Srividya Burra(002985163)

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 2**

* **Task:**

(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.

(Part 2) Implement InsertionSort(in the InsertionSort class) by simply looking up the insertion code used byArrays.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, coantinuing 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.

* **Tasks Performed:**

Implemented the following methods from Timer class:

repeat(), toMillisecs(), getClock().

* Implemented code for the “sort()” in InsertionSort.java
* Implemented main() in Benchmark\_Timer.java with array sizes ranging between 16 and 8192.
* Took an average of 50 iterations.
* Tested insertion sort for random sorted array, sorted array, reverse sorted array and partially sorted array.
* Collected the mean time taken (in milliseconds) and tabulated results for the same.
* Plotted graph for all the above results.
* **Relationship Conclusion:**

**🡪** From the experimental values obtained we can observe that “reverse sorted array” takes the maximum time while the “sorted array” takes the minimum amount of time.

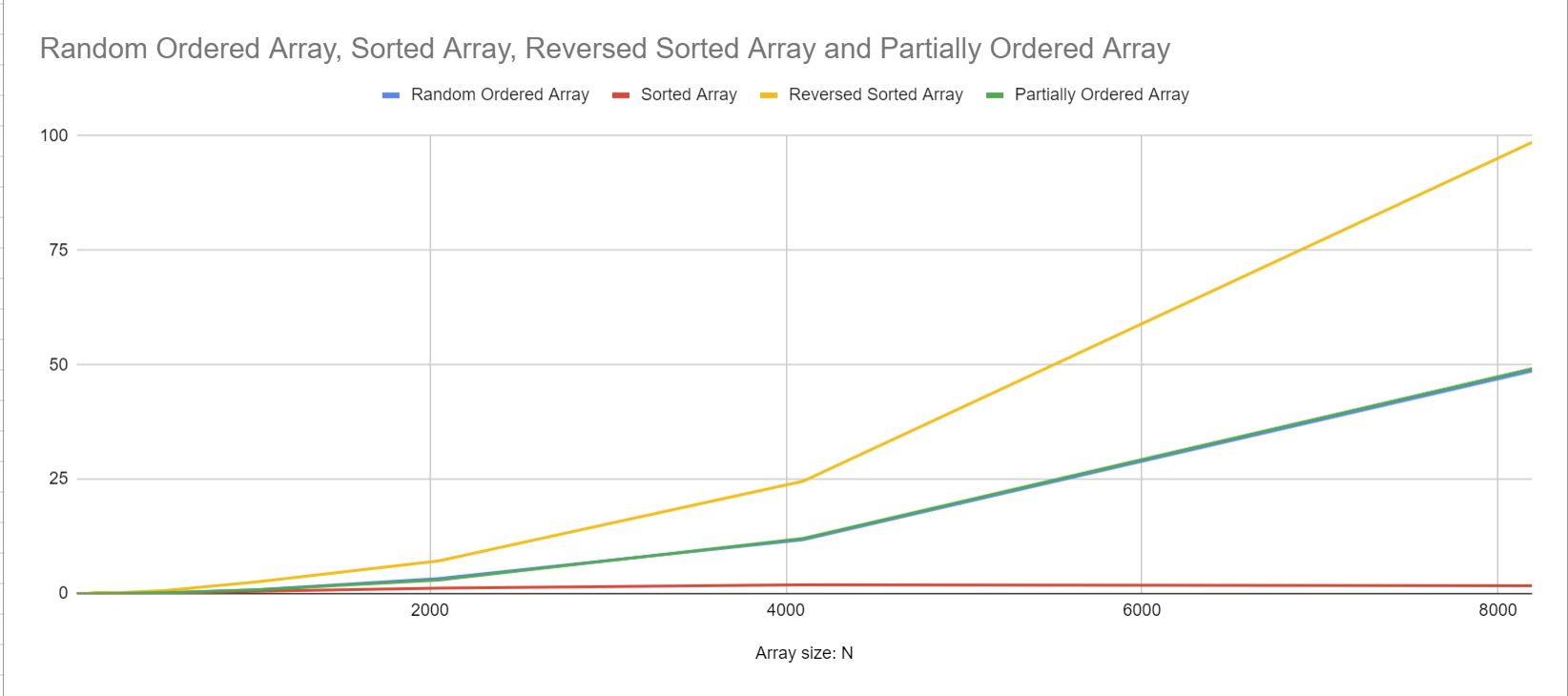
**🡪**We can observe that the “partially ordered array” and “random ordered array” has similar performance.

