Student: PinHo Wang
Instructor: Prof. Robin Hillyard
NUID: 001443435

INFO 6205 Program Structure & Algorithms
Summer Full 2018
Assignment 3 - Benchmarking

Files Description:

+- PinHo_Wang_assignment3

+- Report

+- src

+- main

| +- edu.neu.coe.info6205.sort.simple

| +- Helper.java (default)

| +- Sort.java (default)

| +- InsertionSort.java : Implement insertion sort algorithm

| +- SelectionSort.java: Implement insertion sort algorithm

| +- ShellSort.java: Implement shell sort algorithm

| +- edu.neu.coe.info6205.util

| +- Benchmark.java: main function

|

+- test

| +- edu.neu.coe.info6205.sort.simple

| +- InsertionSortTest.java: Test InsertionSort.java

| +- SelectionSortTest.java: Test SelectionSort.java

| +- ShellSortTest.java: Test ShellSort.java

| +- edu.neu.coe.info6205.util

+- BenchmarkTest.java: Test Benchmark.java

Experiments:

In the Benchmark.java, I used **java Arrays.sort()** method to create four different order array, random, ordered, partial-ordered and reversed-ordered, with five different size (n). Then, I used these array as inputs and put them into three sorting methods. The result shows below.

n = 1000

```
n = 1000
Input Array: RANDOM
InsertionSort: 1000: 1.4312821199999999 millisecs
SelectionSort: 1000: 1.10581603 millisecs
ShellSort:    1000: 0.59180161 millisecs
Input Array: ORDERED
InsertionSort: 1000: 0.01254618 millisecs
SelectionSort: 1000: 0.7946016899999999 millisecs
ShellSort:    1000: 0.01860634 millisecs
Input Array: PARTIAL
InsertionSort: 1000: 0.475578 millisecs
SelectionSort: 1000: 0.8779677499999999 millisecs
ShellSort:    1000: 0.13014859 millisecs
Input Array: REVERSE
InsertionSort: 1000: 2.17487161 millisecs
SelectionSort: 1000: 0.80411454 millisecs
ShellSort:    1000: 0.27911441 millisecs
```

n = 2000

```
n = 2000
Input Array: RANDOM
InsertionSort: 2000: 4.31375151 millisecs
SelectionSort: 2000: 3.4260224499999996 millisecs
ShellSort:    2000: 0.7271067499999999 millisecs
Input Array: ORDERED
InsertionSort: 2000: 0.01990242 millisecs
SelectionSort: 2000: 3.2033077800000003 millisecs
ShellSort:    2000: 0.03260751 millisecs
Input Array: PARTIAL
InsertionSort: 2000: 1.76304004 millisecs
SelectionSort: 2000: 3.35005066 millisecs
ShellSort:    2000: 0.36284320999999997 millisecs
Input Array: REVERSE
InsertionSort: 2000: 8.43414026 millisecs
SelectionSort: 2000: 3.1722367300000003 millisecs
ShellSort:    2000: 1.0830072 millisecs
```

n = 4000

```
n = 4000
Input Array: RANDOM
InsertionSort: 4000: 17.3643452 millisecs
SelectionSort: 4000: 13.874457540000002 millisecs
ShellSort:    4000: 2.56514357 millisecs
Input Array: ORDERED
InsertionSort: 4000: 0.03986725 millisecs
SelectionSort: 4000: 14.20706327 millisecs
ShellSort:    4000: 0.06282222 millisecs
Input Array: PARTIAL
InsertionSort: 4000: 7.434554520000001 millisecs
SelectionSort: 4000: 14.36428144 millisecs
ShellSort:    4000: 1.27312175 millisecs
Input Array: REVERSE
InsertionSort: 4000: 34.289141709999996 millisecs
SelectionSort: 4000: 12.538199839999999 millisecs
ShellSort:    4000: 4.067409420000001 millisecs
```

n = 8000

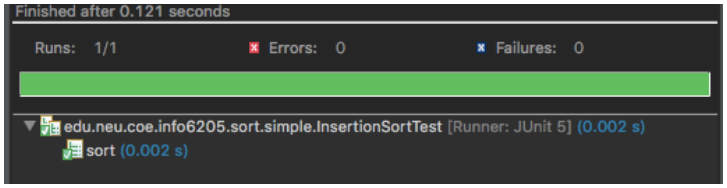
```
n = 8000
Input Array: RANDOM
InsertionSort: 8000: 69.07473159 millisecs
SelectionSort: 8000: 56.037128919999994 millisecs
ShellSort:    8000: 9.595129 millisecs
Input Array: ORDERED
InsertionSort: 8000: 0.44645005 millisecs
SelectionSort: 8000: 62.227222080000004 millisecs
ShellSort:    8000: 0.10659711 millisecs
Input Array: PARTIAL
InsertionSort: 8000: 32.74812182 millisecs
SelectionSort: 8000: 62.3886825 millisecs
ShellSort:    8000: 4.60995649 millisecs
Input Array: REVERSE
InsertionSort: 8000: 134.92300063000002 millisecs
SelectionSort: 8000: 49.88964815 millisecs
ShellSort:    8000: 16.13188367 millisecs
```

