

INFO6205 –Assignment2

Manasa Bhat

Section 5

NU ID: 001032278

Main Task Solution

1. Implement three methods of Timer class.

I have implemented methods as below:

i) repeat : implemented to record timing for function processing for input n repeat times and used logic to pause during prefunction and post function processing. In the end it will return the time taken by calling meanLapTime().

ii) getClock : return the number of ticks from system clock in nanoTime.

iii) toMillisecs: convert and return time in milliseconds.

2. Implement InsertionSort.

As per instructions, I have successfully implemented InsertionSort using helper.swap method.

3. Implement a main program to measure the running times of this sort.

I have created a new program called **InsertionSortTimerTest** to measure the running times of insertion sort using 4 different initial array ordering situations.

Sort0() ---> provides mean time for calculating Insertion Sort on Ordered Arrays.

Sort1() ---> provides mean time for calculating Insertion Sort on Reverse Ordered Arrays.

Sort2() ---> provides mean time for calculating Insertion Sort on Partially Ordered Arrays.

Sort3() ---> provides mean time for calculating Insertion Sort on Random Arrays.

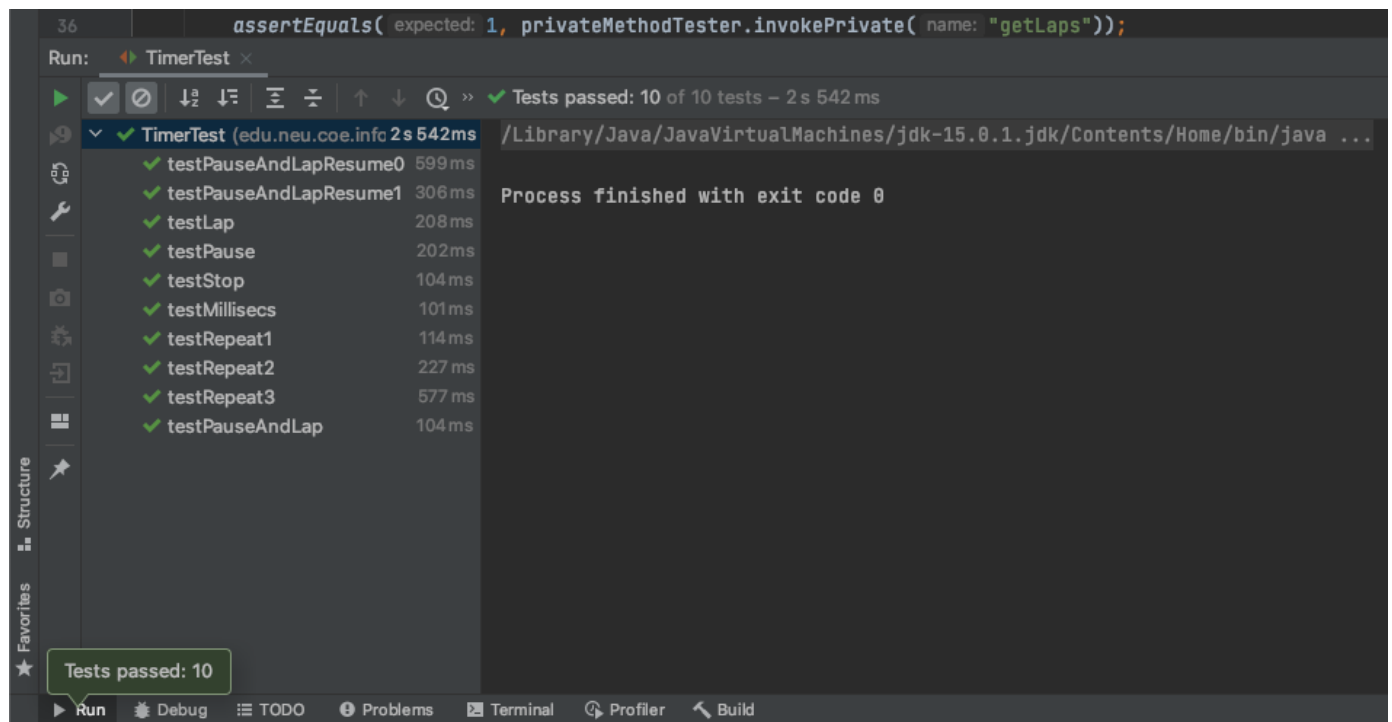
Further in this report you can see using doubling method I have drawn observations and conclusion by testing for 5 values of n.

Note: I have included the new main program InsertionSortTimeTest under test folder(Same folder as InsertionSortTest) in the repository that I have submitted in Git.

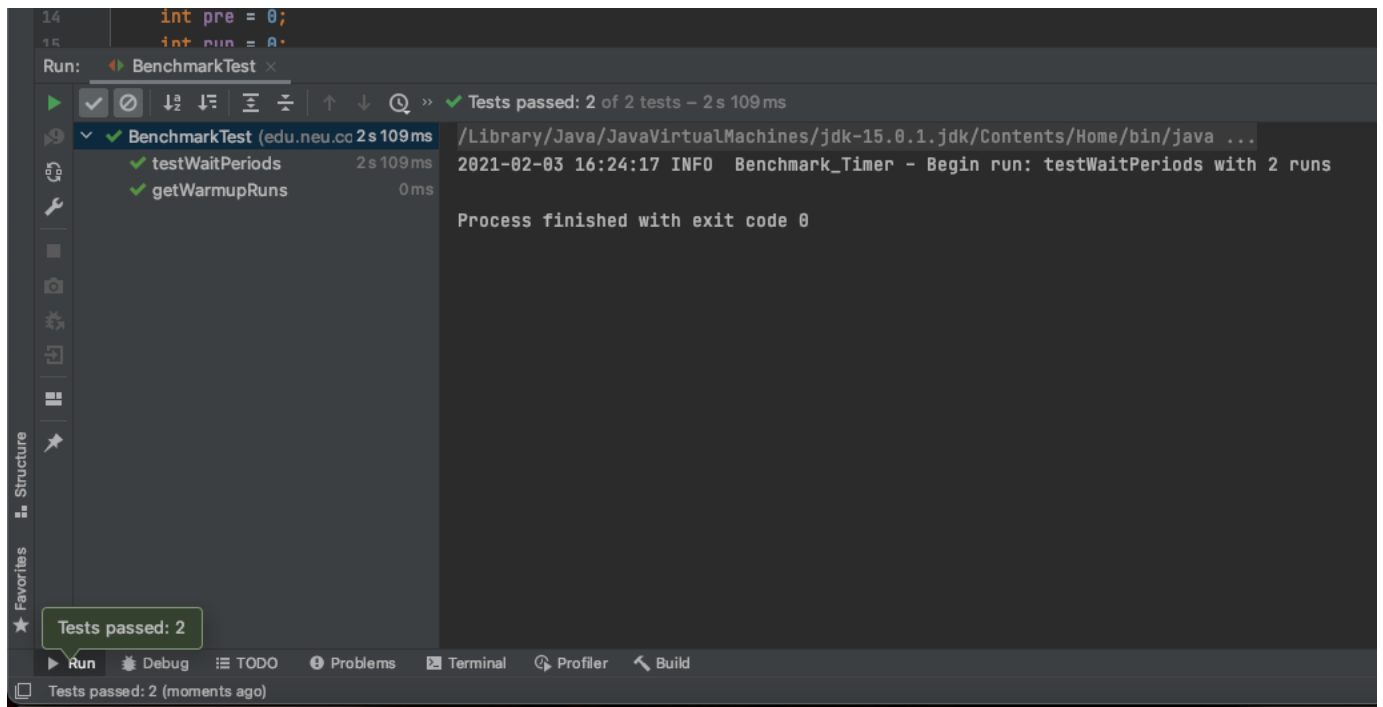
UnitTest Results:

Below are the three passed testcase outputs for three unit test programs.