**🡪**By this, we can conclude:

**Reverse Order > Random Order > Partially sorted > Sorted**

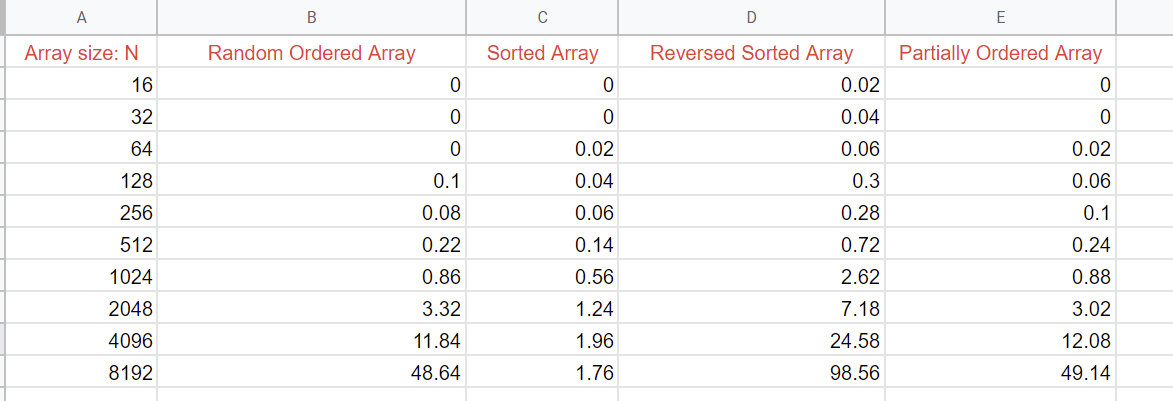
**O(N^2) O(N^2) O(N^2) O(N)**

* **Evidence to support the conclusion:**

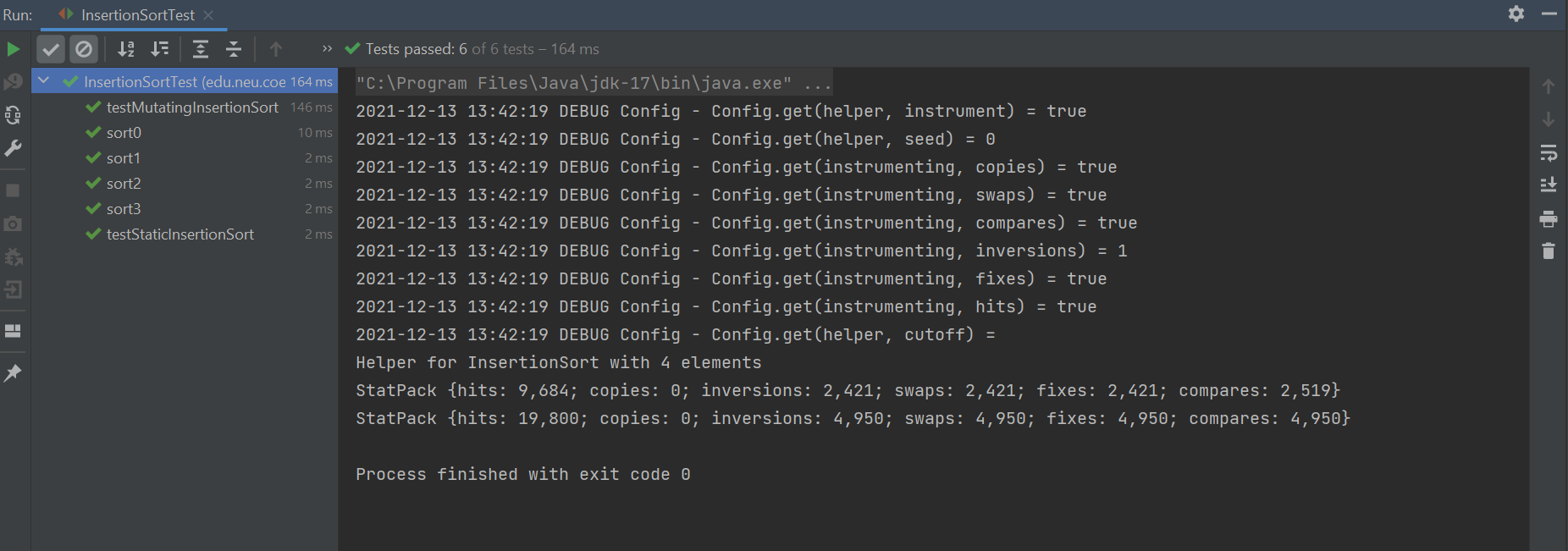
****

X-axis : Array sizes (16-8192)

