Program Structures and Algorithms
Spring 2024

NAME: Suchita Arvind Dabir
NUID: 002957879
GITHUB LINK: https://github.com/suchitadabir/INFO6205

## Task

(Part 1) To implement three methods in the Timer class repeat, getClock, and toMillisecs.
(Part 2) Implement the InsertionSort class and ensure correctness through unit tests.
(Part 3) Execute the benchmarks to measure the running time of sort across four different initial array orderings: random, ordered, partially ordered, and reverse ordered.

## Conclusion

On average, the time taken by different ordering methods are,

| Scenario | Time Complexity | Description |
|---|---|---|
| **Random Order** | O($n^2$) | Extensive comparisons and swaps required, resulting in quadratic time complexity. |
| **Ordered (Best Case)** | O(n) | Elements are already sorted, leading to linear time complexity. |
| **Partially Ordered (Varies with degree of ordering)** | Approaches O(n) | Nearly sorted arrays approach O(n). |
| | Approaches O($n^2$) | Less ordered arrays approach O($n^2$) |
| **Reverse-Ordered (Worst Case)** | O($n^2$) | Extensive comparisons and swaps required, resulting in quadratic time complexity. |

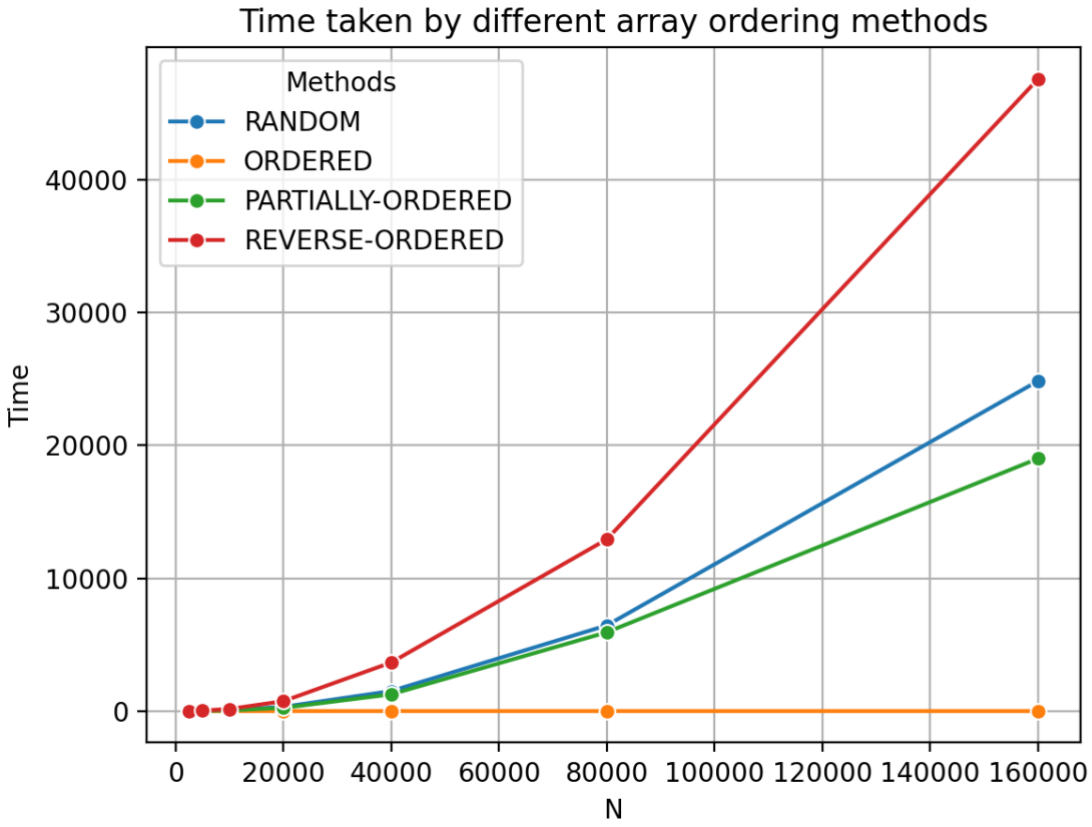
## Evidence to support that conclusion:

- I utilized the Benchmark_Timer and Timer classes from the repository to measure the runtime of the InsertionSort algorithm across four different initial array ordering scenarios.
- In the Assignment3Benchmark class, I conducted benchmarks by choosing seven values of N using the doubling method and a safetyFactor set to 10 to ensure enough elements across various array ordering scenarios.
- The final values of N are 2500, 5000, 10000, 20000, 40000, 80000, and 160000.
- Subsequently I exported all data to a CSV file viz. Assignment3Benchmark.csv.
- The table below displays the runtime in milliseconds for each ordering scenarios at the selected value of N for various Run counts.

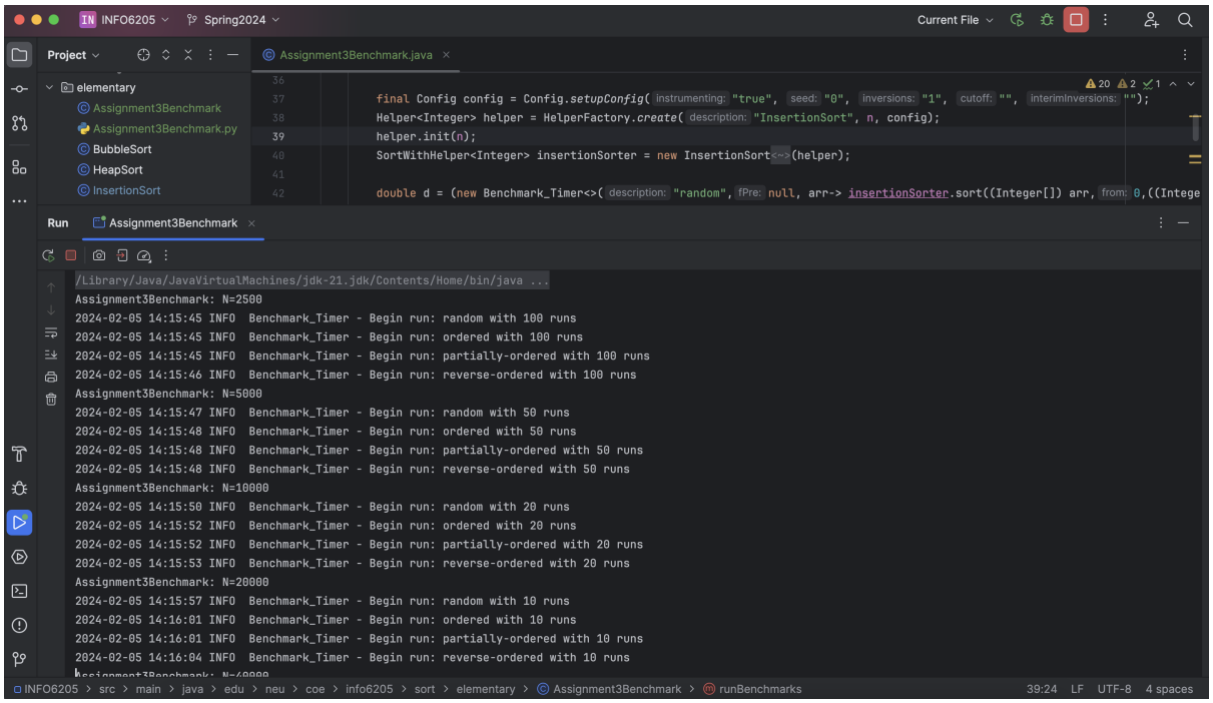
Data for run times of Insertion sort.

