

Program Structures & Algorithms

Spring 2022

Assignment No. 2

Benchmark - Insertion Sort

Thomas John
NEU ID: 002933800

Task

- (Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) 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);  
}
```

[*Supplier* is a Java function type which supplies values of type *T* using the method: *get()*.]

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

[*Consumer*<*T*> is a Java function type which consumes a type *T* with the method: *accept*(*t*).

UnaryOperator<*T*> is essentially an alias of *Function*<*T*, *T*> which defines *apply*(*t*) which takes a *T* and returns a *T*.]

```
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>
    function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    // TO BE IMPLEMENTED
}
```

[*Function*<*T*, *U*> defines a method *U apply*(*t*: *T*), which takes a value of *T* and returns a value of *U*.]

```
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.

Output Screenshot

```
Run: Unnamed
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
2022-02-12 17:15:10 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 2000 with 50 runs
2022-02-12 17:15:10 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 2000 with 50 runs
2022-02-12 17:15:11 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 2000 with 50 runs
2022-02-12 17:15:11 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 2000 with 50 runs
@@@--- *****
Integers to be sorted = 2000
Average time taken for : Sorted Array = 0.04 ms
Average time taken for : Random Array = 2.9 ms
Average time taken for : Partially Ordered Array = 2.2 ms
Average time taken for : Reversed Ordered Array = 5.76 ms
@@@--- *****
2022-02-12 17:15:11 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 4000 with 50 runs
2022-02-12 17:15:11 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 4000 with 50 runs
2022-02-12 17:15:12 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 4000 with 50 runs
2022-02-12 17:15:12 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 4000 with 50 runs
@@@--- *****
Integers to be sorted = 4000
Average time taken for : Sorted Array = 0.0 ms
Average time taken for : Random Array = 11.82 ms
Average time taken for : Partially Ordered Array = 9.22 ms
Average time taken for : Reversed Ordered Array = 23.38 ms
@@@--- *****
2022-02-12 17:15:14 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 8000 with 50 runs
2022-02-12 17:15:14 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 8000 with 50 runs
2022-02-12 17:15:16 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 8000 with 50 runs
2022-02-12 17:15:18 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 8000 with 50 runs
@@@--- *****
Integers to be sorted = 8000
Average time taken for : Sorted Array = 0.02 ms
Average time taken for : Random Array = 46.5 ms
Average time taken for : Partially Ordered Array = 35.1 ms
Average time taken for : Reversed Ordered Array = 93.22 ms
```