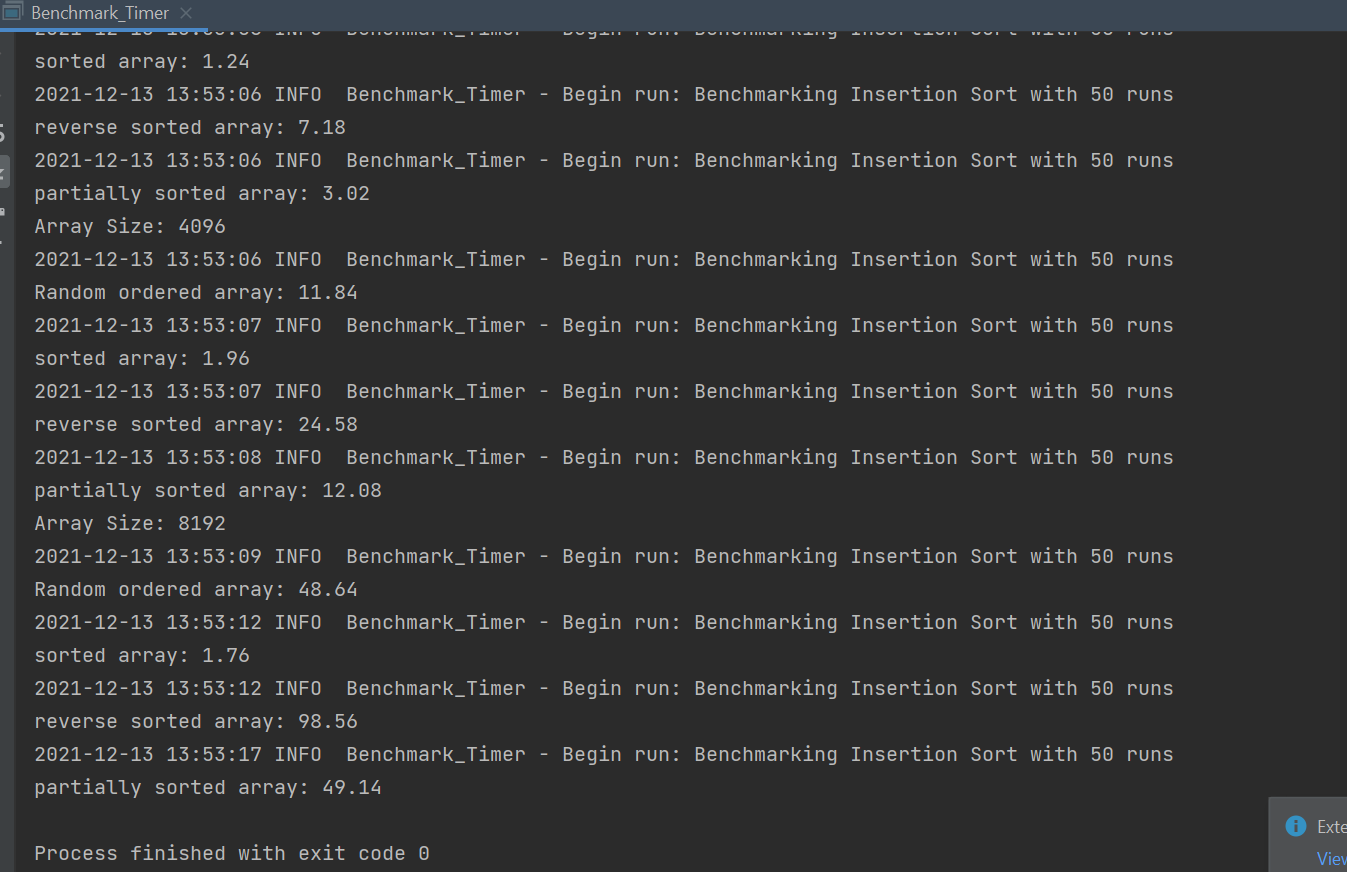
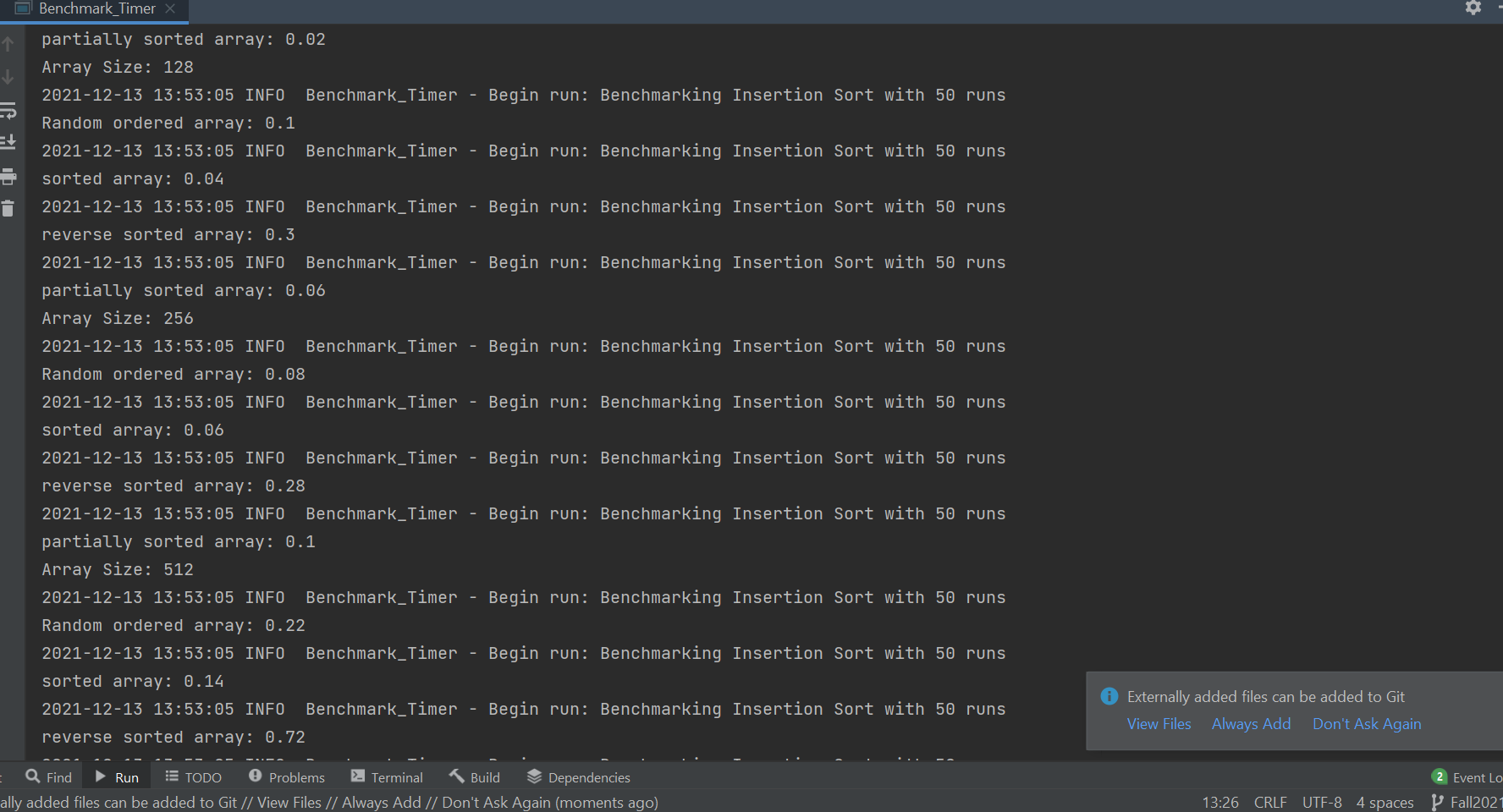
