PSA Assignment 3

Name: Yash Pravin Pawar

Github: github.com/NeuYash/PSAAssignments

Email: [pawar.ya@northeastern.edu](mailto:pawar.ya@northeastern.edu)

Classes Changed:  
 1]InsertionSort.java  
 2]Benchmark\_Timer.java

3]Timer.java

Images attached:

1]Analysis Graph

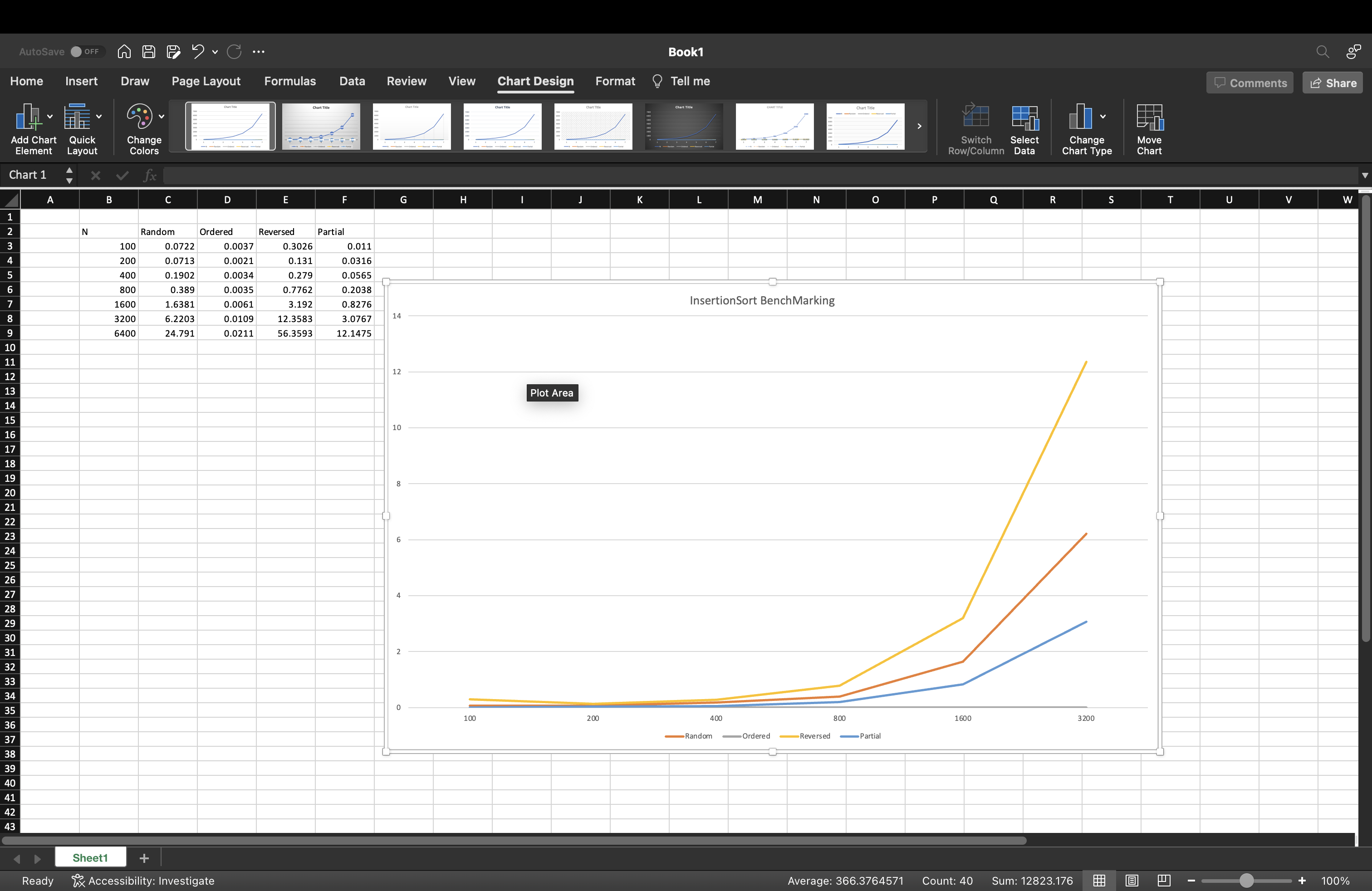
2]BenchmarkTest

3]TimerTest

4]InsertionSortTest

5]BenchMarks for InsertionSort

We had to implement BenchmarkTimer and Timer classes to map insertionSort timings on various different types of arrays. We had to test InsertionSort benchmarks on Randomized, Ordered, Reverse Ordered and Partially Ordered Integer arrays. We Oversaw a trend in which, Insertion Sort takes more time for ReversedOrdered Integer arrays followed by Randomized Integer arrays followed by Partial Integer arrays. InsertionSort takes least time for Ordered Integer arrays.  
As you can see in the following graph, Insertion Sort takes most time for Reversed Ordered Integer arrays and then Random Arrays.