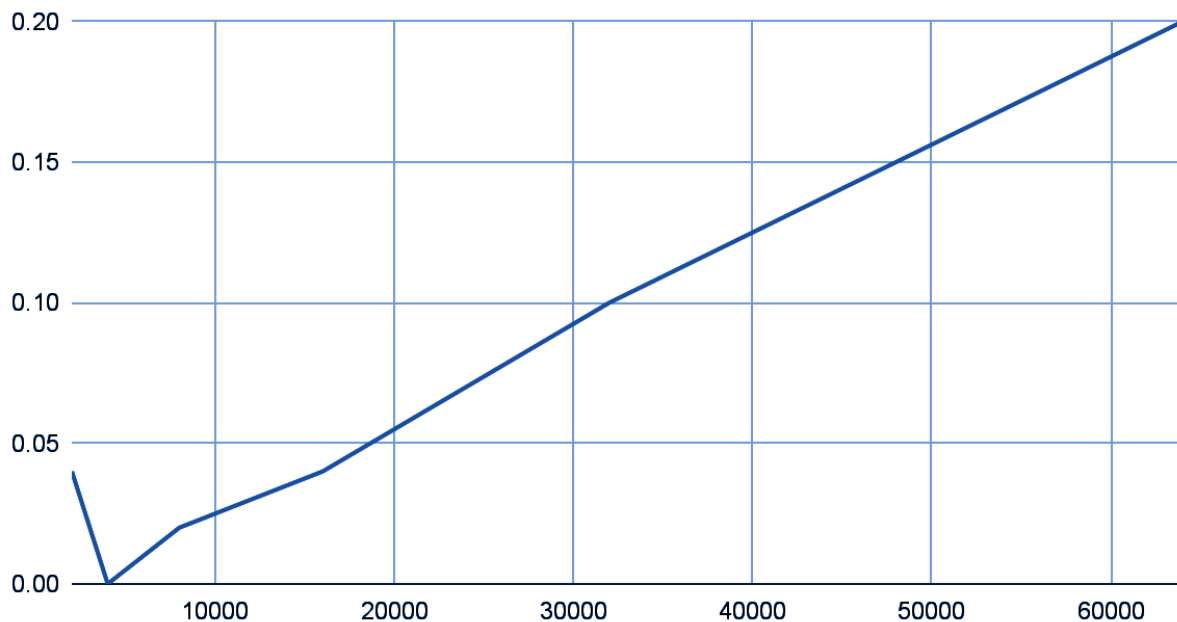
```

Average time taken for : Reversed Ordered Array = 70.22 ms
@@@--- *****
2022-02-12 17:15:23 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 16000 with 50 runs
2022-02-12 17:15:23 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 16000 with 50 runs
2022-02-12 17:15:34 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 16000 with 50 runs
2022-02-12 17:15:42 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 16000 with 50 runs
@@@--- *****
Integers to be sorted = 16000
Average time taken for : Sorted Array = 0.04 ms
Average time taken for : Random Array = 194.18 ms
Average time taken for : Partially Ordered Array = 141.72 ms
Average time taken for : Reversed Ordered Array = 375.68 ms
@@@--- *****
2022-02-12 17:16:02 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 32000 with 50 runs
2022-02-12 17:16:02 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 32000 with 50 runs
2022-02-12 17:16:48 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 32000 with 50 runs
2022-02-12 17:17:19 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 32000 with 50 runs
@@@--- *****
Integers to be sorted = 32000
Average time taken for : Sorted Array = 0.1 ms
Average time taken for : Random Array = 827.04 ms
Average time taken for : Partially Ordered Array = 575.16 ms
Average time taken for : Reversed Ordered Array = 1520.72 ms
@@@--- *****
2022-02-12 17:18:43 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 64000 with 50 runs
2022-02-12 17:18:43 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 64000 with 50 runs
2022-02-12 17:22:08 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 64000 with 50 runs
2022-02-12 17:24:20 INFO Benchmark_Timer - Begin run: Benchmarking for insertion Sort: No. of Integers = 64000 with 50 runs
@@@--- *****
Integers to be sorted = 64000
Average time taken for : Sorted Array = 0.22 ms
Average time taken for : Random Array = 3714.32 ms
Average time taken for : Partially Ordered Array = 2391.28 ms
Average time taken for : Reversed Ordered Array = 6180.04 ms
@@@--- *****

Process finished with exit code 0
```

Relationship Conclusion

Insertion Sort - Sorted Array



Average time taken for 2000 items : Sorted Array = 0.04 ms

Average time taken for 4000 items: Sorted Array = 0.0 ms

Average time taken for 8000: Sorted Array = 0.02 ms

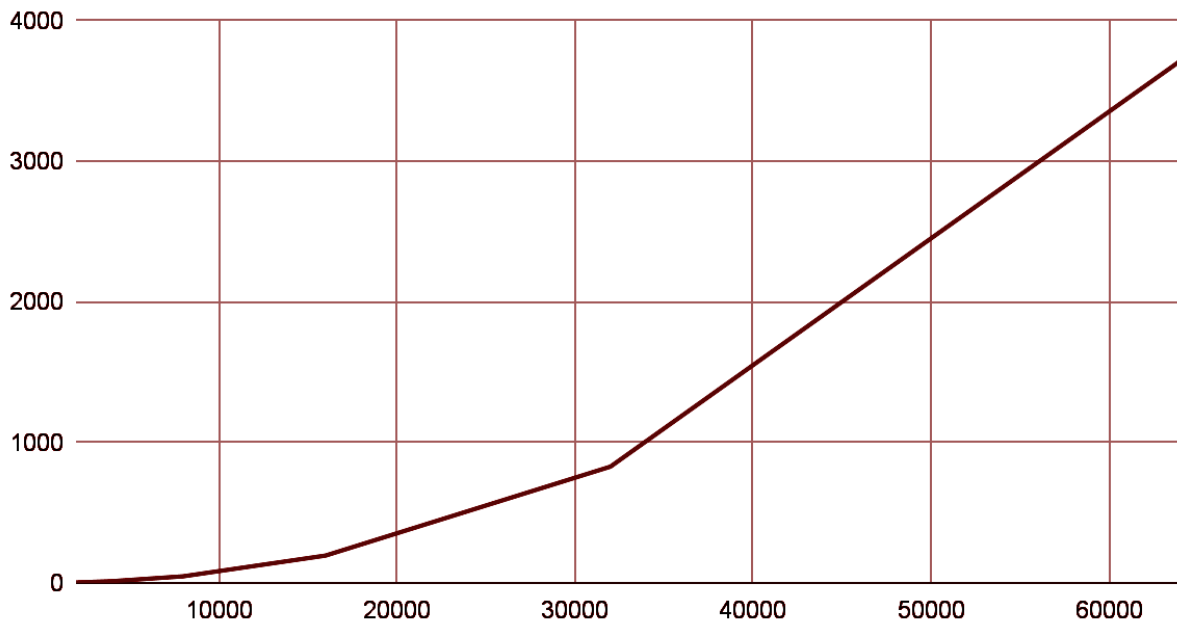
Average time taken for 16000: Sorted Array = 0.04 ms