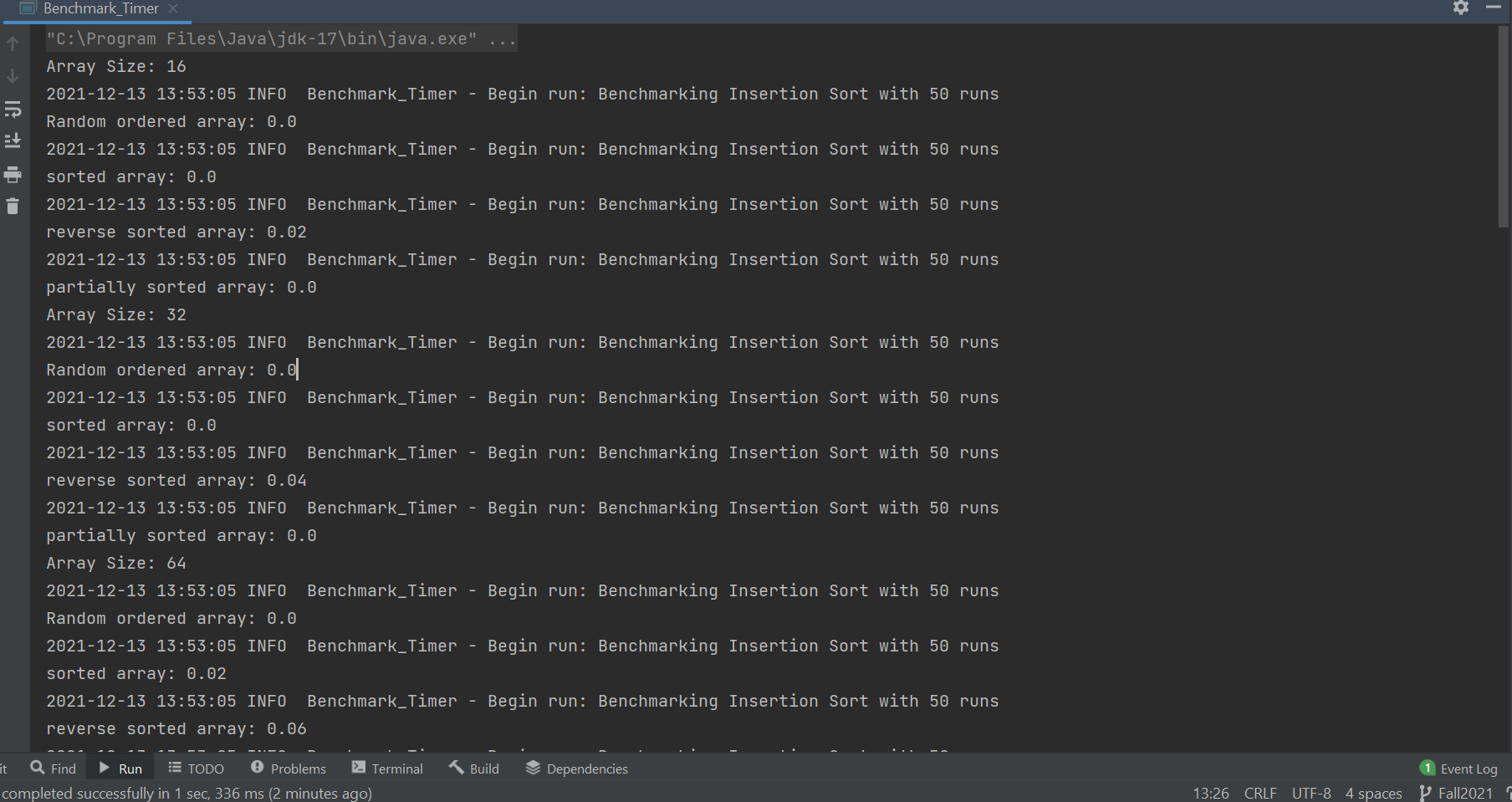
Y-axis : time taken in milliseconds.



**Test Cases:**

****

* **Output:**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |