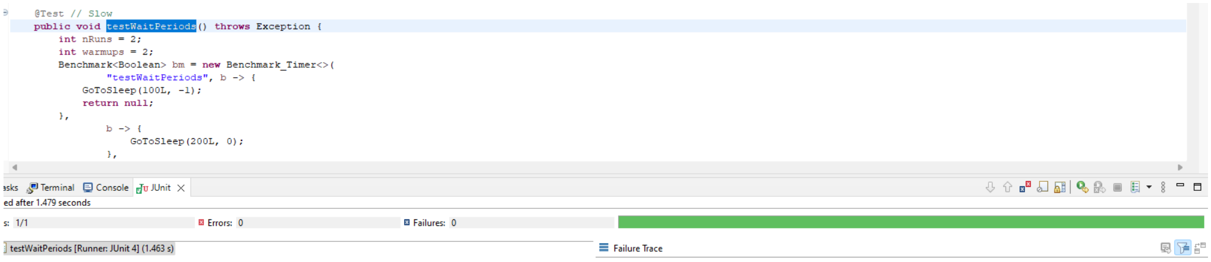
INFO6205 Assignment 3 Insertion sort

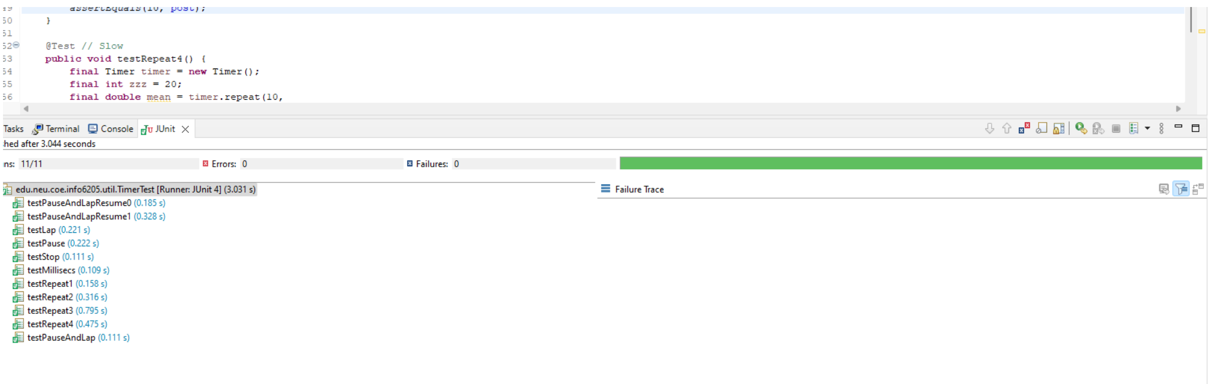
Ruizhe Zeng

1. Unit Tests :

a. Benchmark test



b. Timer test



c. Insertion sort test

2. Program output

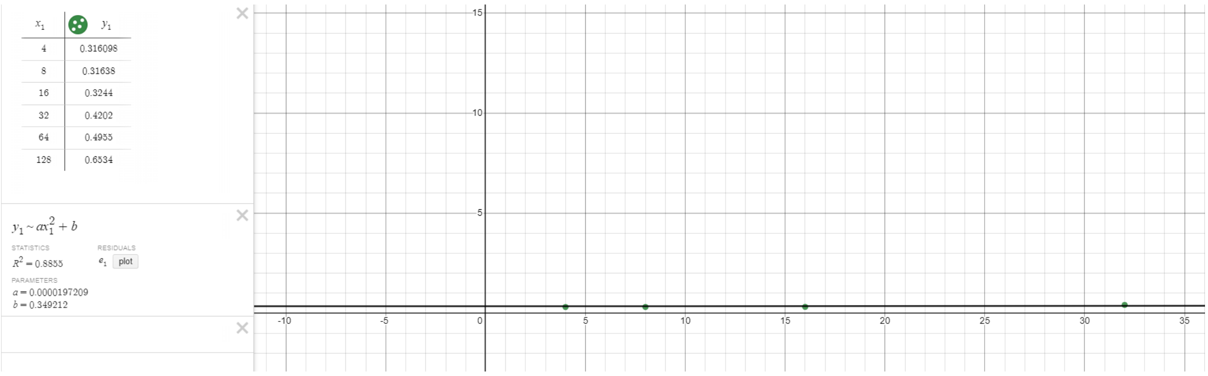
```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 400 run 100 times:
Sorted array : 0.316098 (msec)
Random array : 0.6057429999999999 (msec)
Revert array : 0.6646589999999999 (msec)
First half sorted array : 0.596866 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 800 run 100 times:
Sorted array : 0.31638 (msec)
Random array : 0.9457249999999999 (msec)
Revert array : 1.5212099999999997 (msec)
First half sorted array : 0.8917860000000001 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 1600 run 100 times:
Sorted array : 0.32449999999999996 (msec)
Random array : 2.951425 (msec)
Revert array : 5.2999410000000005 (msec)
First half sorted array : 2.38869 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 3200 run 100 times:
Sorted array : 0.42021499999999995 (msec)
Random array : 10.350585999999998 (msec)
Revert array : 19.168796999999998 (msec)
First half sorted array : 7.875369999999999 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 6400 run 100 times:
Sorted array : 0.49558199999999997 (msec)
Random array : 39.116265 (msec)
Revert array : 75.475143 (msec)
First half sorted array : 29.497257 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Array with size of 12800 run 100 times:
Sorted array : 0.653442 (msec)
Random array : 153.36466199999998 (msec)
Revert array : 304.057242 (msec)
First half sorted array : 102.66385 (msec)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

In order to get more accurate data, I run each code 100 times, and each time I double the length of the input array. Let n be the length of the array, I have n = 400,800,1600,3200,6400,12800. The reason why I start with 400 is because , when n<400 runs too fast, it is tricky to see the relationships.

3. **Graphs and Conclusion**

Notice x_1 is the number of n while n = 100, so x_1 means the array.length = 400.

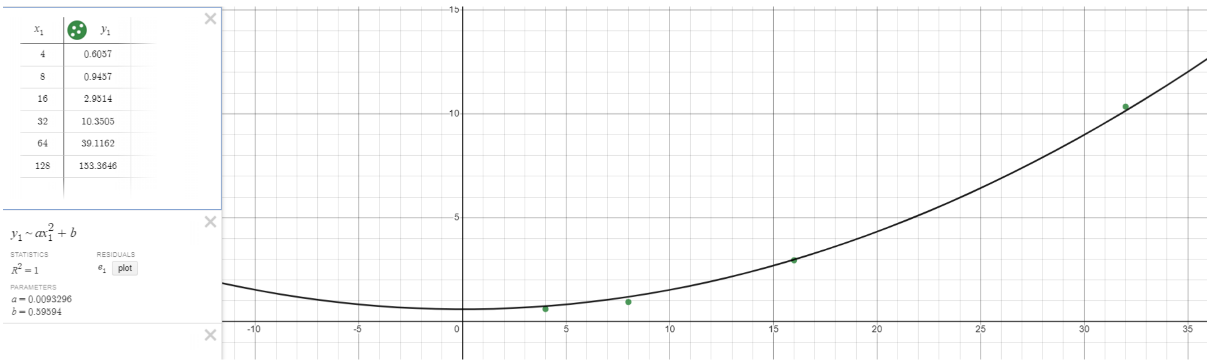
A. **Sorted array**



Time = 0.0000197209n^2+0.349212

Grow at a very slow rate.

B. **Random array**



Time = 0.0093296n^2+0.59594

Grow much faster than the sorted array.

C. Reversed array

x_1	y_1
4	0.6646
8	1.5212
16	5.2999
32	19.1687
64	75.4751
128	304.0572

$y_1 \sim ax_1^2 + b$

STATISTICS

$R^2 = 1$

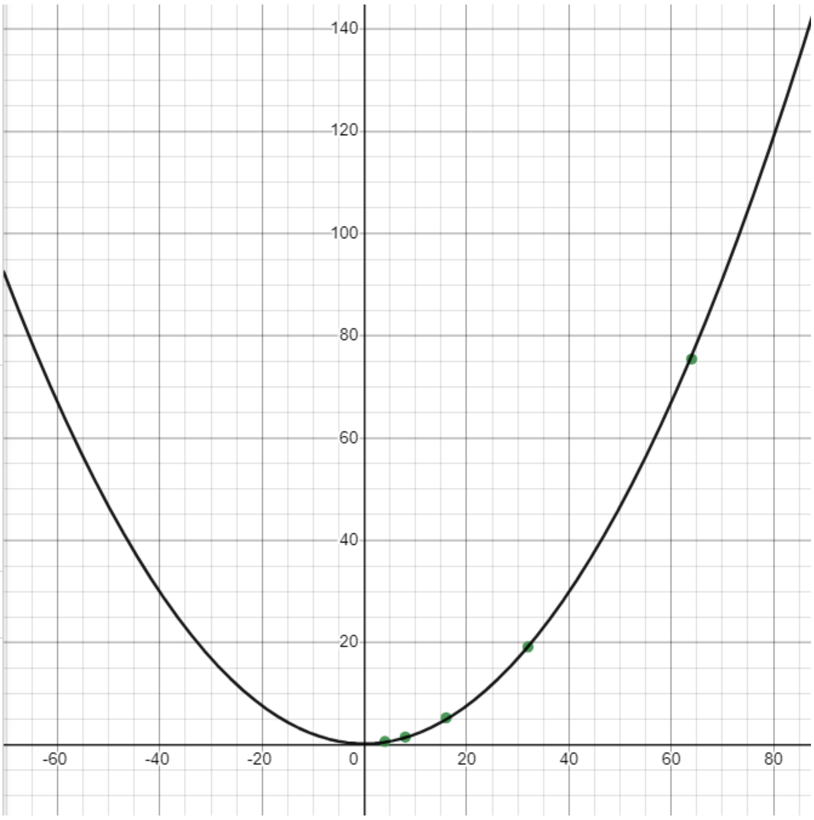
RESIDUALS

e_1 plot

PARAMETERS

$a = 0.0185337$

$b = 0.235073$



Time = 0.0185n^2 + 0.235

Its time has the fastest growth rate. When double the n, it requires a significant amount of time to process. The worst case of the insertion sort.

D. First half sorted array

x_1	y_1
4	0.5968
8	0.8917
16	2.3886
32	7.8733
64	29.4972
128	102.6638

$y_1 \sim ax_1^2 + b$

STATISTICS

$R^2 = 0.9988$

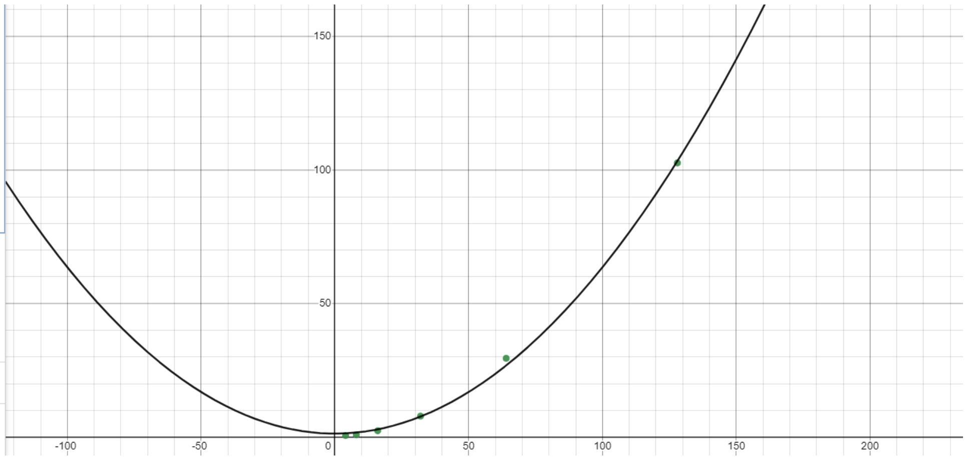
RESIDUALS

e_1 plot

PARAMETERS

$a = 0.00622608$

$b = 1.32264$