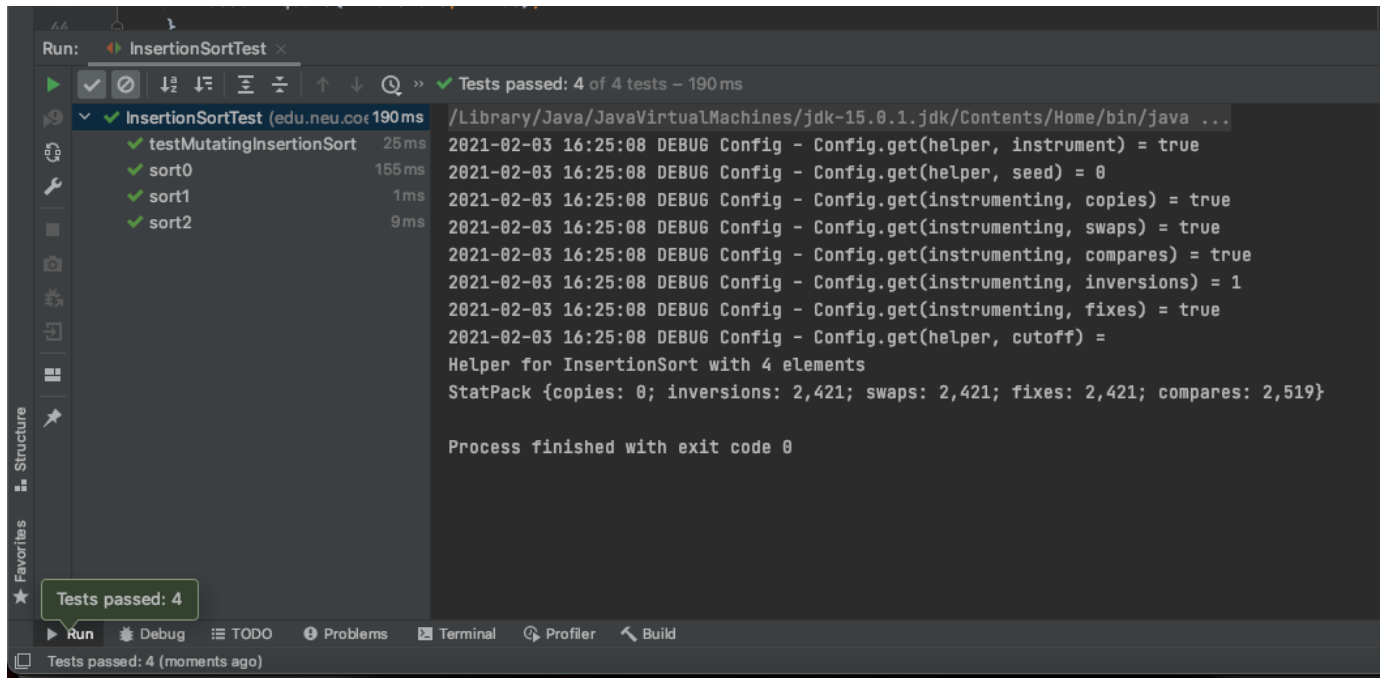
1. TimerTest:



2. BenchmarkTest:



3. InsertionSortTest:



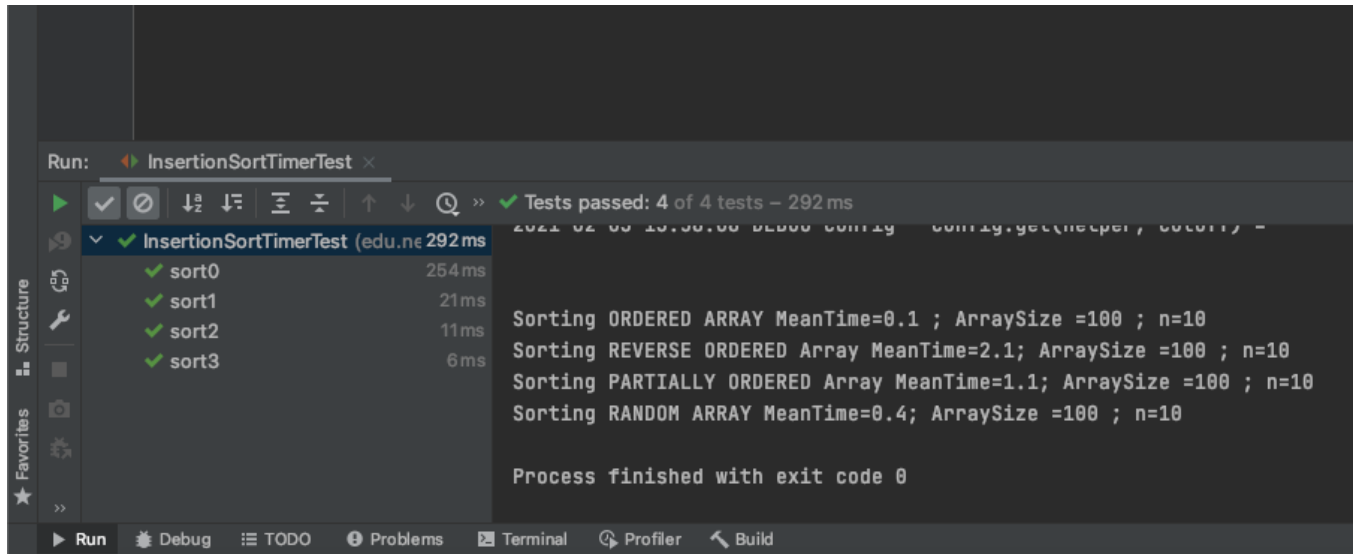
Console Output:

I created a main program called **InsertionSortTimerTest** to pass 4 differently ordered arrays to insertion sorting and tracked the time using Timer program.

I have used doubling method to increase the array size to observe order of growth.

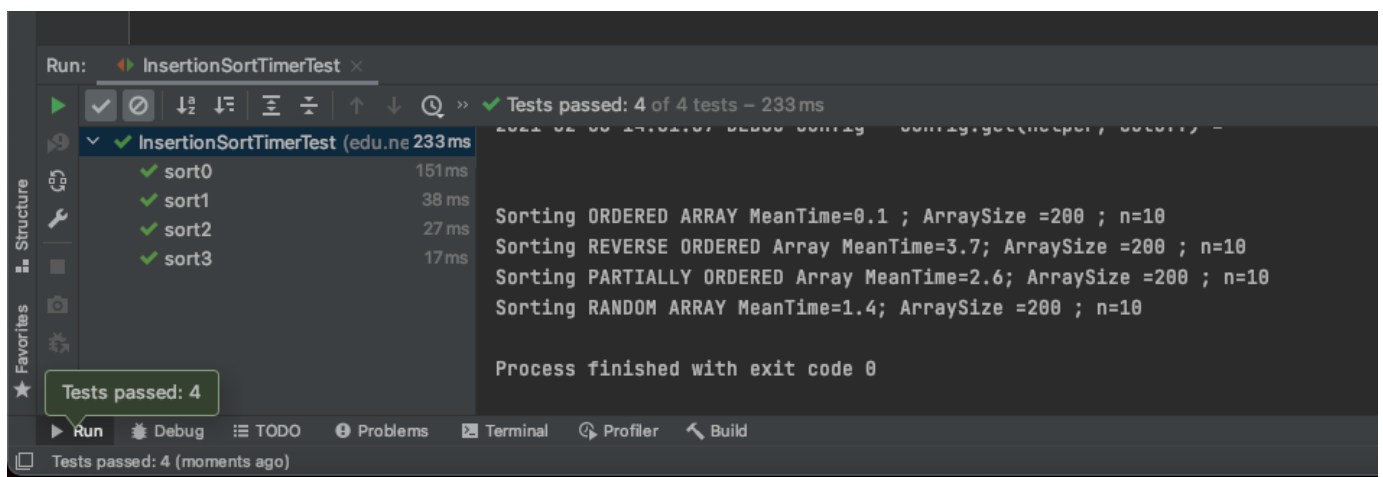
I have run multiple times to notice the growth pattern with different array sizes and repeat numbers(n).

1. Console Output for n=10; array size = **100**



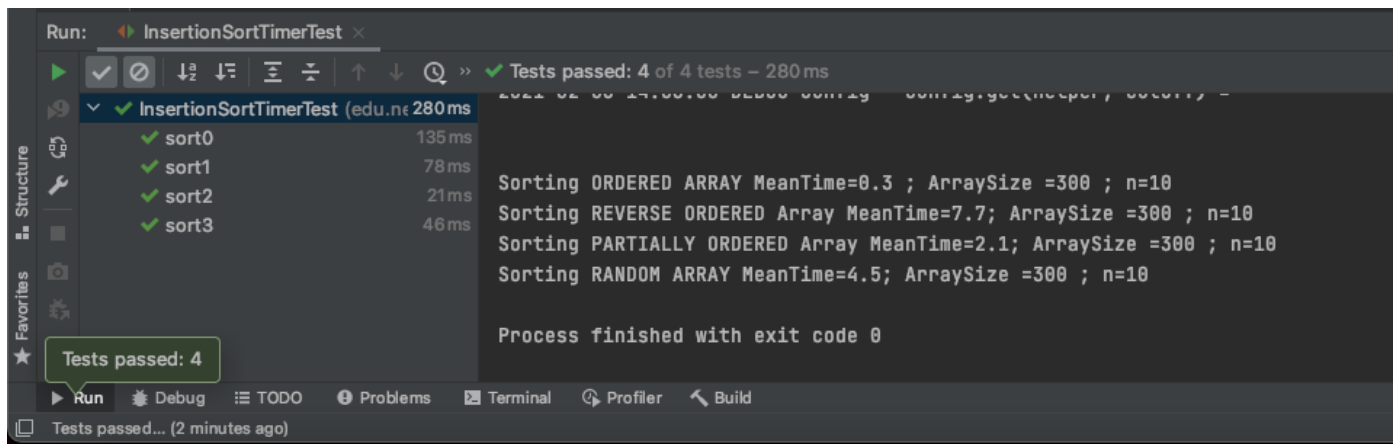
```
Run: InsertionSortTimerTest x
Tests passed: 4 of 4 tests - 292 ms
InsertionSortTimerTest (edu.ne 292 ms)
  sort0 254 ms
  sort1 21 ms
  sort2 11 ms
  sort3 6 ms
Sorting ORDERED ARRAY MeanTime=0.1 ; ArraySize =100 ; n=10
Sorting REVERSE ORDERED Array MeanTime=2.1; ArraySize =100 ; n=10
Sorting PARTIALLY ORDERED Array MeanTime=1.1; ArraySize =100 ; n=10
Sorting RANDOM ARRAY MeanTime=0.4; ArraySize =100 ; n=10
Process finished with exit code 0
```

2. Console Output for n=10; array size = **200**

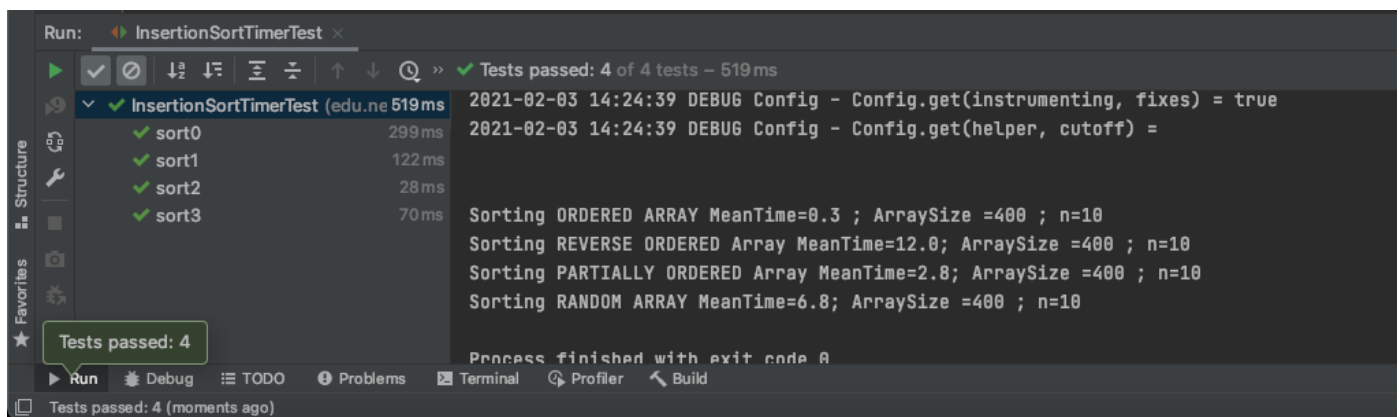


```
Run: InsertionSortTimerTest x
Tests passed: 4 of 4 tests - 233 ms
InsertionSortTimerTest (edu.ne 233 ms)
  sort0 151 ms
  sort1 38 ms
  sort2 27 ms
  sort3 17 ms
Sorting ORDERED ARRAY MeanTime=0.1 ; ArraySize =200 ; n=10
Sorting REVERSE ORDERED Array MeanTime=3.7; ArraySize =200 ; n=10
Sorting PARTIALLY ORDERED Array MeanTime=2.6; ArraySize =200 ; n=10
Sorting RANDOM ARRAY MeanTime=1.4; ArraySize =200 ; n=10
Process finished with exit code 0
Tests passed: 4
```

