**Program Structures and Algorithms  
Spring 2024**

NAME: Amar Nagargoje  
NUID: 002273113  
GITHUB LINK: <https://github.com/amarneu/INFO6205/commit/20e02f9b3b21b988150822bbe05e2739826380bb>

Excel: [Book.xlsx](https://northeastern-my.sharepoint.com/:x:/g/personal/nagargoje_a_northeastern_edu/EXfYIxvR-sJAjtr8cZO30W0BHIbvwb8aush32IPDB5WEzg?e=IjTZWD)

**Task:** To understand benchmarking of algorithm: *InsertionSort*

**Relationship Conclusion:**

On running the main method in the Benchmark\_Main class multiple times,   
by using the doubling method to test for different values of input arrays   
(from array length = 100 to array length =51200) for arrays of 4 different   
types: ordered arrays, partially ordered arrays, randomly ordered arrays   
and reversely ordered arrays, running them for 10 runs each, I established   
the following conclusion between length of the input(n) and the mean-time   
taken for the Insertion Sort after plotting the values give below.

For sorted array InsertionSort: 739.46 millisecs for n=200

For reverse array InsertionSort: 924.73 millisecs for n=200

For partially sorted array InsertionSort: 361.77 millisecs for n=200

For random array InsertionSort: 363.83 millisecs for n=200

For sorted array InsertionSort: 296.22 millisecs for n=400

For reverse array InsertionSort: 814.86 millisecs for n=400

For partially sorted array InsertionSort: 345.2 millisecs for n=400

For random array InsertionSort: 431.66 millisecs for n=400

For sorted array InsertionSort: 313.78 millisecs for n=800

For reverse array InsertionSort: 2099.44 millisecs for n=800

For partially sorted array InsertionSort: 294.84 millisecs for n=800

For random array InsertionSort: 915.4 millisecs for n=800

For sorted array InsertionSort: 322.5 millisecs for n=1600

For reverse array InsertionSort: 7526.94 millisecs for n=1600

For partially sorted array InsertionSort: 309.35 millisecs for n=1600

For random array InsertionSort: 2288.01 millisecs for n=1600

For sorted array InsertionSort: 323.12 millisecs for n=3200

For reverse array InsertionSort: 30498.76 millisecs for n=3200

For partially sorted array InsertionSort: 335.26 millisecs for n=3200

For random array InsertionSort: 8138.5 millisecs for n=3200

For sorted array InsertionSort: 353.36 millisecs for n=6400

For reverse array InsertionSort: 118158.09 millisecs for n=6400

For partially sorted array InsertionSort: 24894.24 millisecs for n=6400

For random array InsertionSort: 31693.09 millisecs for n=6400

A graph of different colored lines

Description automatically generated

* Insertion sort operates by examining adjacent elements within the array and rearranging them if they are out of order.

• As illustrated in the graph above, Insertion sort exhibits its longest execution time when dealing with arrays sorted in reverse order. This is due to the necessity of swapping elements for every adjacent pair, resulting in the worst-case scenario for this sorting algorithm.

• Conversely, it demonstrates the shortest execution time when presented with an already sorted array.

• In terms of performance, the following ranking is observed for the insertion sort algorithm across the provided array types:

Sorted Array < Partially Ordered Array < Randomly Ordered Array < Reverse Ordered Array.

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**Evidence to support that conclusion: (Partwise)**

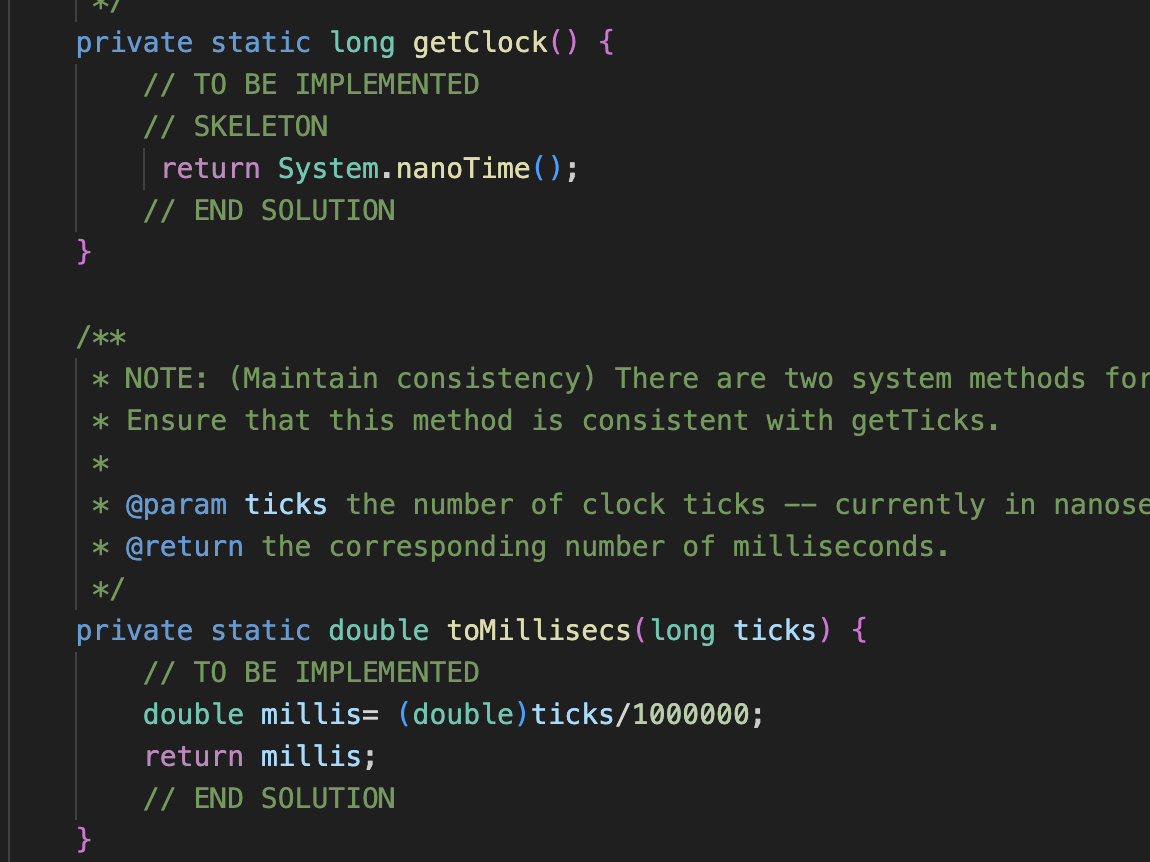
Part 1: Implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*)

Repeat: To calculate average milliseconds per repetition

A screen shot of a computer

Description automatically generated

Get clock: to get nano

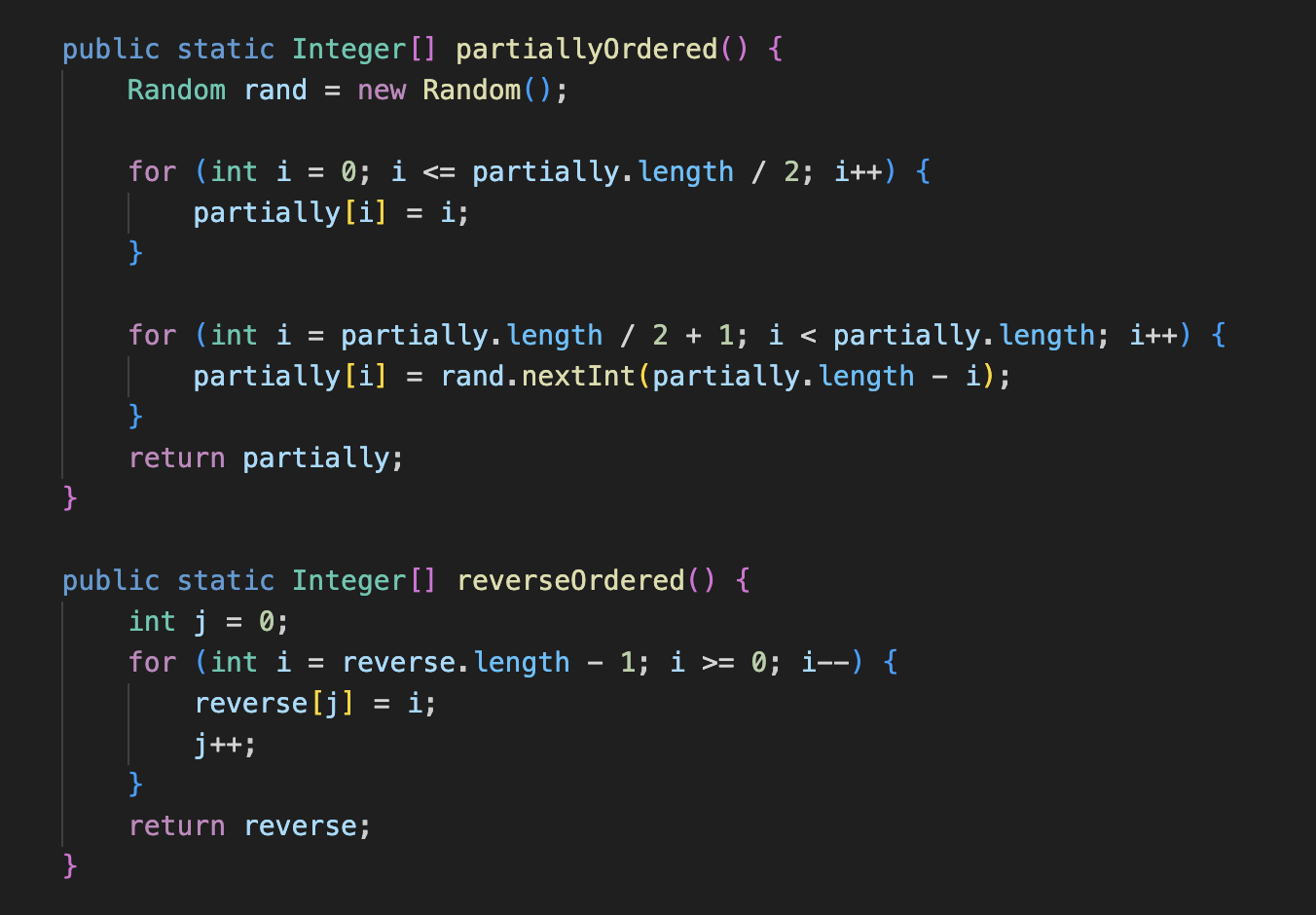
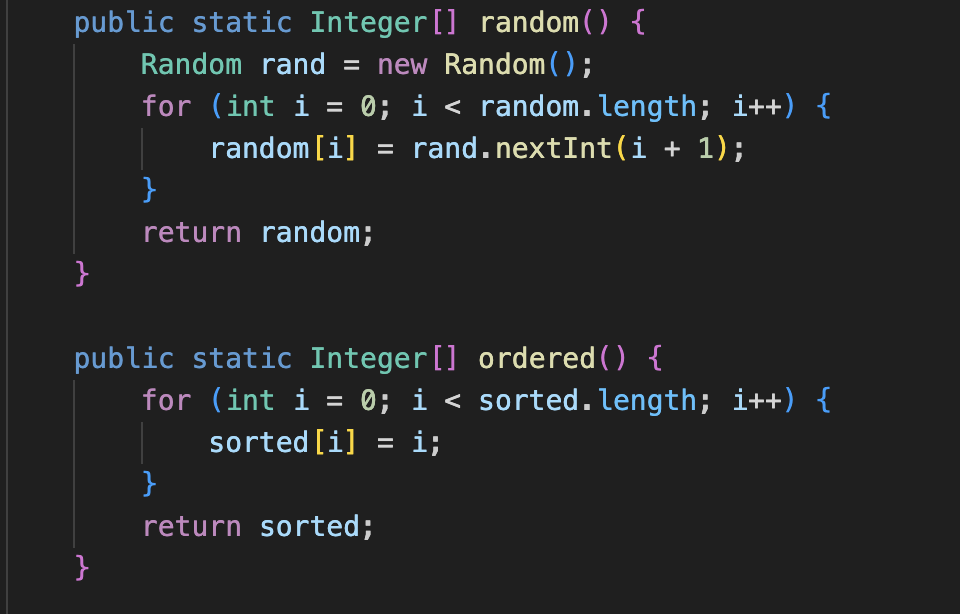
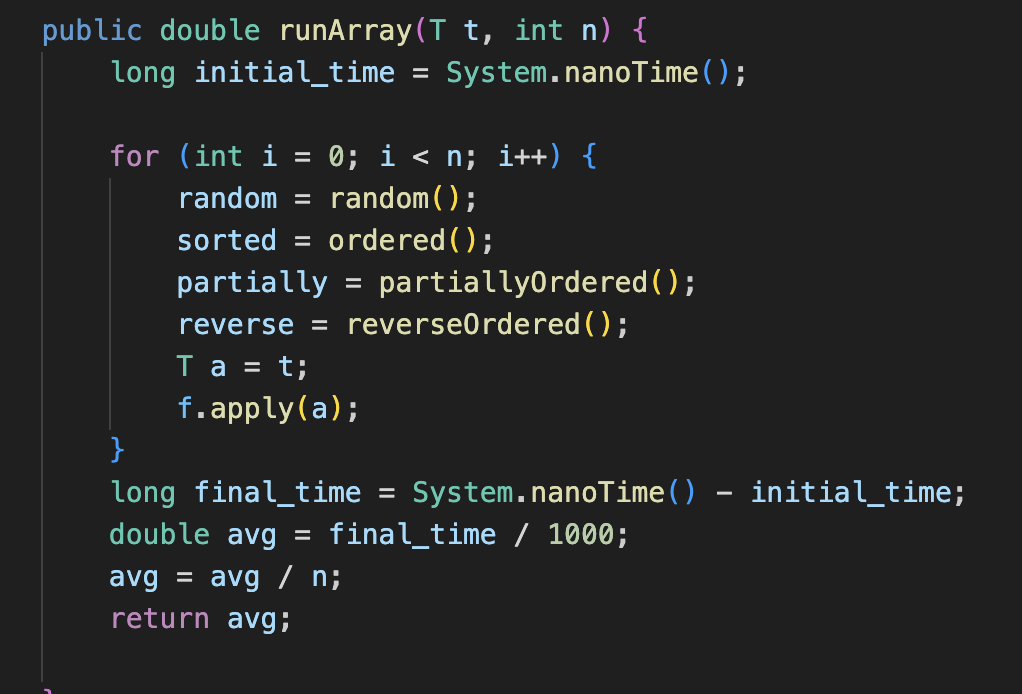
Get Milisecs: to get mil

Part 2: Implement *InsertionSort*

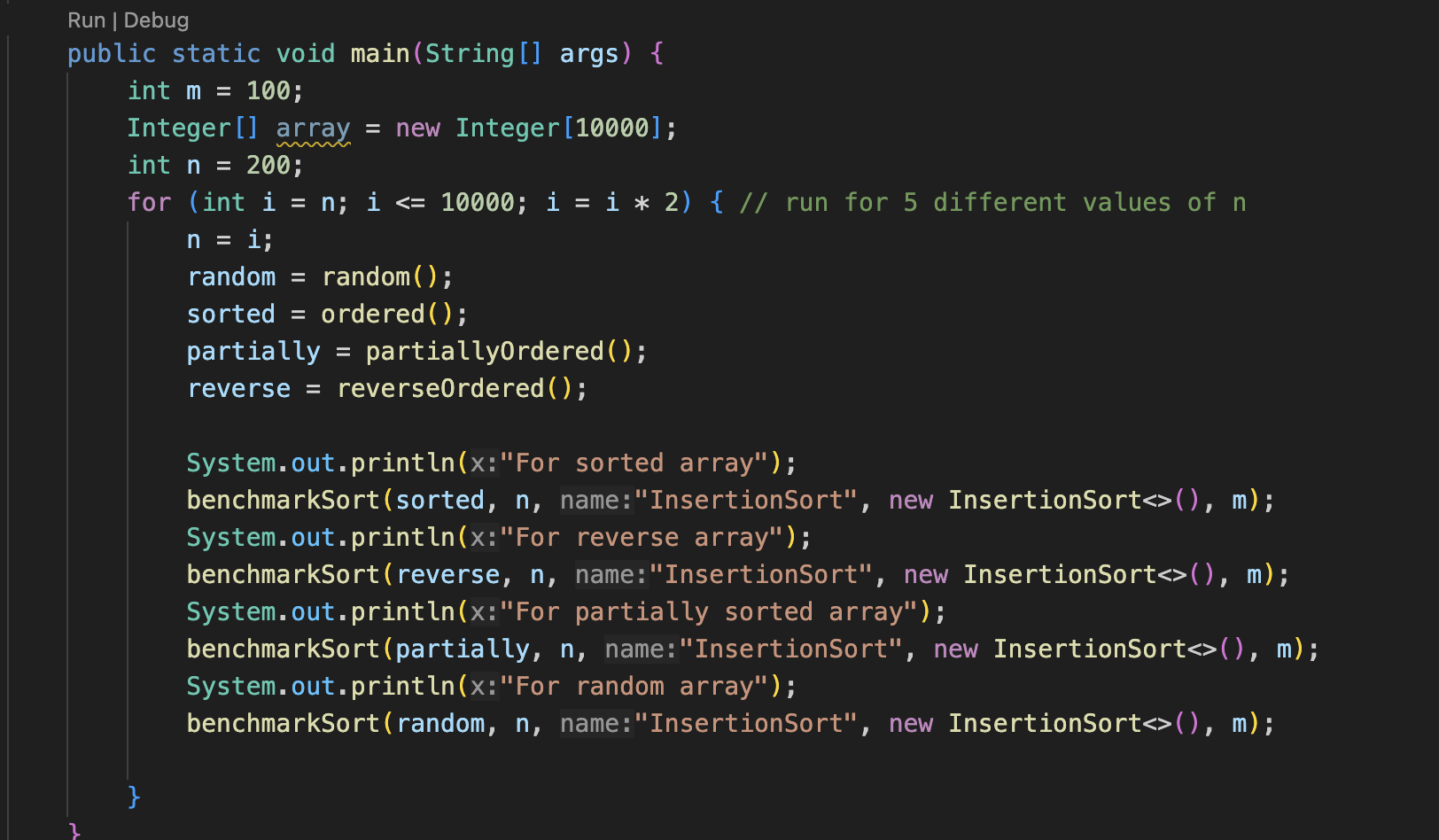
A screen shot of a computer code

Description automatically generated

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.



Run for 5 different values:



**Unit Test Screenshots:**

Part 1: BenchmarkTest and TimeTest

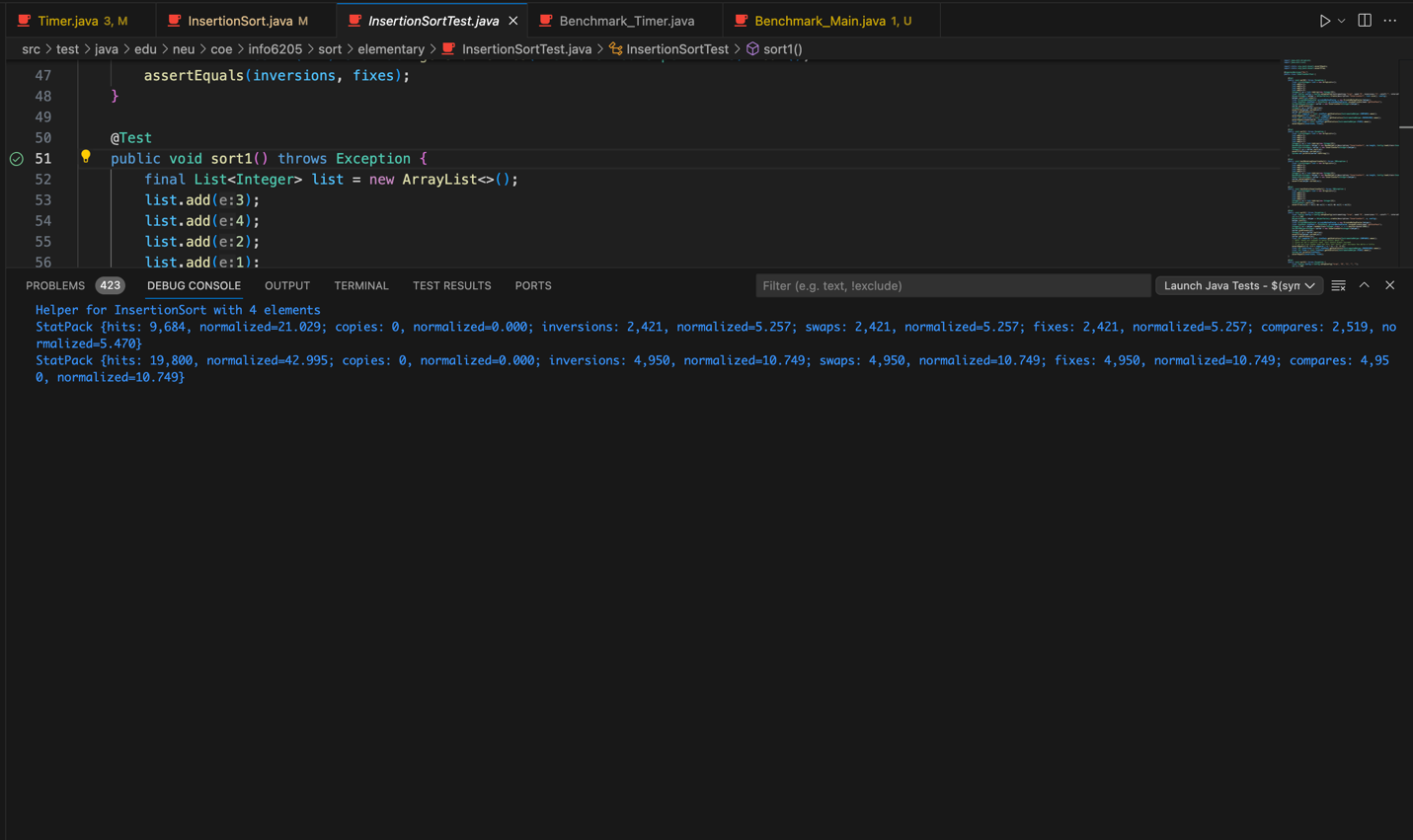
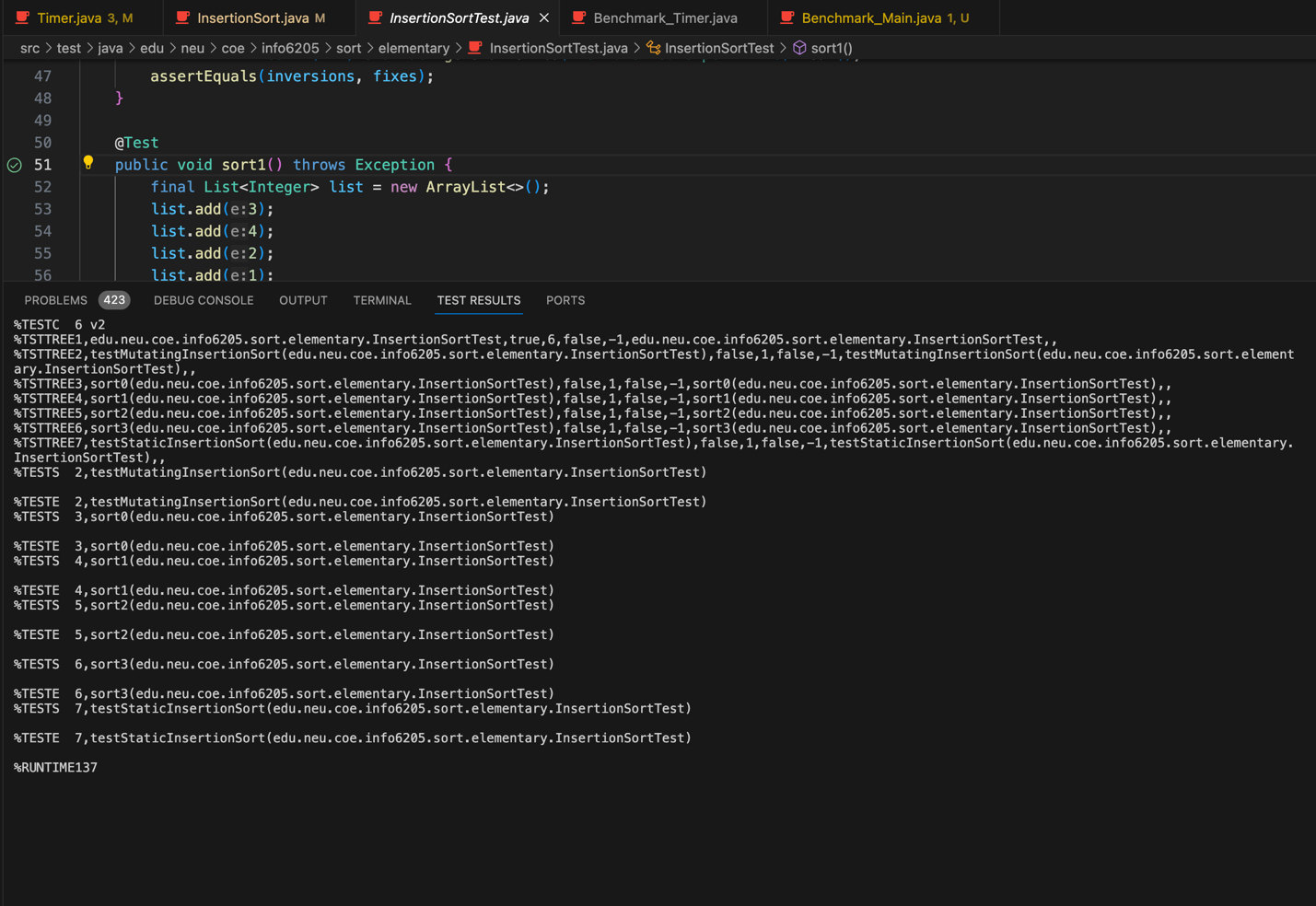
A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

Part 2: Insertion Sort Test Cases:



Part 3:

A screen shot of a computer

Description automatically generated