Average time taken for 32000: Sorted Array = 0.1 ms

Average time taken for 64000: Sorted Array = 0.22 ms

Best case Time Complexity for Insertion Sort = $O(n)$

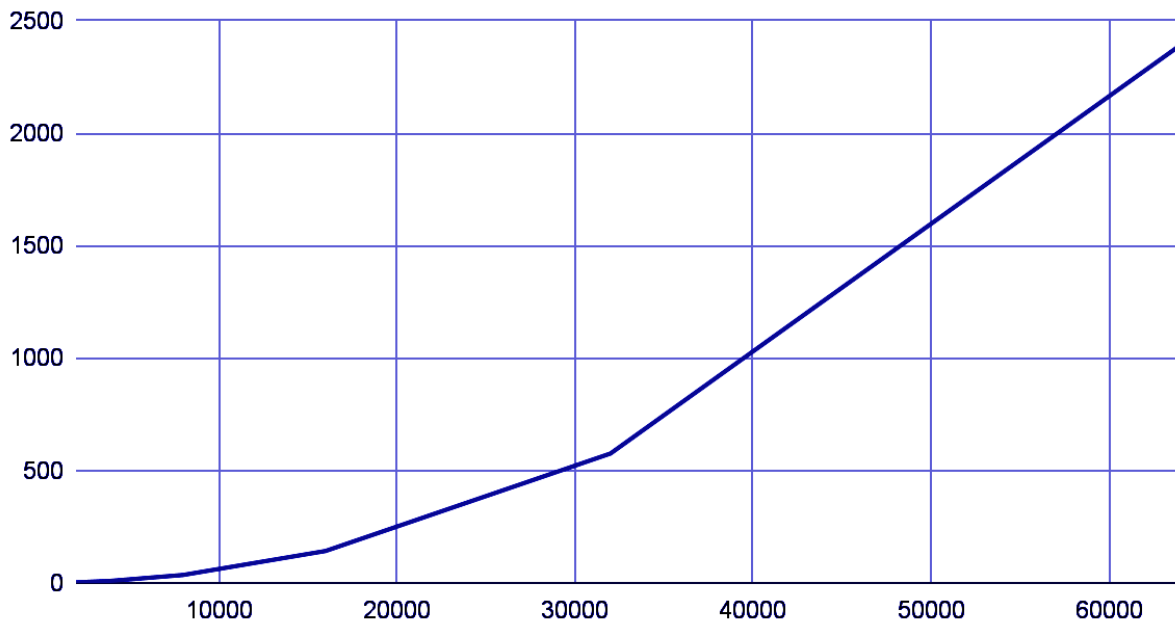
Insertion Sort - Unsorted Array



Average time taken for 2000 items: Random Array = 2.9 ms
Average time taken for 4000 items: Random Array = 11.82 ms
Average time taken for 8000 items: Random Array = 46.5 ms
Average time taken for 16000 items: Random Array = 194.18 ms
Average time taken for 32000 items: Random Array = 827.04 ms
Average time taken for 64000 items: Random Array = 3714.32 ms

Time Complexity for randomly sorted array for Insertion Sort = $O(n^2)$

Insertion Sort - Partially Sorted Array



Average time taken for 2000 items: Partially Ordered Array = 2.2 ms

Average time taken for 4000 items: Partially Ordered Array = 9.22 ms

Average time taken for 8000 items: Partially Ordered Array = 35.1 ms

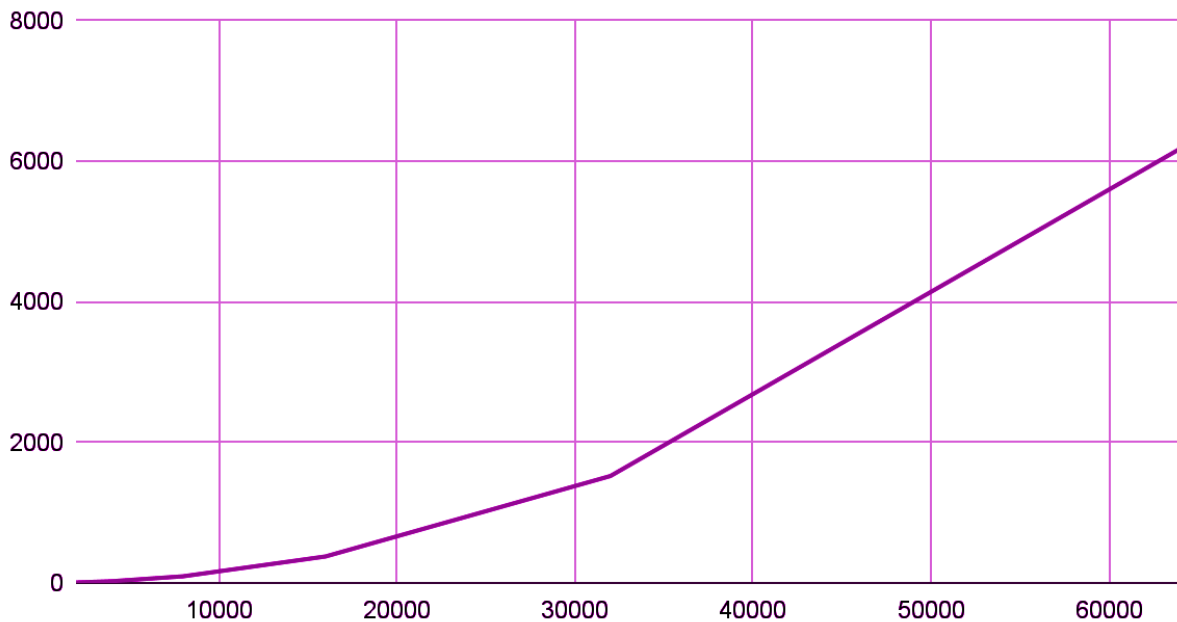
Average time taken for 16000 items: Partially Ordered Array = 141.72 ms

Average time taken for 32000 items: Partially Ordered Array = 575.16 ms

Average time taken for 64000 items: Partially Ordered Array = 2391.28 ms

Time Complexity for partially sorted array for Insertion Sort = $O(n^2)$

Insertion Sort - Reverse Sorted Array

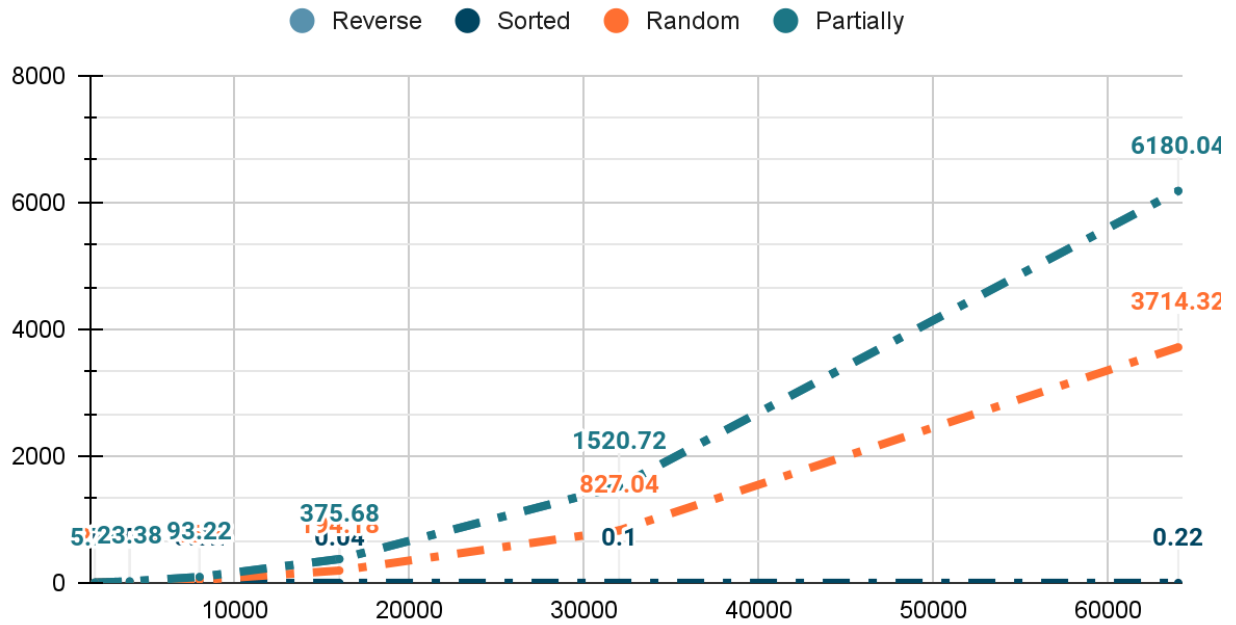


Average time taken for 2000 items: Reversed Ordered Array = 5.76 ms
Average time taken for 4000 items: Reversed Ordered Array = 23.38 ms
Average time taken for 8000 items: Reversed Ordered Array = 93.22 ms
Average time taken for 16000 items: Reversed Ordered Array = 375.68 ms
Average time taken for 32000 items: Reversed Ordered Array = 1520.72 ms
Average time taken for 64000 items: Reversed Ordered Array = 6180.04 ms