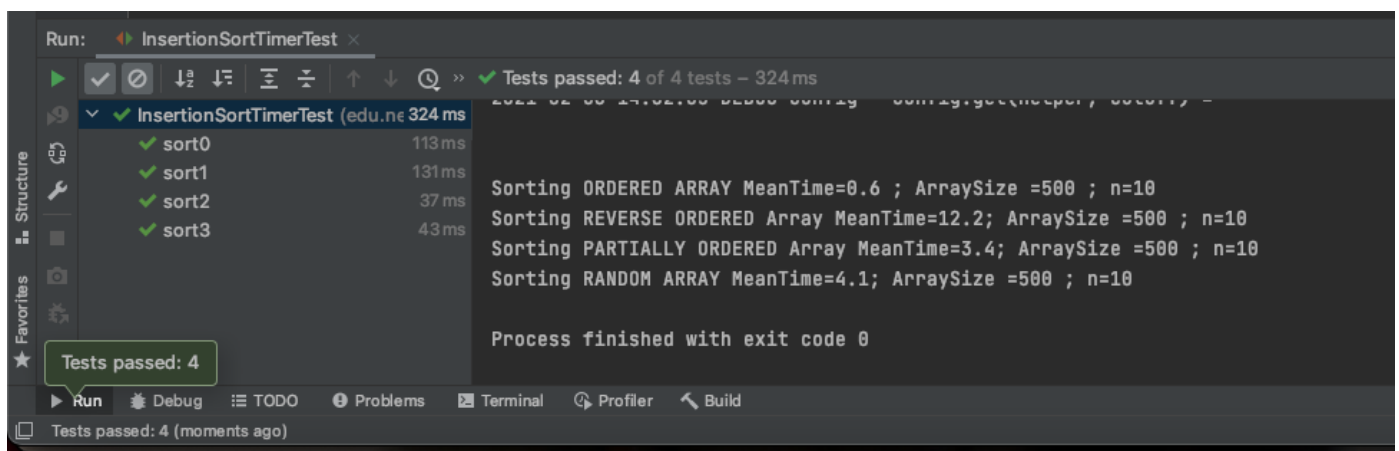
3. Console Output for n=10; array size = **300**



4. Console Output for n=10; array size = **400**



5. Console Output for n=10; array size = **500**



Observation of Order of Growth

I conducted several runs for a combination of both different array sizes and repeat numbers. In all the runs I could see there was an increase in the mean time for sorting as the array size increased.

The order of growth was completely scattered with several runs. There was no one particular order of growth for all the 4 types of arrays taken. I then started analyzing the order individually for different array types passed.

I observed that for Ordered and Partially ordered arrays, the mean time for insertion sort grew by 0.1, 0.2 milliseconds as the array size increased, whereas for reverse ordered arrays the mean time had a drastic increase by 5 to 10 milliseconds. Random ordered arrays had a mixed response in their order of growth.

I was then able to understand that the order of growth depends on the number of inversions needed in the array to sort. In case of Partially ordered or Ordered arrays, the number of inversions is less hence the mean time for sorting these arrays is lesser and also this is the reason there is smaller order of growth. Considering Reverse Ordered array, the number of inversions is maximum, hence mean time to sort them is high. We could also find the mean time for sorting random array if we could get the number of inversions in the array.

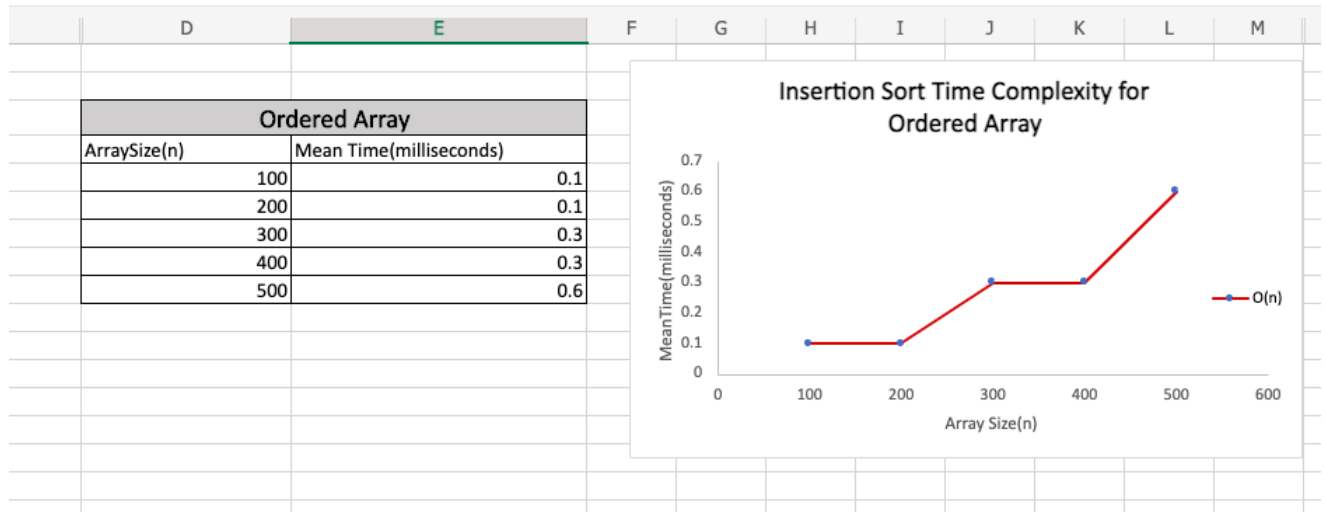
Below are the observations:

- In case of Ordered or Partially ordered arrays, there is gradual increase in the mean time taken for sorting with increase in array size.
- In case of Random Ordered arrays, the order of growth with increase in array size is random and can be found by knowing the number of inversions in the array.
- For Reverse Ordered Arrays, it takes higher mean time for sorting and there is drastic increase in order of growth with increase in array size.

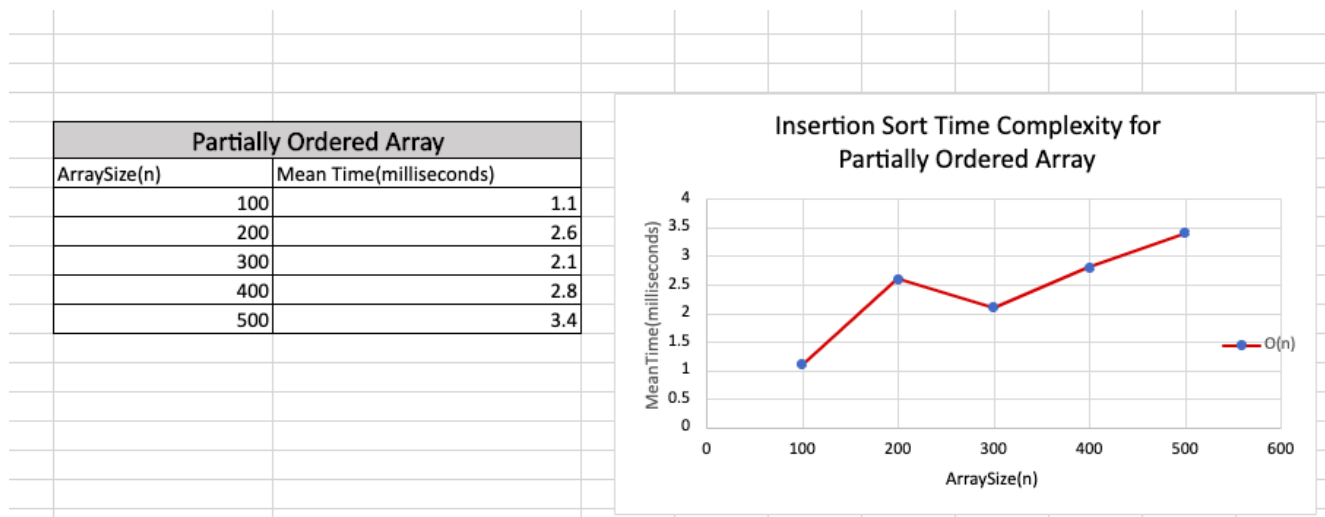
Supporting Evidence

I have plotted time taken for sorting versus array size graphs for each type of initial array ordering situations.

