Program Structures and Algorithms
Spring 2023(SEC –03)

NAME: Patel Dhruv Rajeshkumar
NUID: 002928881

Task: Implement Benchmarks on Insertion Sort with different types of arrays. The arrays used are following:

- Random array of integers
- Sorted array of integers
- Reverse sorted array of integers
- Partially sorted array of integers (half sorted and half random)

Relationship Conclusion: After comparing time taken by all the arrays, it is visible that reverse sorted array took longest to get sorted by the insertion sort algorithm (Theoretically, this is the worst case of insertion sort). Also, the already sorted array took the least time. (Sorted array is the best case for insertion sort). These can be proved by the given curves.

Time taken for different type of array of integers by insertion sort is (in ms) :
Sorted array < Partially sorted < Random array < Reverse sorted array

Time complexity of Reverse sorted array will be $O(n^2)$.
Time complexity of sorted array will be O(n).

Thus, we can conclude that if the array has already sorted values, it is better to use Insertion Sort.

A few real-life examples where data is partially or fully sorted are:

- Stock prices
- Employee salaries
- Weather forecast results

Why does insertion sort works better for partially sorted data?

Whenever a key finds a value smaller than itself, the while loop is stopped. Thus, we only have to travel through the array (the For loop).

If for example:-
For loop is executed n times
While loop is executed m times

Then time complexity for the insertion sort is O(n * m)

Now, for a sorted array, while loop is executed only once (to check the condition), which makes time complexity of while loop O(1)
Thus, the overall time complexity is O(n * 1) which is O(n) (linear time).

Evidence to support that conclusion:

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 2000 with 10 runs
Time taken to sort random array - 3.0168247

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 2000 with 10 runs
Time taken to sort sorted array - 0.00435

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 2000 with 10 runs
Time taken to sort reversed ordered array - 4.9909709

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 2000 with 10 runs
Time taken to sort partially ordered array - 1.9111417

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 4000 with 10 runs
Time taken to sort random array - 9.9668751

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 4000 with 10 runs
Time taken to sort sorted array - 0.0084207

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 4000 with 10 runs
Time taken to sort reversed ordered array - 19.9250294

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 4000 with 10 runs
Time taken to sort partially ordered array - 7.2896833999999995

2023-02-04 22:47:16 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 8000 with 10 runs
Time taken to sort random array - 43.5695543

2023-02-04 22:47:17 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 8000 with 10 runs
Time taken to sort sorted array - 0.0153334

2023-02-04 22:47:17 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 8000 with 10 runs
Time taken to sort reversed ordered array - 79.9307417

2023-02-04 22:47:18 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 8000 with 10 runs
Time taken to sort partially ordered array - 30.364045800000003

2023-02-04 22:47:18 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 16000 with 10 runs
Time taken to sort random array - 177.049846

2023-02-04 22:47:20 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 16000 with 10 runs
Time taken to sort sorted array - 0.027491599999999998

2023-02-04 22:47:20 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 16000 with 10 runs
Time taken to sort reversed ordered array - 308.3564377

2023-02-04 22:47:24 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 16000 with 10 runs
Time taken to sort partially ordered array - 116.19991660000001

2023-02-04 22:47:25 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 32000 with 10 runs
Time taken to sort random array - 665.5481833

2023-02-04 22:47:33 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 32000 with 10 runs
Time taken to sort sorted array - 0.053712499999999996

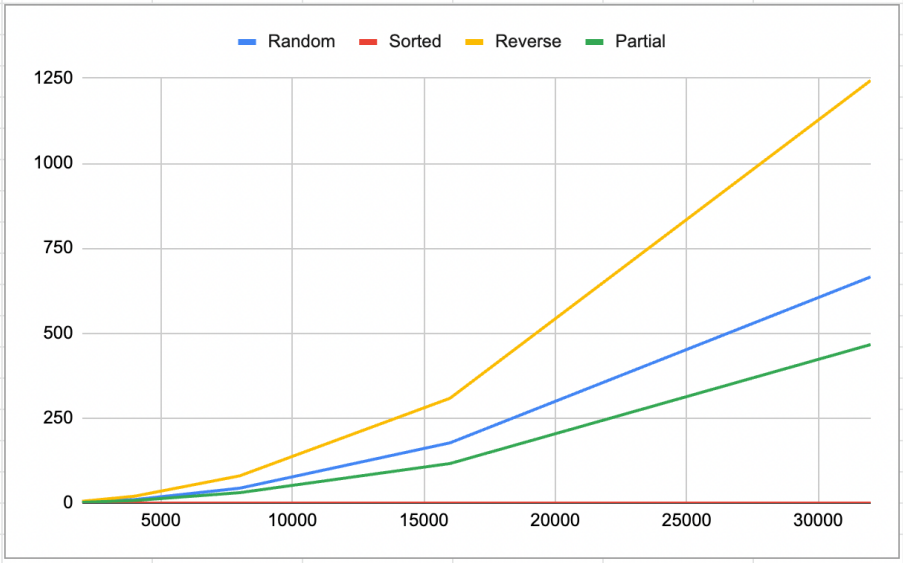
2023-02-04 22:47:33 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 32000 with 10 runs
Time taken to sort reversed ordered array - 1243.8425791