InsertionSort Code Fix :

public void sort(X[] xs, int from, int to) {

final Helper<X> helper1 = getHelper();

for (int i = from + 1; i < to; i++) {

int k = i;

while (k > from && helper1.swapStableConditional(xs, k)) {

k--;

}

}

}

Similarly we changed the Repeat Function in Timer Class as well as 2 other functions:

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");

// **FIXME**: note that the timer is running when this method is called and should still be running when it returns. by replacing the following code

T t = supplier.get();

pause();

for (int i = 0; i < n; i++) {

if (preFunction != null) {

t = preFunction.apply(t);

}

resume();

U u = function.apply(t);

pauseAndLap();

if (postFunction != null) {

postFunction.accept(u);

}

}

double meantime = meanLapTime();

resume();

return meantime;

// END

}

------

private static long getClock() {

// **FIXME** by replacing the following code

return System.*nanoTime*();

// END

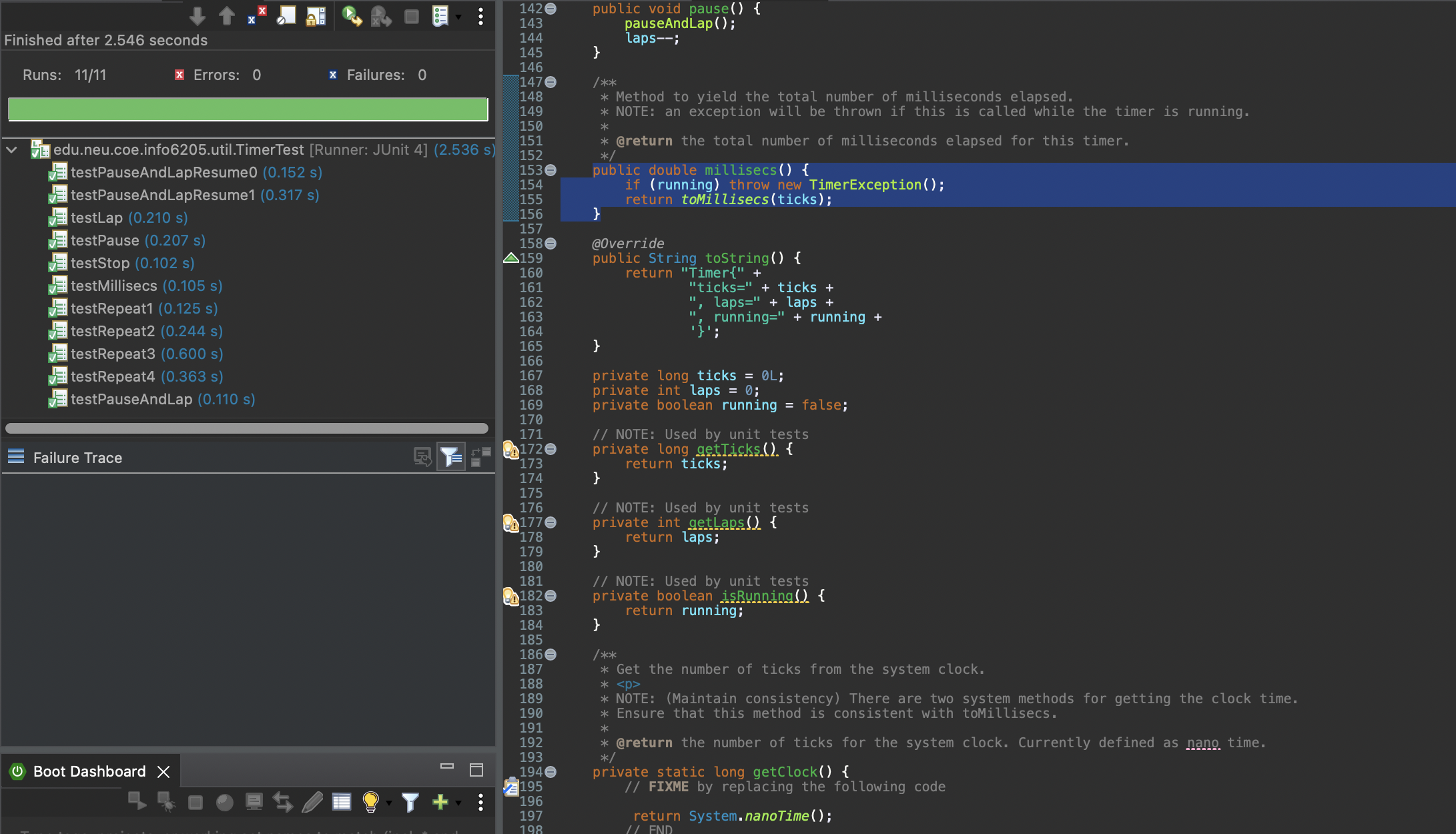
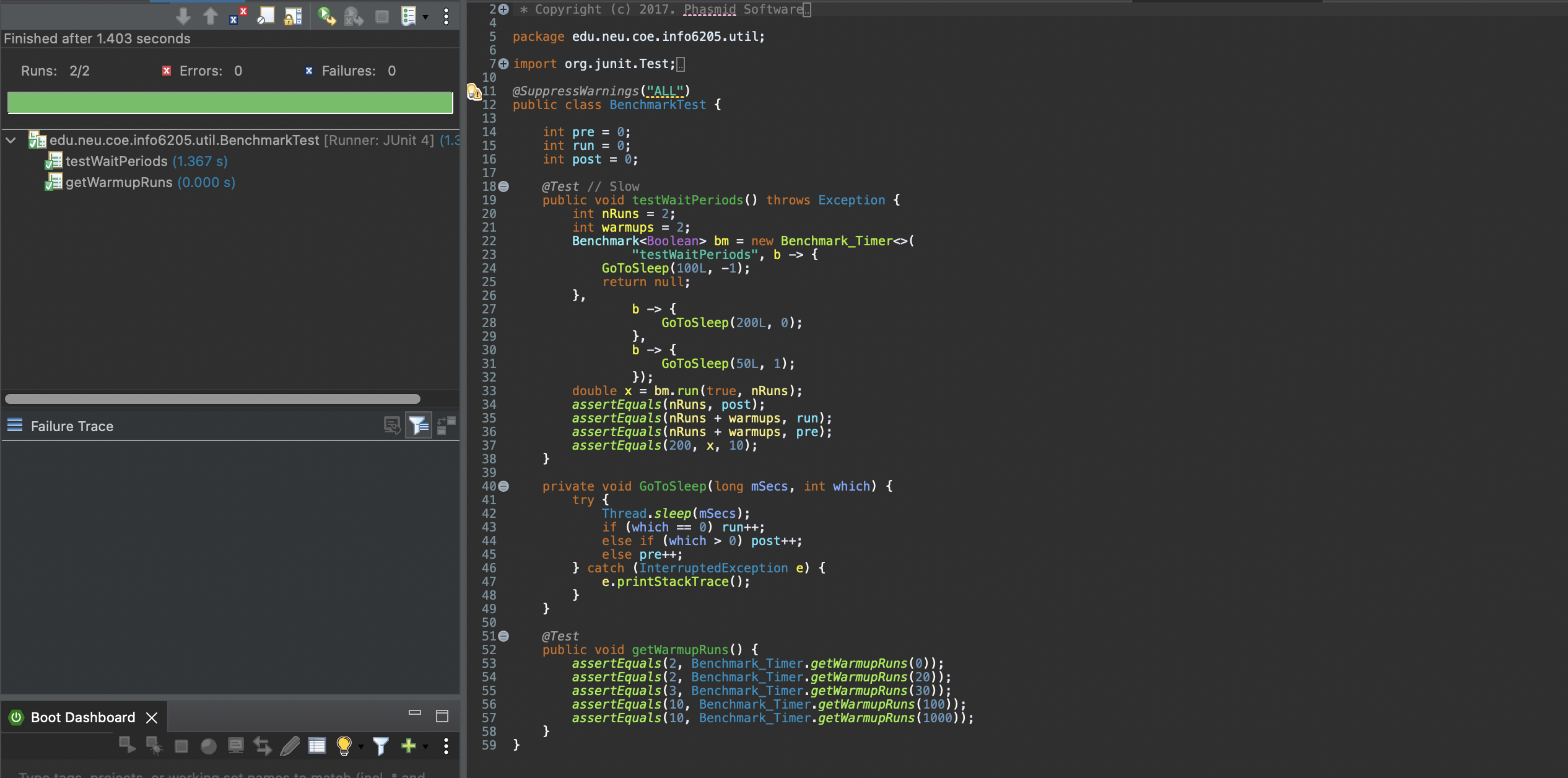
}