Graph1: Order of Growth for **Ordered** Array Insertion Sort.

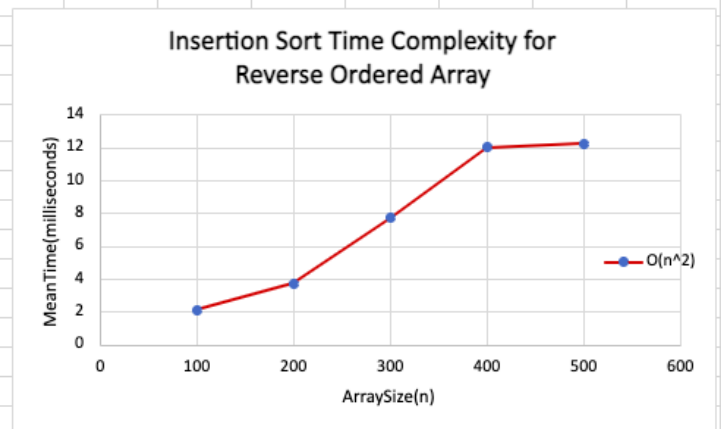


Graph2: Order of Growth for **Partially Ordered** Array Insertion Sort.



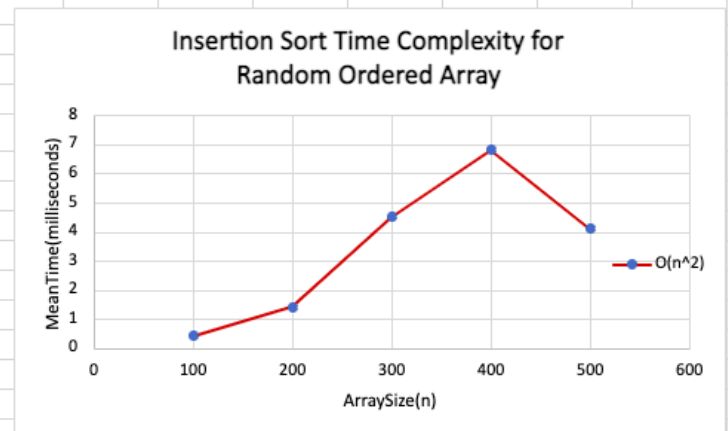
Graph3: Order of Growth for **Reverse Ordered** Array Insertion Sort.

Reverse Ordered Array	
ArraySize(n)	Mean Time(millisecond)
100	2.1
200	3.7
300	7.7
400	12
500	12.2



Graph4: Order of Growth for **Randomly Ordered** Array Insertion Sort.

Random Ordered Array	
ArraySize(n)	Mean Time(millisecond)
100	0.4
200	1.4
300	4.5
400	6.8
500	4.1



As seen in the plots, the rate of growth in Ordered and partially ordered arrays is 0.1 to 0.5 milliseconds. There is a gradual increase of meantime with increase in array size. As per the observations, the order of growth for Ordered and partially ordered arrays for n array size can be deduced to be $O(n)$.

As shown in Graph3, The rate of growth for reverse ordered array is 3 to 5 milliseconds. There is a drastic increase of meantime with increase in array size. Hence based on the observations, the order of growth for reverse ordered array of n array size can be deduced to be $O(n^2)$.

The growth rate fluctuates for randomly ordered array sorting. But considering extreme case, the random array could have maximum inversions (which is similar to a reverse ordered array) and the order of growth could be deduced as $O(n^2)$.

Conclusion

From the test results and observations, it can be concluded that Insertion Sorting time increases with increase in the array size. The rate of increase depends on the number of inversions in the array.

- For an Ordered or Partially Ordered array, the number of inversions are lesser, the order of growth is gradual with increase in array size.

The Sorting time complexity for Ordered or Partially Ordered array of array size 'n' can be given as $O(n)$ (BigO of n).

- For a Randomly Ordered Array, the inversions can vary from 0 to maximum. We can find the order of growth by knowing the number of inversions in this array type. Considering maximum inversion that is possible, this array will behave like a reverse ordered array.
- For a Reverse Ordered Array, there is highest inversions, the order of growth is drastic in this case.

Sorting Time complexity for Reverse Ordered Array or Randomly ordered array of n size can be given as: $O(n^2)$ (Big O of n square).