2023-02-04 22:47:48 INFO Benchmark_Timer - Begin run: Insertion Sort with array size 32000 with 10 runs
Time taken to sort partially ordered array - 466.3646

Process finished with exit code 0

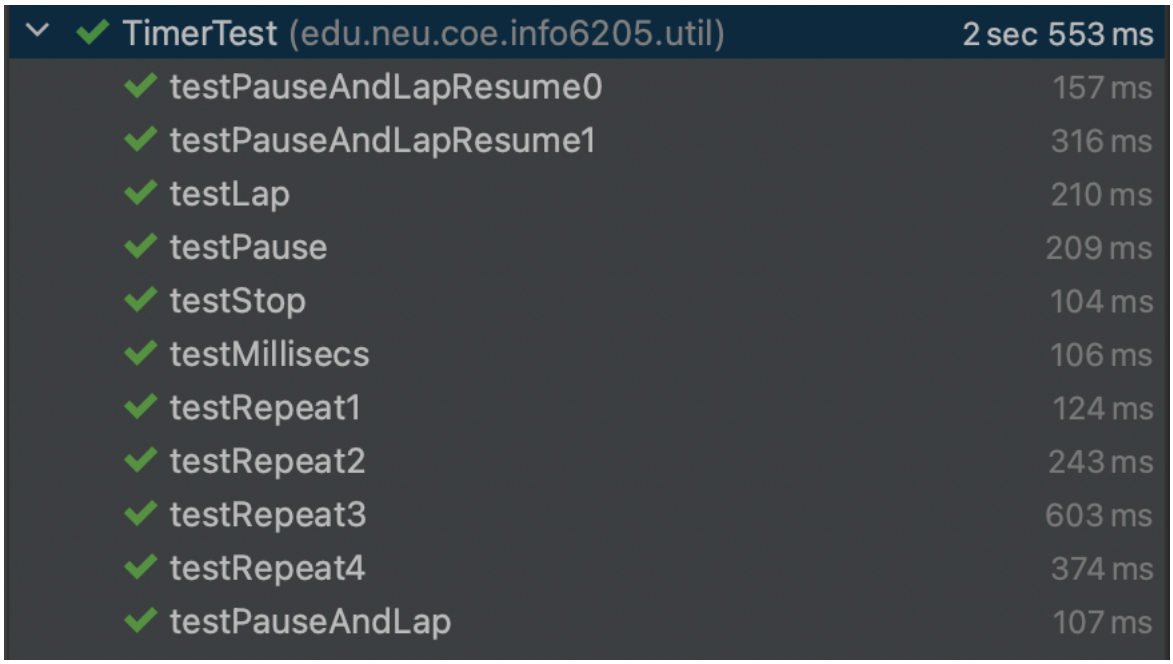
Graphical Representation:

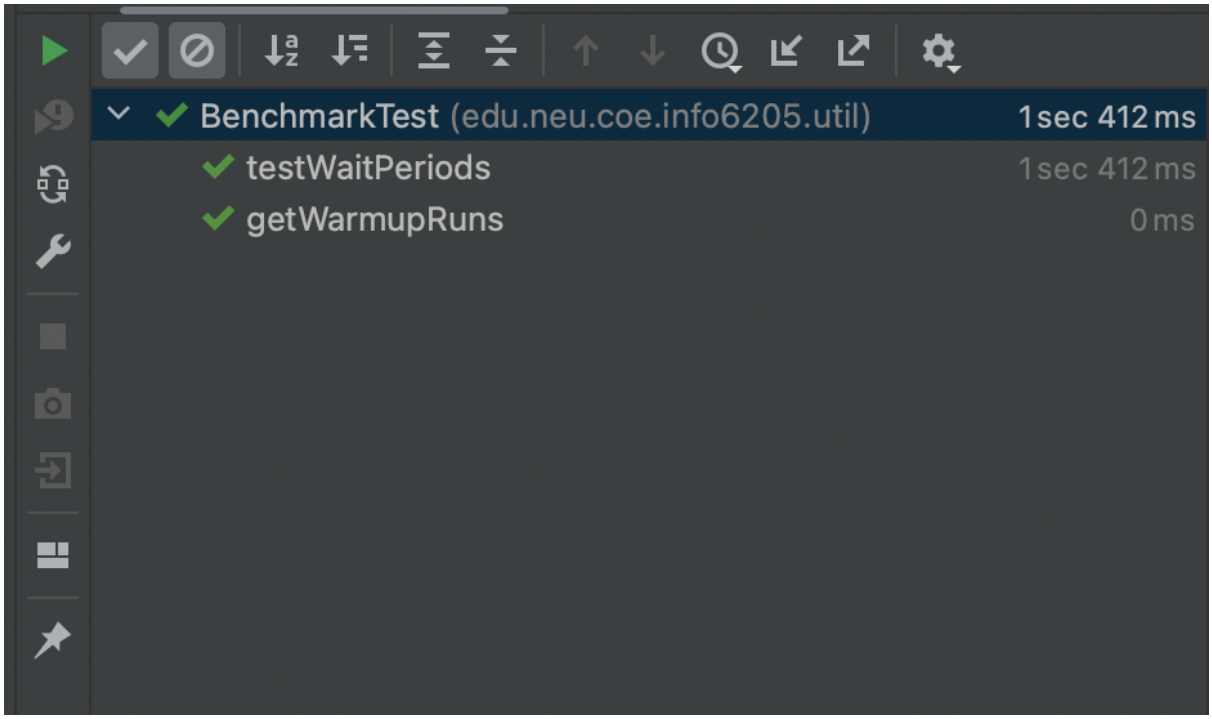
	2000	4000	8000	16000	32000
Random	3.0168247	9.9668751	43.5695543	177.049846	665.5481833
Sorted	0.00435	0.0084207	0.0153334	0.0274916	0.0537125
Reverse	4.9909709	19.9250294	79.9307417	308.3564377	1243.842579
Partial	1.9111417	7.2896834	30.3640458	116.1999166	466.3646