public double millisecs() {

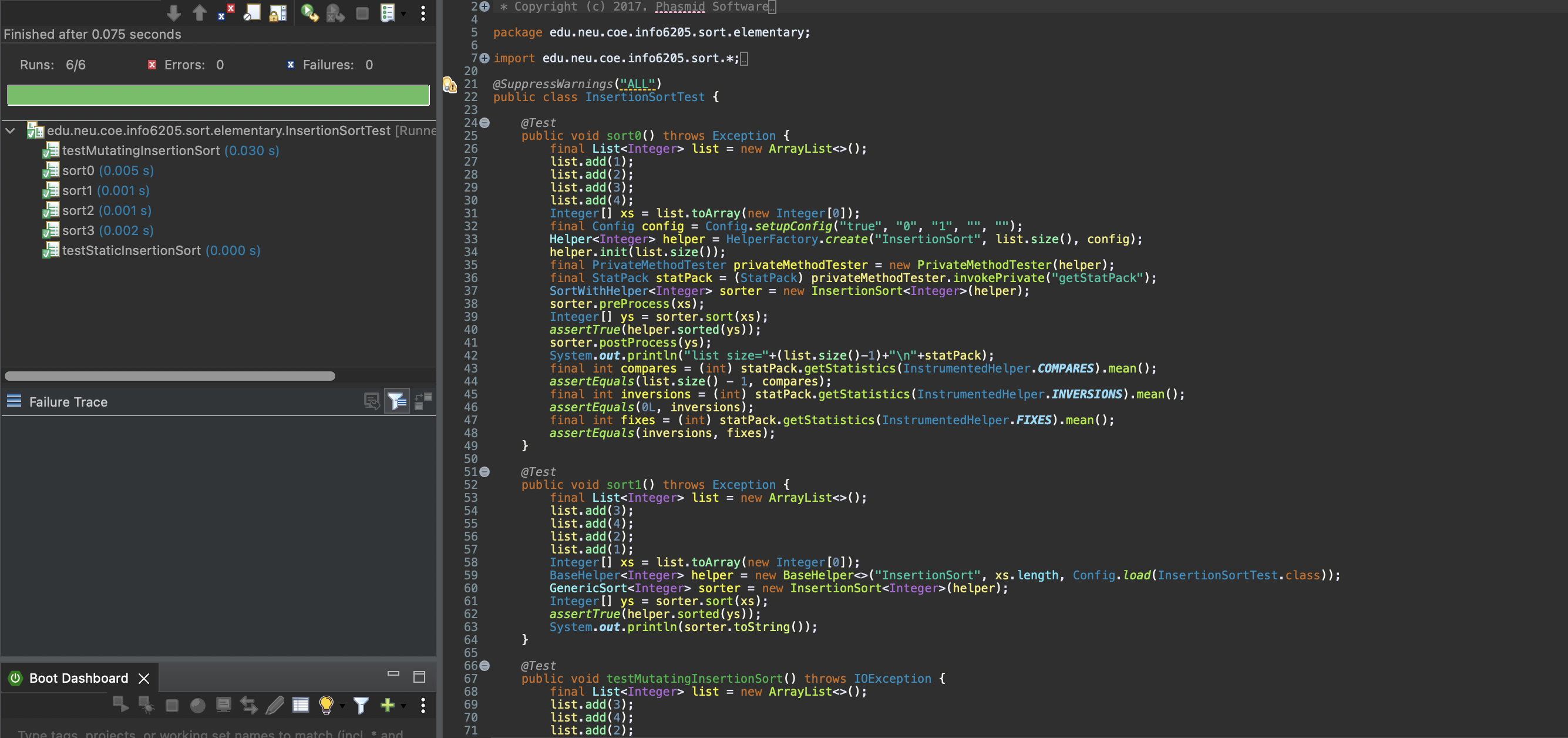
if (running) throw new TimerException();

return *toMillisecs*(ticks);