Time = 0.00622608n^2+1.3226

Its growth rate is somewhere between the sorted array and random array.

Conclusion:

For every set of data we can clearly see, the time **T** is :

T Sorted < T Half sorted < T Random < T Reversed

Which makes when the input array is in reversed order is the **WORST** case of insertion sort, and if the input array is in the sorted order, that makes it the **BEST** case of insertion sort. This result definitely met our expectations. Consider the number of swaps we did during insertion sort. If the array is already sorted, no swaps are needed, that is why the sorted array runs very fast. If the array is half sorted, it only needs to do the swaps in the other half of the array, which makes it slower than sorted, but faster than a completely random array. And Reversed will do the every possible array the expectation will be :

let n = array.length
number of swap the reversed array has to do = 1+2+3+4+.....(n-1) = (n(n+1))/2
So it is O(n^2).

in data we have size = 6400, time = 75.4751 msec
let (n(n+1))/2 = 75.4751
n=11.7963
plug in into (2n(2n+1))/2 we have 290.1016
so I predict when size doubled -> size = 12800 time = 290.1016
In data we have size = 12800 time =304.0573 close to our prediction

The following is the equations I find of relationship of each type of array:

Sorted : Time = 0.0000197209n^2+0.349212

Random: Time = 0.0093296n^2+0.59594

Reversed: Time = 0.0185n^2 + 0.235

Half-Sorted: Time = 0.00622608n^2+1.3226

Please check the Graphs section for proof.