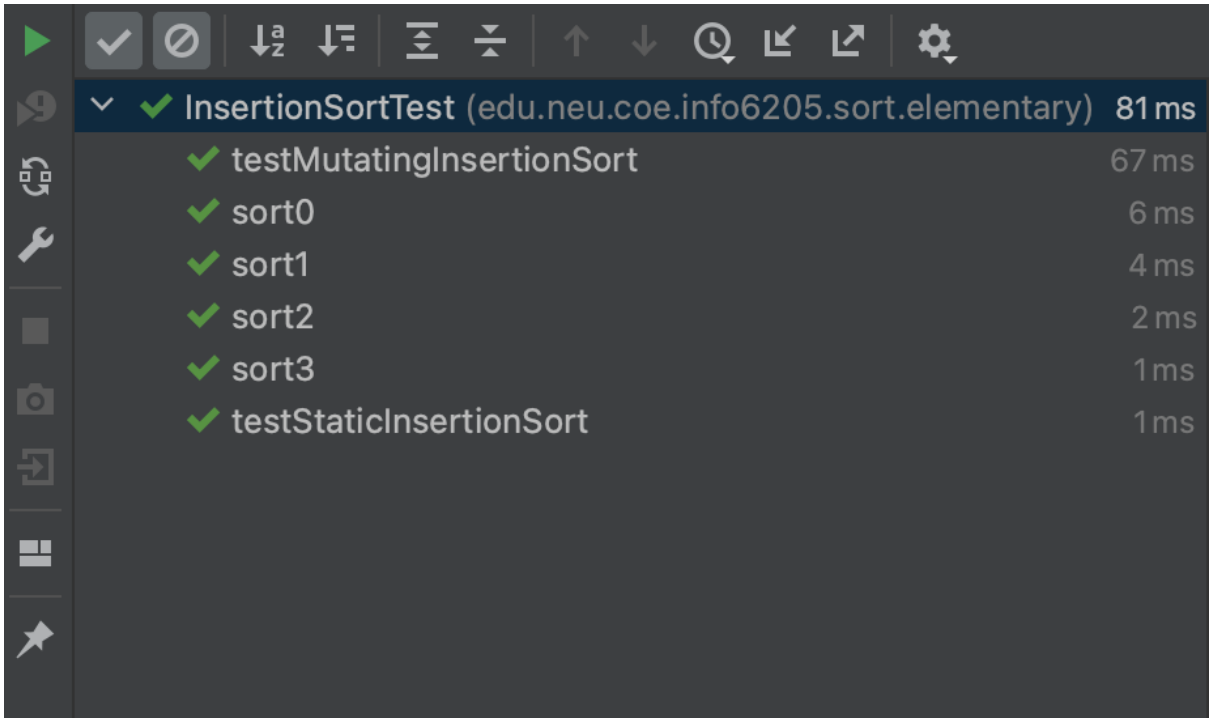


Here, x axis represents size of array of integers and y axis represents average time taken by insertion sort to sort the array in milliseconds.

Unit Test Screenshots:






InsertionSortTest (edu.neu.coe.info6205.sort.elementary) 81 ms
testMutatingInsertionSort 67 ms
sort0 6 ms
sort1 4 ms
sort2 2 ms
sort3 1 ms
testStaticInsertionSort 1 ms