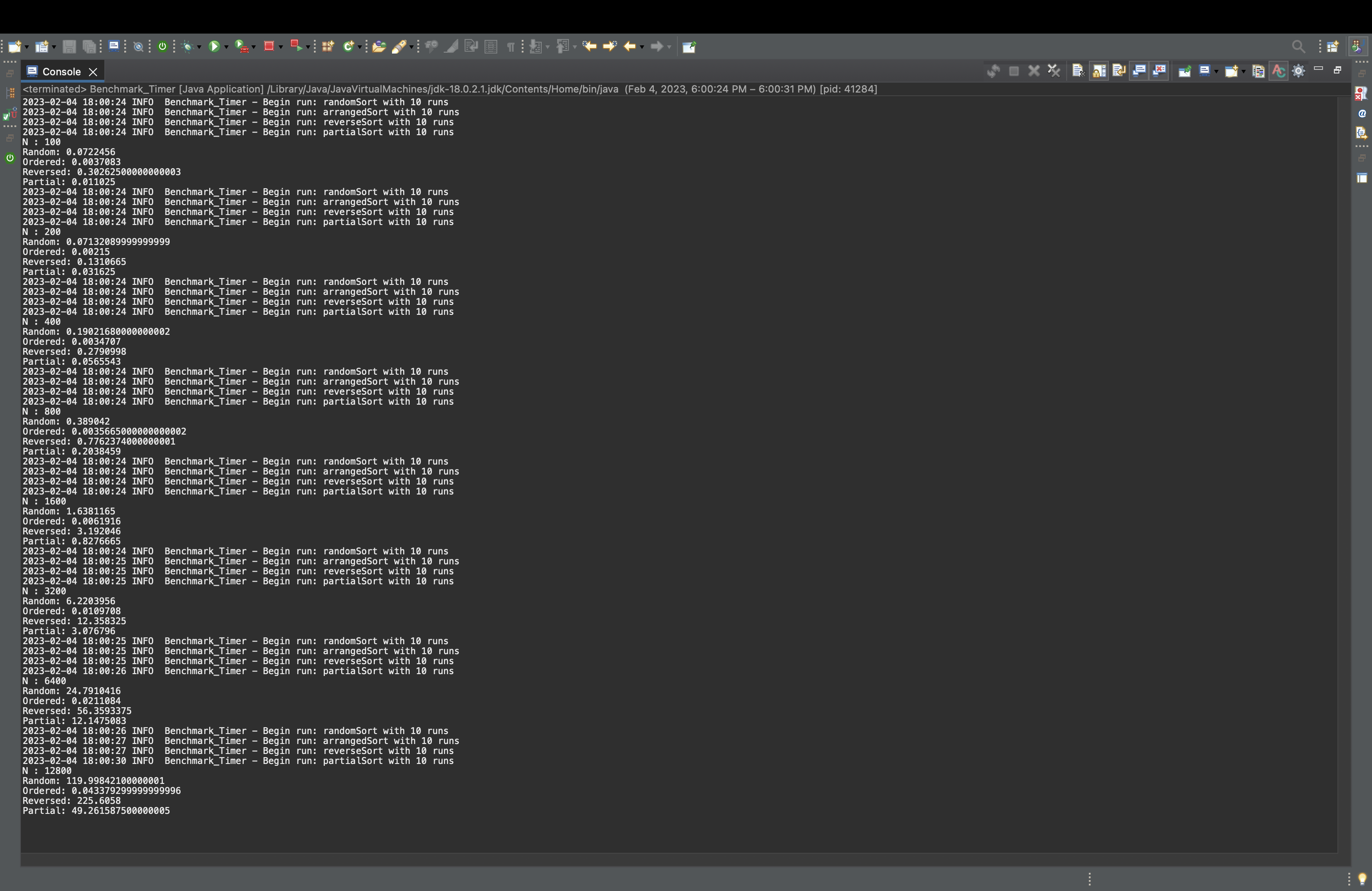
}

--------------------------------------------------------------------------------------------------------------------------------  
We will now showcase all the testCase outputs   
  
TimerTest   
  
BenchMarkTest  


InsertionSortTest



InsertionSortBenchmark:  
We created a Main Function In BenchmarkTimer Class that will create 4 different types of arrays and time it with Sort method in Insertion Sort java class.   
Following are the timing results for the same.

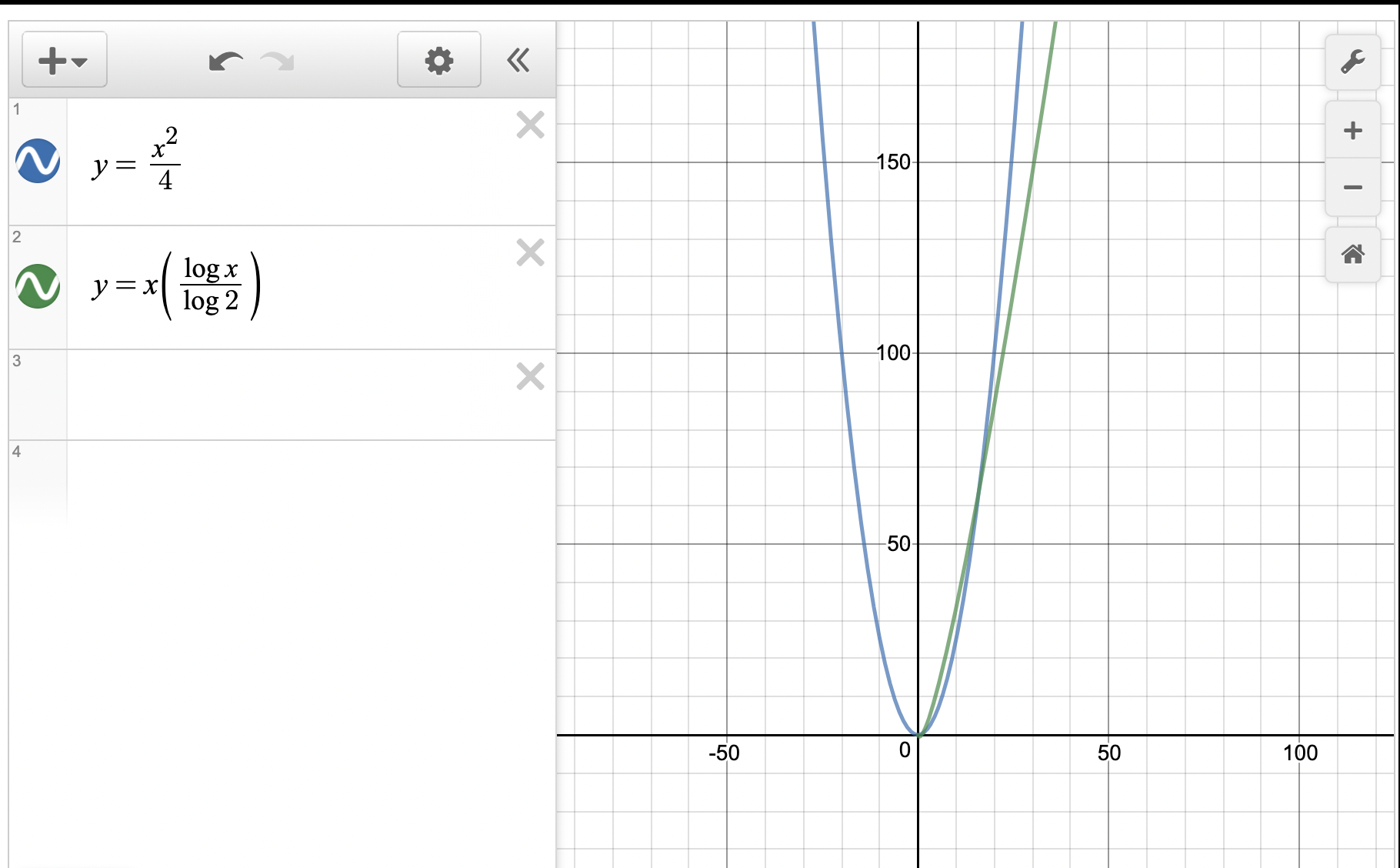


Additional Information :-

1] insertion sort requires **1/4(N2 - N)** comparisons, and thus should require about 1/2 the comparisons needed by selection sort.

2]Insertion Sort runs in O(n) time in best case (ordered list) and O(n^2) time in worst case( reverse ordered list ).

3]InsertionSort is a stable sort and has space complexity of O(1).

4] Number of Inversions in worst case (reverse ordered list) in Insertion sort is N\*(N-1)/2.   
5] When the array is already Sorted, there are no Inversions.  
6] Insertion sort though being a O(n^2) time complexity Algorithm for sorting. It can be better than a MergeSort (O(nlogn)) sorting algorithm when size of array is less than 16.  
This Analysis can be found out by the following graph  
  
Blue curve : Insertion Sort   
Green Curve   
7] Space Complexity of InsertionSort is better than MergeSort.