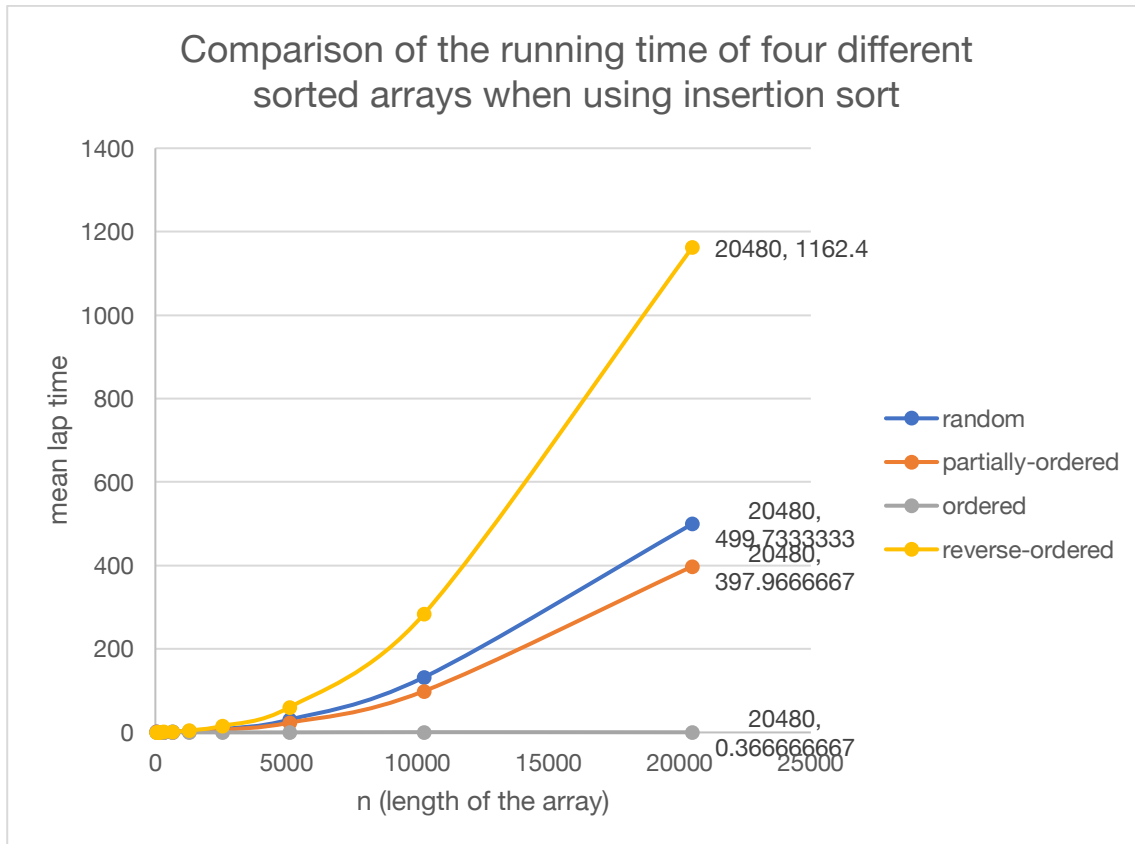
2. Graphical Representation

Data:

n	random	partially-ordered	ordered	reverse-ordered
40	1.366666667	1.333333333	0.8	0.733333333
80	0.7	1	0.8	0.633333333
160	0.8	0.766666667	0.5	0.7
320	0.766666667	0.7	0.633333333	1.066666667
640	1.066666667	0.966666667	0.466666667	1.6
1280	3.633333333	2.133333333	0.266666667	4.8
2560	8.266666667	6.3	0.133333333	15.66666667
5120	30.8	23.36666667	0.166666667	60.46666667
10240	132.1	98.93333333	0.666666667	283.4666667
20480	499.7333333	397.9666667	0.366666667	1162.4

“n” is the number of elements of the array.

Graph:



◦ Unit tests result:

BenchmarkTest:

The screenshot shows an IDE with the Package Explorer on the left, the BenchmarkTest.java file open in the center, and the Console at the bottom. The Package Explorer shows the test results for BenchmarkTest, including testWaitPeriods (1.519 s) and getWarmupRuns (0.001 s). The Console shows the output of the test, including the time taken for the test and the results of the assertions.

```
20 * Copyright (c) 2017. Phasmid Software[]
4
5 package edu.neu.coe.info6205.util;
6
7 import org.junit.Test;
8
9
10
11 @SuppressWarnings("ALL")
12 public class BenchmarkTest {
13
14     int pre = 0;
15     int run = 0;
16     int post = 0;
17
18     @Test // Slow
19     public void testWaitPeriods() throws Exception {
20         int nRuns = 2;
21         int warmups = 2;
22         Benchmark_Timer<Boolean> bm = new Benchmark_Timer<>("testWaitPeriods", b -> {
23             GoToSleep(100L, -1);
24             return null;
25         });
26         b -> {
27             GoToSleep(200L, 0);
28         };
29         b -> {
30             GoToSleep(50L, 1);
31         };
32     };
33     double x = bm.run(true, nRuns);
34     assertEquals(nRuns, post);
35     assertEquals(nRuns + warmups, run);
36     assertEquals(nRuns + warmups, pre);
37     assertEquals(200, x, 10);
38 }
39
40 private void GoToSleep(long mSecs, int which) {
41     try {
```

Console output:

```
<terminated> BenchmarkTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (2021年9月26日 下午5:47:50)
2021-09-26 17:47:50 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
meantime:202.0
meantime:202.5
```

InsertionSortTest:

The screenshot shows an IDE with the Package Explorer on the left, the InsertionSortTest.java file open in the center, and the Console at the bottom. The Package Explorer shows the test results for InsertionSortTest, including testMutatingInsertionSort (0.117 s), sort0 (0.087 s), sort1 (0.002 s), sort2 (0.007 s), sort3 (0.003 s), and testStaticInsertionSort (0.003 s). The Console shows the output of the test, including the time taken for the test and the results of the assertions.

```
69 Integer[] xs = list.toArray(new Integer[0]);
70 BaseHelper<Integer> helper = new BaseHelper<>("InsertionSort", xs.length, Config.load(InsertionSortTest.class));
71 GenericSort<Integer> sorter = new InsertionSort<Integer>(helper);
72 sorter.mutatingSort(xs);
73 assertTrue(helper.sorted(xs));
74
75
76 @Test
77 public void testStaticInsertionSort() throws IOException {
78     final List<Integer> list = new ArrayList<>();
79     list.add(3);
80     list.add(4);
81     list.add(2);
82     list.add(1);
83     Integer[] xs = list.toArray(new Integer[0]);
84     InsertionSort.sort(xs);
85
86     //
87 }
88
89 public void sort(X[] xs, int from, int to) {
90     final Helper<X> helper = getHelper();
91     // TO BE IMPLEMENTED
92
93     for(int i = from+1; i<to; i++){
94         for(int j = i; j > from; j--){
95             if (helper.compare(xs[j-1], xs[j])>0){
96                 helper.swap(xs, j-1, j);
97             }
98             else
99                 break;
100         }
101     }
102 }
```

Console output:

```
<terminated> InsertionSortTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (2021年9月26日 下午4:02:28)
2021-09-26 16:02:29 DEBUG Config - Config.get(helper, instrument) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(helper, seed) = 0
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, copies) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, compares) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(instrumenting, hits) = true
2021-09-26 16:02:29 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684; copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
StatPack {hits: 19,800; copies: 0; inversions: 4,950; swaps: 4,950; fixes: 4,950; compares: 4,950}
```

TimerTest:

The screenshot displays an IDE window titled "2021fall - Java - INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/util/TimerTest.java - Ec". The interface is divided into three main sections:

- Package Explorer (Left):** Shows the test results for `edu.neu.coe.info6205.util.TimerTest` [Runner: JUnit 4] (2.157 s). The results list several test methods and their durations:
 - `testPauseAndLapResume0` (0.170 s)
 - `testPauseAndLapResume1` (0.309 s)
 - `testLap` (0.206 s)
 - `testPause` (0.204 s)
 - `testStop` (0.105 s)
 - `testMillisecs` (0.105 s)
 - `testRepeat1` (0.166 s)
 - `testRepeat2` (0.227 s)
 - `testRepeat3` (0.563 s)
 - `testPauseAndLap` (0.102 s)
- Source Code (Right):** Displays the code for `TimerTest.java`. The code includes:
 - Line 76: `final Timer timer = new Timer();`
 - Line 77: `GoToSleep(TENTH, 0);`
 - Line 78: `timer.pause();`
 - Line 79: `GoToSleep(TENTH, 0);`
 - Line 80: `timer.resume();`
 - Line 81: `final double time = timer.stop();`
 - Line 82: `assertEquals(TENTH_DOUBLE, time, 10.0);`
 - Line 83: `assertEquals(2, run);`
 - Line 84: `}`
 - Line 86: `@Test`
 - Line 87: `public void testMillisecs() {`
 - Line 88: `final Timer timer = new Timer();`
 - Line 89: `GoToSleep(TENTH, 0);`
 - Line 90: `timer.stop();`
 - Line 91: `final double time = timer.millisecs();`
 - Line 92: `assertEquals(TENTH_DOUBLE, time, 10.0);`
 - Line 93: `assertEquals(1, run);`
 - Line 94: `}`
 - Line 96: `@Test`
 - Line 97: `public void testRepeat1() {`
 - Line 98: `final Timer timer = new Timer();`
 - Line 99: `final double mean = timer.repeat(10, () -> {`
 - Line 100: `GoToSleep(HUNDREDTH, 0);`
 - Line 101: `return null;`
 - Line 102: `});`
 - Line 103: `assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));`
 - Line 104: `assertEquals(TENTH_DOUBLE / 10, mean, 20);`
 - Line 105: `assertEquals(10, run);`
 - Line 106: `assertEquals(0, pre);`
 - Line 107: `assertEquals(0, post);`
 - Line 108: `}`
- Console (Bottom):** Shows the output of the test run:
 - `<terminated> TimerTest [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java (2021年9月26日 下午4:22:22)`
 - `meantime:22.3`
 - `meantime:23.3`