| Array-Ordering | Runs | N | Time |
|---|---|---|---|
| RANDOM | 100 | 2500 | 4.86834682 |
| ORDERED | 100 | 2500 | 0.00641077 |
| PARTIALLY-ORDERED | 100 | 2500 | 3.6011954900000000 |
| REVERSE-ORDERED | 100 | 2500 | 9.47078336 |
| RANDOM | 50 | 5000 | 18.77655426 |
| ORDERED | 50 | 5000 | 0.01190922 |
| PARTIALLY-ORDERED | 50 | 5000 | 14.122764240000000 |
| REVERSE-ORDERED | 50 | 5000 | 37.56323086 |
| RANDOM | 20 | 10000 | 75.6358021 |
| ORDERED | 20 | 10000 | 0.02340835 |
| PARTIALLY-ORDERED | 20 | 10000 | 57.6078627 |
| REVERSE-ORDERED | 20 | 10000 | 158.60355395 |
| RANDOM | 10 | 20000 | 318.3298709 |
| ORDERED | 10 | 20000 | 0.048345700000000000 |
| PARTIALLY-ORDERED | 10 | 20000 | 241.8123959 |
| REVERSE-ORDERED | 10 | 20000 | 744.8288082 |
| RANDOM | 5 | 40000 | 1505.9122666000000 |
| ORDERED | 5 | 40000 | 0.105158 |
| PARTIALLY-ORDERED | 5 | 40000 | 1259.3582832 |
| REVERSE-ORDERED | 5 | 40000 | 3656.9848578000000 |
| RANDOM | 3 | 80000 | 6451.881485333330 |
| ORDERED | 3 | 80000 | 0.268625 |
| PARTIALLY-ORDERED | 3 | 80000 | 5933.221625000000 |
| REVERSE-ORDERED | 3 | 80000 | 12930.14075 |
| RANDOM | 2 | 160000 | 24849.614271 |
| ORDERED | 2 | 160000 | 0.697625 |
| PARTIALLY-ORDERED | 2 | 160000 | 19009.708812 |
| REVERSE-ORDERED | 2 | 160000 | 47581.80202 |

- I wrote a simple python script to plot the run time in millisecond along Y-axis and N across X-axis.
- Below plotted graph shows the performance of the Insertion Sort algorithm on arrays with different initial conditions:
  - Random: Time increases at a likely quadratic rate as we increase the size of an array.
  - Ordered: It involves O(n) comparisons and O(1) swaps leading to O(n) overall time complexity resulting into the best-case scenario for Insertion sort.
  - Partially-Ordered: Time increases at a rate between the best and worst cases.
  - Reverse-Ordered: Time increases sharply, rate as we increase the size of an array. This shows a quadratic time complexity, which is the worst case for Insertion Sort.
- Hence, experiments suggest that Insertion Sort is fastest on already sorted data and slowest on reverse-sorted data. Its performance degrades significantly as the array size grows.

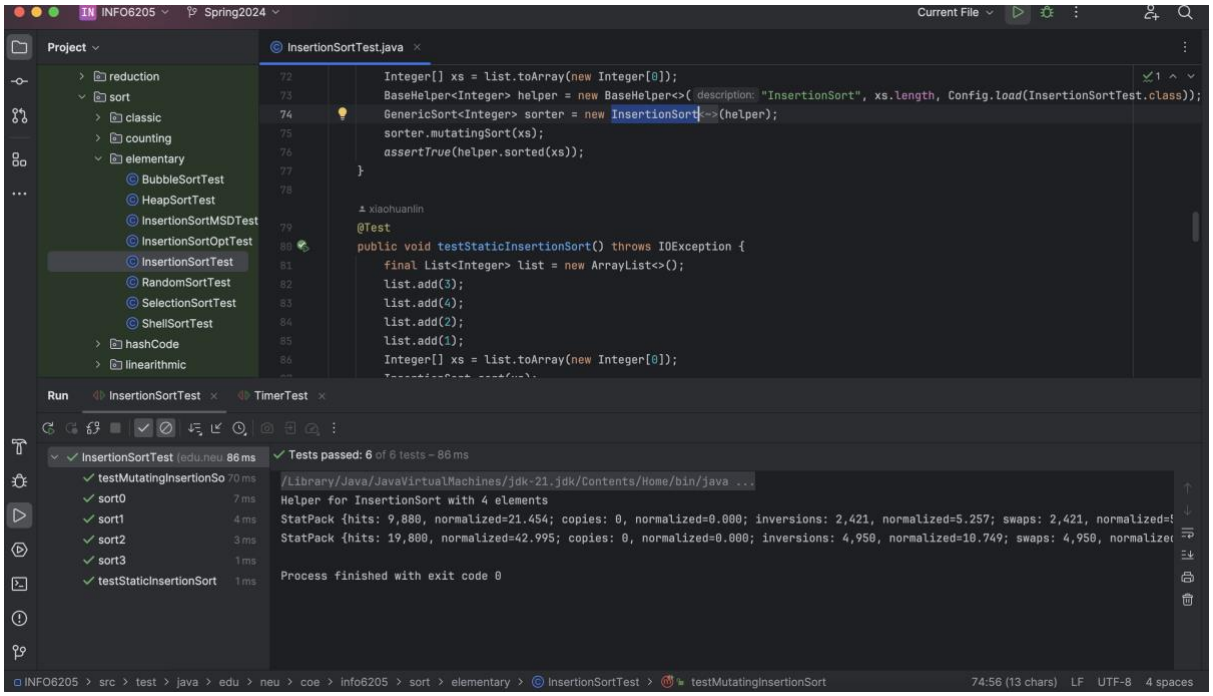



## Runs Screenshot

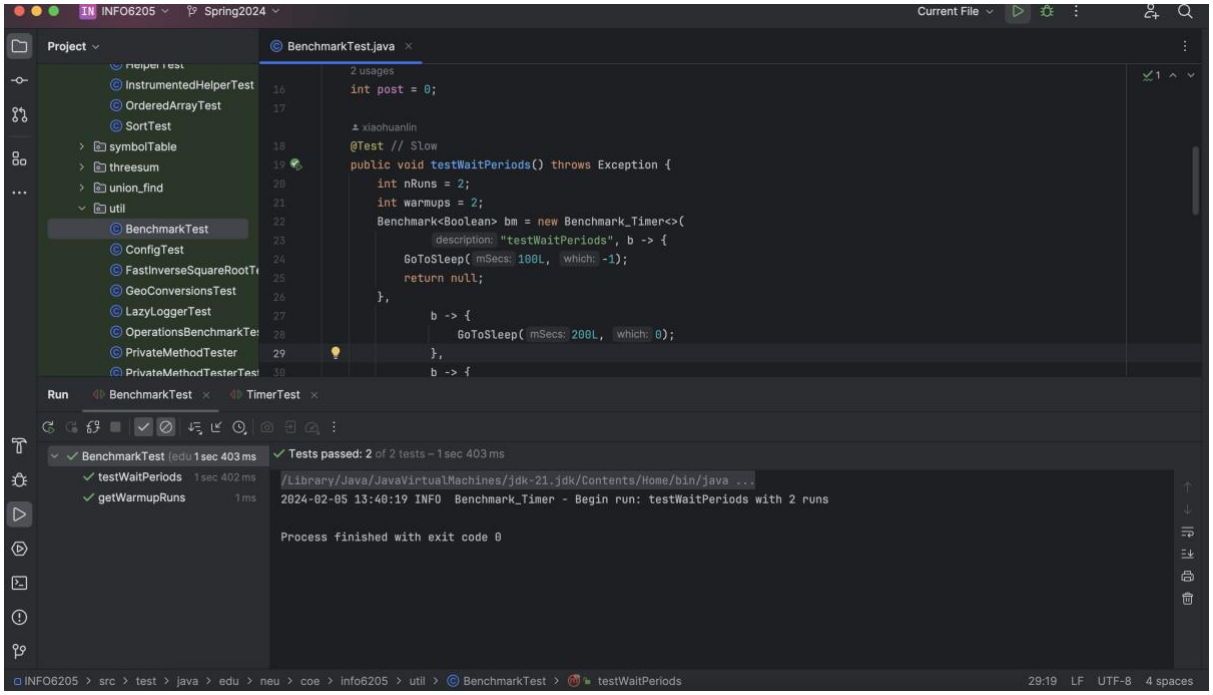


## Unit Test Screenshots

InsertionSortTest – 6 of 6 tests passed.




BenchmarkTest – 2 of 2 Tests passed.



TimerTest – 11 of 11 tests passed.