Mohit Negi (001549103)

Program Structures & Algorithms

Fall 2021

Assignment No. 2

- ## Task (List down the tasks performed in the Assignment)
  - **Timer class:**
    - Implemented repeat, getClock and toMillisecs functions.
    - Returning mean lap time in repeat function for the whole process, starting from pre-function, to applying the function {insertion sort} and finally post function to check the results.
  - **Becnchmark_Timer class:**
    - Added a main function. We run the insertion sort class over four different array types, namely: "Random", "Ordered", "Reverse" and "Partial".
    - The time is recorded for these four types of arrays. We assign each array with minimum of 500 elements
  - **InsertionSort class**:
    - Implemented the sort function.

- ## Output & Unit test results:

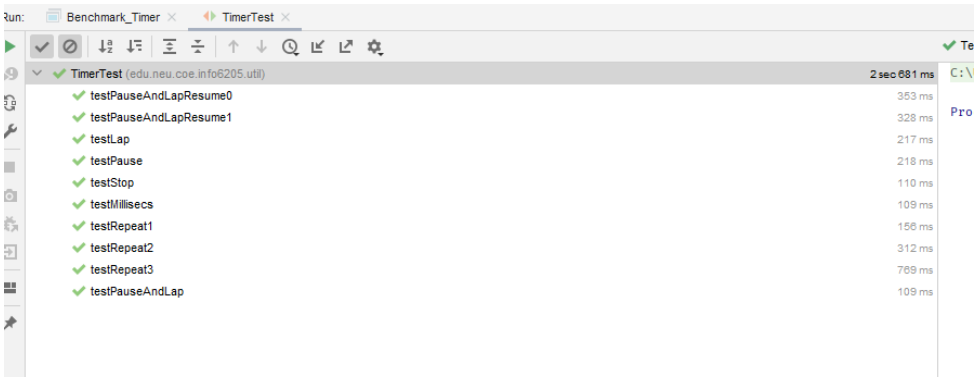

Fig.1: Timer Test

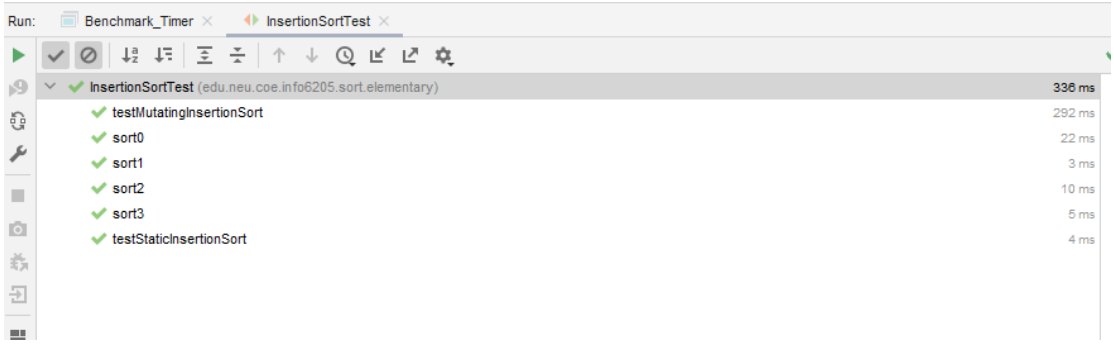

Fig.2: InsertionSortTest

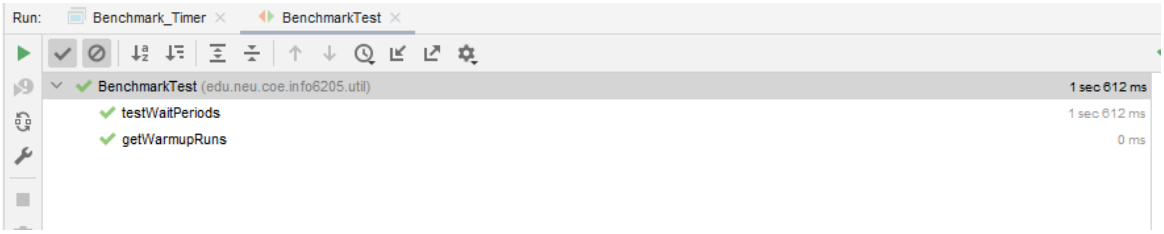

Fig.3: BenchmarkTest

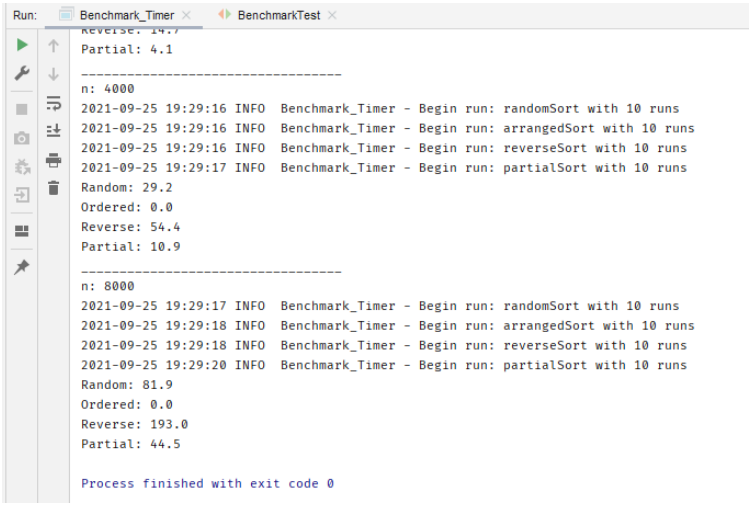

Fig.4: Benchmark_Timer

- **Observation:**

n	500	1000	2000	4000	8000
Random	2 ms	1.9 ms	8.3 ms	32.8 ms	87 ms
Ordered	0 ms	0 ms	0 ms	0 ms	0 ms
Reverse	4.4 ms	3.9 ms	16.1 ms	60.2 ms	186.2 ms
Partial	0.3 ms	0.9 ms	4.1 ms	9.5 ms	77.3 ms

From the above table we can infer the following:

- When the elements are doubled, the time taken growth by the algorithm is quadratic in nature. The average being $O(n^2)$.
- For the ordered arrays, time remained zero since only traversing operation took place and no elements were swapped. Here the time taken is $O(n)$.

- The reverse ordered arrays provided us with the maximum time taken out of all the four.
- Partially ordered arrays provided us with numbers which are in between random and ordered arrays.
- We can easily identify the advantages of having a non-quadratic algorithm when time is the factor we are trying to minimize.