Worst-case Time Complexity for reverse sorted array for Insertion Sort = $O(n^2)$

Evidence/Graph

Insertion Sort



Code Repository

Unit Test Results

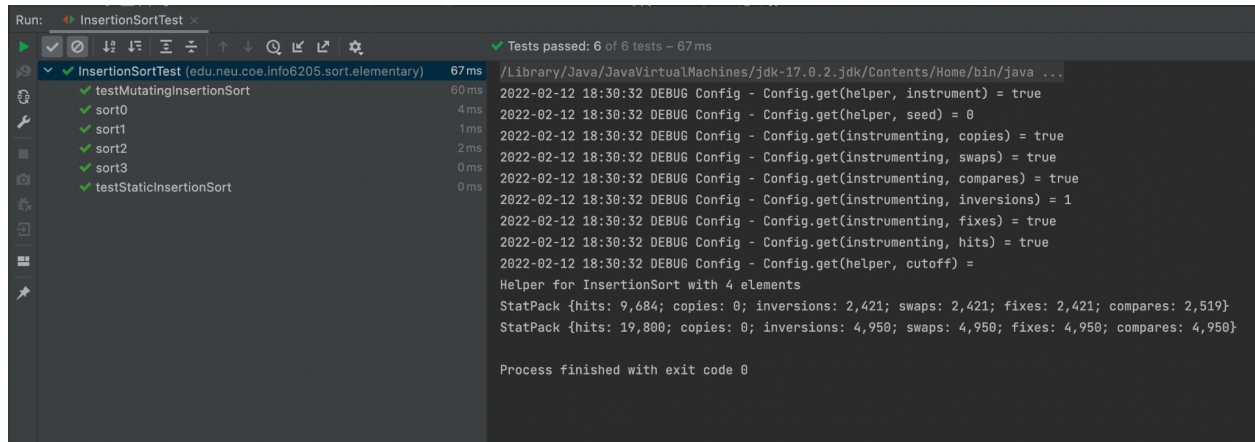
TimerTest

Run: TimerTest x			✓ Tests passed: 10 of 10 tests – 2 sec 158 ms
	✓ TimerTest (edu.neu.coe.info6205.util)	2 sec 158 ms	/Library/Java/JavaVirtualMachines/jdk-17.0.2.
	✓ testPauseAndLapResume0	156 ms	Process finished with exit code 0
	✓ testPauseAndLapResume1	313 ms	
	✓ testLap	205 ms	
	✓ testPause	211 ms	
	✓ testStop	103 ms	
	✓ testMillisecs	106 ms	
	✓ testRepeat1	119 ms	
	✓ testRepeat2	234 ms	
	✓ testRepeat3	606 ms	
	✓ testPauseAndLap	105 ms	

BenchmarkTest

Run: BenchmarkTest x			✓ Tests passed: 2 of 2 tests – 1 sec 396 ms
	✓ BenchmarkTest (edu.neu.coe.info6205.util)	1 sec 396 ms	/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
	✓ testWaitPeriods	1 sec 396 ms	2022-02-12 18:29:35 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
	✓ getWarmupRuns	0 ms	Process finished with exit code 0

InsertionSortTest



The screenshot shows the Run window of an IDE with the following details:

- Run:** InsertionSortTest
- Tests passed:** 6 of 6 tests – 67 ms
- Test Results Table:**

Test Name	Duration
InsertionSortTest (edu.neu.coe.info6205.sort.elementary)	67 ms
testMutatingInsertionSort	60 ms
sort0	4 ms
sort1	1 ms
sort2	2 ms
sort3	0 ms
testStaticInsertionSort	0 ms
- Debug Output:**

```
2022-02-12 18:30:32 DEBUG Config - Config.get(helper, instrument) = true
2022-02-12 18:30:32 DEBUG Config - Config.get(helper, seed) = 0
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, copies) = true
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, swaps) = true
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, compares) = true
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, inversions) = 1
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, fixes) = true
2022-02-12 18:30:32 DEBUG Config - Config.get(instrumenting, hits) = true
2022-02-12 18:30:32 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}

Process finished with exit code 0
```