

Deep Shah (002766755)
Program Structure & Algorithms
Spring 2023(Sec 03)

Assignment-3

Task:

There are three task to be performed.

Task list:

- (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.
- (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.

Relationship Conclusion:

Using the doubling method and with the help of Benchmarking the timings for 4-different types of arrays and running each of them 10 runs for 10 array length 20 to 10240. Establishing the relationship between the length of an array and the mean time taken for Insertion Sort.

Length of Array(n)	Ordered Array	Partially Ordered Array	Randomly Ordered Array	Reversely Ordered Array
20	0.0030541	0.0031086	4.92E-04	5.16E-03
40	0.0061791	0.0071459	6.37E-04	9.54E-03
80	0.0038168	0.0149918	0.0012248	0.0105585
160	0.0077544	0.1035166	0.0031417	0.0242249
320	0.0149875	0.0440665	0.0084332	0.0704416
640	0.0248709	0.0522583	0.0282876	0.1337163
1280	0.0487127	0.157075	0.0951874	0.1881
2560	0.100029	0.1503917	0.3671333	0.7279291
5120	0.0480794	0.5430582	1.3877834	3.0293333
10240	0.0926332	2.0285916	5.4631291	11.2180623

Evidence to Support Conclusion:

We created a Benchmark_SortOutput class to check the Insertion sort meantime for 4 different types of arrays (i.e. Ordered Array, Partially Ordered Array, Randomly Ordered Array, Reversely Ordered Array) with 10 different array size starting with 20 and using the doubling method to run the benchmark.

1. Output for Ordered Array

The screenshot shows an IDE with the following components:

- Project Explorer:** Displays a project structure with folders like 'src', 'main', 'java', and 'edu.neu.coe.info6205'.
- Editor:** Shows the code for the `Benchmark_SortOutput` class. It includes imports for `InsertionSort` and `Random`, and defines methods for generating random arrays and ordered arrays.
- Run Console:** Displays the output of the benchmark. It shows the start of the benchmarking process for an ordered array and the resulting mean times for various array sizes.

The output in the Run Console is as follows:

```

-----INSERTION SORT FOR ORDERED ARRAY-----
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 20 takes a meantime of 0.0030540999999999997
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 40 takes a meantime of 0.0061791
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 80 takes a meantime of 0.0038168
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 160 takes a meantime of 0.0077544
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 320 takes a meantime of 0.0149875
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 640 takes a meantime of 0.0248709
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Ordered Array of size 1280 takes a meantime of 0.0487127

```

2. Output for Partially Ordered Array

The screenshot shows an IDE with a project named 'INFO6205_PSA'. The 'src/main/java' directory contains a package 'edu.neu.coe.info6205' with a sub-package 'sort'. The 'Benchmark_SortOutput.java' file is open, showing the following code:

```
import edu.neu.coe.info6205.sort.elementary.InsertionSort;

public class Benchmark_SortOutput {

    private static final int MIN_N = 20;

    private static Integer[] randomArray(int n) {
        Random r = new Random();
        Integer[] array = new Integer[n];
        for(int i=0; i<n; i++) {
            array[i] = r.nextInt(n);
        }
        return array;
    }

    private static Integer[] orderedArray(int n) {
        Integer[] array = new Integer[n];
        for(int i=0; i<n; i++) {
            array[i] = i;
        }
        return array;
    }
}
```

The 'Run' output shows the following results for Insertion Sort on a partially ordered array:

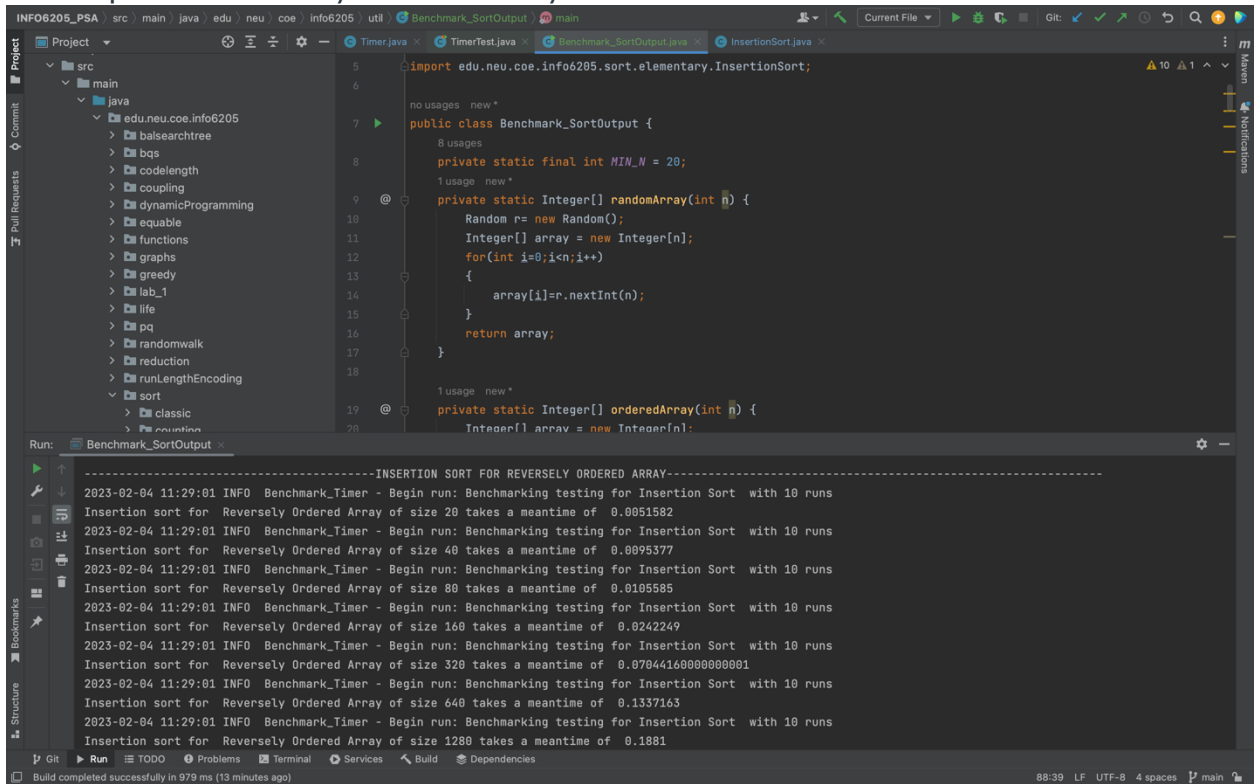
```
-----INSERTION SORT FOR PARTIALLY ORDERED ARRAY-----
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 20 takes a meantime of 0.0031086
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 40 takes a meantime of 0.007145899999999999
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 80 takes a meantime of 0.0149918
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 160 takes a meantime of 0.1035166
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 320 takes a meantime of 0.044066499999999995
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 640 takes a meantime of 0.0522583
2023-02-04 11:29:07 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Partially Ordered Array of size 1280 takes a meantime of 0.15707500000000002
```

3. Output for Randomly Ordered Array

The screenshot shows the same IDE as above, but the 'Run' output shows the results for Insertion Sort on a randomly ordered array:

```
-----INSERTION SORT FOR RANDOMLY ORDERED ARRAY-----
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 20 takes a meantime of 4.917999999999999E-4
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 40 takes a meantime of 6.372E-4
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 80 takes a meantime of 0.0012248
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 160 takes a meantime of 0.0031417
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 320 takes a meantime of 0.0084332
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 640 takes a meantime of 0.028287600000000003
2023-02-04 11:29:08 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Randomly Ordered Array of size 1280 takes a meantime of 0.0951874
```

4. Output for Reversely Ordered Array



```
import edu.neu.coe.info6205.sort.elementary.InsertionSort;

public class Benchmark_SortOutput {

    private static final int MIN_N = 20;

    private static Integer[] randomArray(int n) {
        Random r = new Random();
        Integer[] array = new Integer[n];
        for(int i=0; i<n; i++)
        {
            array[i]=r.nextInt(n);
        }
        return array;
    }

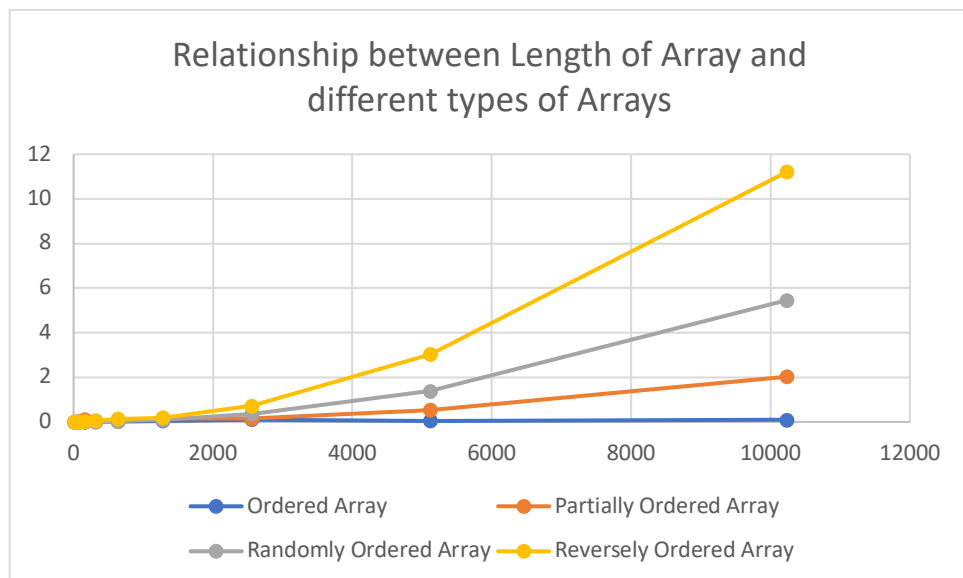
    private static Integer[] orderedArray(int n) {
        Integer[] array = new Integer[n];
        for(int i=0; i<n; i++)
        {
            array[i]=i;
        }
        return array;
    }
}
```

Run: Benchmark_SortOutput

```
-----INSERTION SORT FOR REVERSELY ORDERED ARRAY-----
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 20 takes a meantime of 0.0051582
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 40 takes a meantime of 0.0095377
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 80 takes a meantime of 0.0105585
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 160 takes a meantime of 0.0242249
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 320 takes a meantime of 0.07044160000000001
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 640 takes a meantime of 0.1337163
2023-02-04 11:29:01 INFO Benchmark_Timer - Begin run: Benchmarking testing for Insertion Sort with 10 runs
Insertion sort for Reversely Ordered Array of size 1280 takes a meantime of 0.1881
```

Graphical Representation:

As per the data plotted in the excel sheet for different types of arrays. We get the following graph.



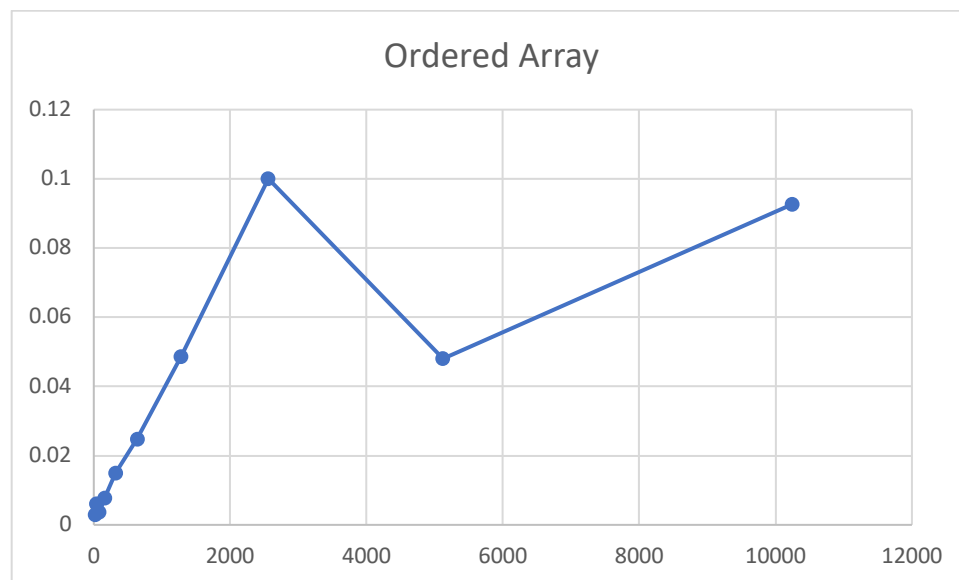
With the help of the graph, we can conclude that Reversely Ordered Array takes the most time as it is reverse and has to swap each and every element in Insertion Sort and Ordered Array takes the least time as the array is ordered.
So, we can conclude that

Ordered Array < Partially Ordered Array < Randomly Ordered Array < Reversely Ordered Array

1. Ordered Array

Length of Array(n) vs Meantime

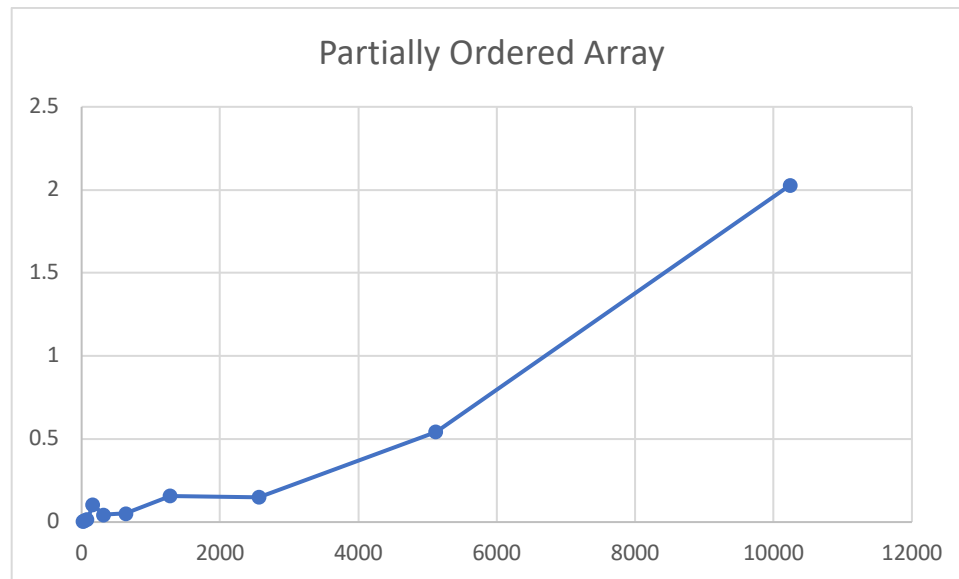
Length of Array(n)	Ordered Array
20	0.0030541
40	0.0061791
80	0.0038168
160	0.0077544
320	0.0149875
640	0.0248709
1280	0.0487127
2560	0.100029
5120	0.0480794
10240	0.0926332



2. Partially Ordered Array

Length of Array(n) vs Meantime

Length of Array(n)	Partially Ordered Array
20	0.0031086
40	0.0071459
80	0.0149918
160	0.1035166
320	0.0440665
640	0.0522583
1280	0.157075
2560	0.1503917
5120	0.5430582
10240	2.0285916

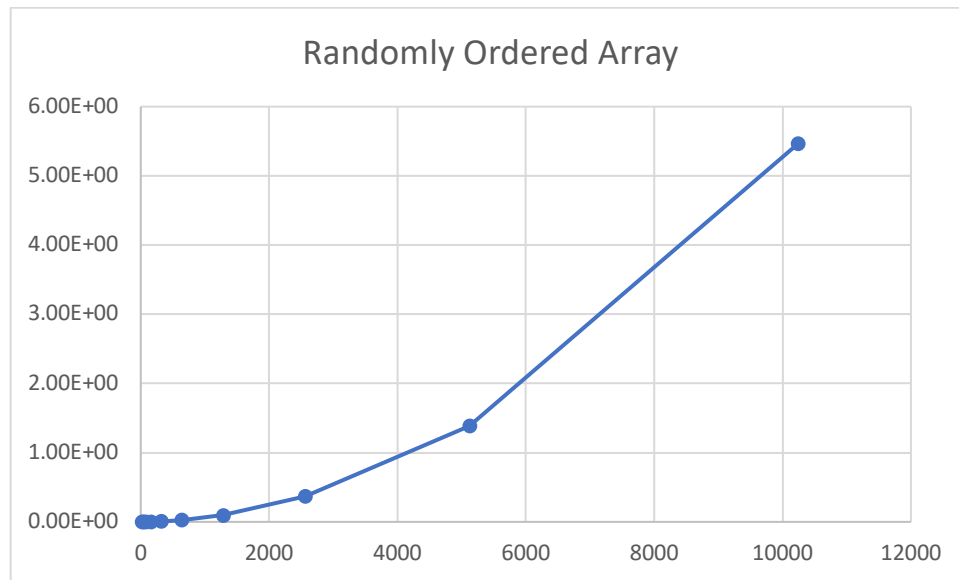


3. Randomly Ordered Array

Length of Array(n) vs Meantime

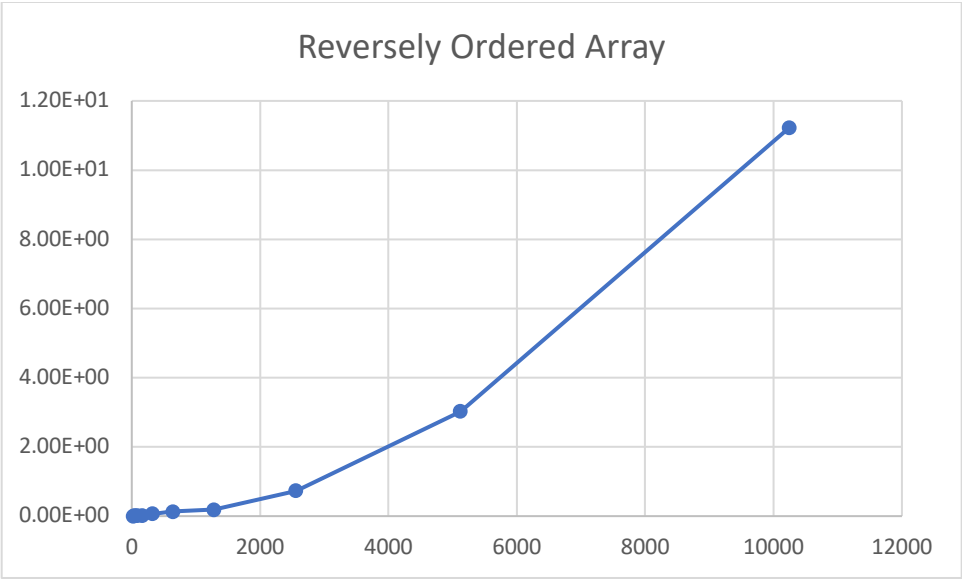
Length of Array(n)	Randomly Ordered Array
20	4.92E-04
40	6.37E-04
80	0.0012248
160	0.0031417
320	0.0084332
640	0.0282876
1280	0.0951874

2560	0.3671333
5120	1.3877834
10240	5.4631291



4. Reversely Ordered Array Length of Array(n) vs Meantime

Length of Array(n)	Reversely Ordered Array
20	5.16E-03
40	9.54E-03
80	0.0105585
160	0.0242249
320	0.0704416
640	0.1337163
1280	0.1881
2560	0.7279291
5120	3.0293333
10240	11.2180623



Unit Tests Result:

The screenshot shows an IDE with the `TimerTest` class selected in the project view. The code editor displays the `testStop()` and `testPauseAndLap()` methods. The Run window at the bottom shows the test results for `TimerTest`, indicating that all 11 tests passed successfully.

Test Results:

Test Name	Duration (ms)
testPauseAndLapResume0	164
testPauseAndLapResume1	315
testLap	210
testPause	207
testStop	106
testMillisecs	102
testRepeat1	124
testRepeat2	252
testRepeat3	601
testRepeat4	368
testPauseAndLap	105

Overall Test Summary: Tests passed: 11 of 11 tests – 2 sec 554 ms