n = 16000

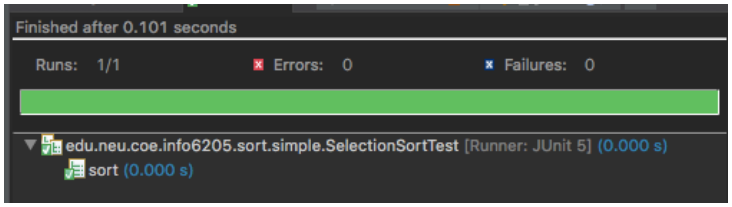
```
n = 16000
Input Array: RANDOM
InsertionSort: 16000: 294.08240127 millisecs
SelectionSort: 16000: 239.77529621000002 millisecs
ShellSort:    16000: 36.18850007 millisecs
Input Array: ORDERED
InsertionSort: 16000: 0.28172743 millisecs
SelectionSort: 16000: 202.28267 millisecs
ShellSort:    16000: 0.21196311 millisecs
Input Array: PARTIAL
InsertionSort: 16000: 140.92125292 millisecs
SelectionSort: 16000: 283.334996 millisecs
ShellSort:    16000: 18.25732224 millisecs
Input Array: REVERSE
InsertionSort: 16000: 771.4098780600001 millisecs
SelectionSort: 16000: 361.98357174999995 millisecs
ShellSort:    16000: 72.67132778 millisecs
```

Test Cases:

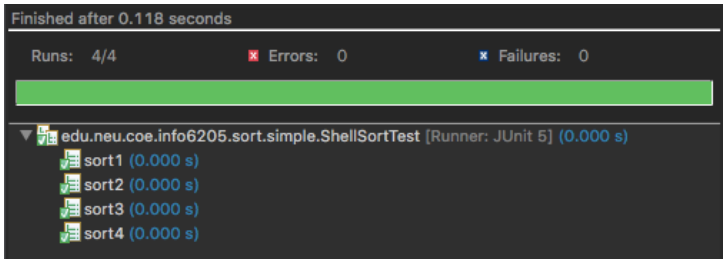
InsertionSortTest.java



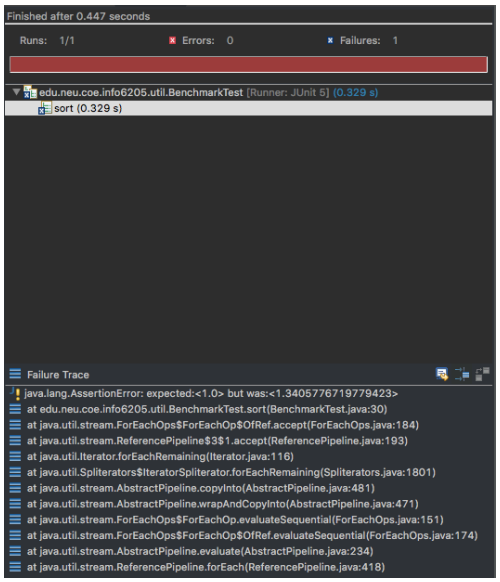
SelectionSortTest.java



ShellSortTest.java



BenchmarkTest.java



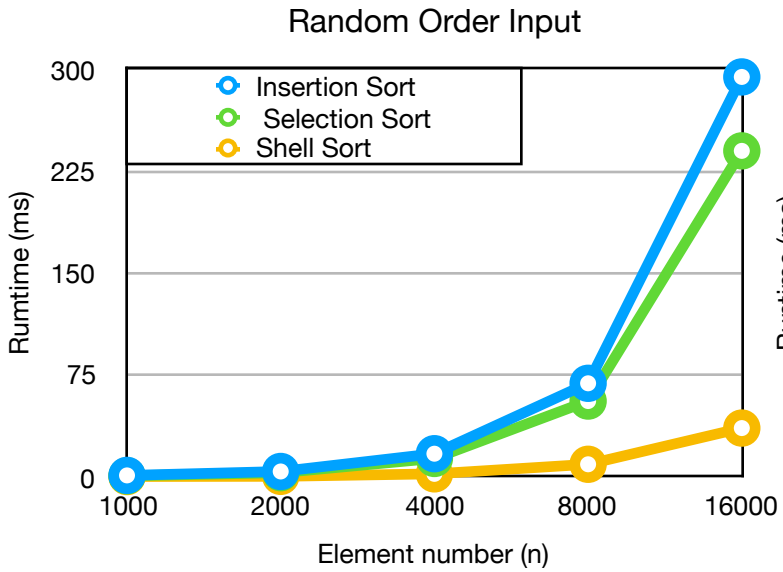
The BenchmarkTest failed, the result of mine is **1.340577**.

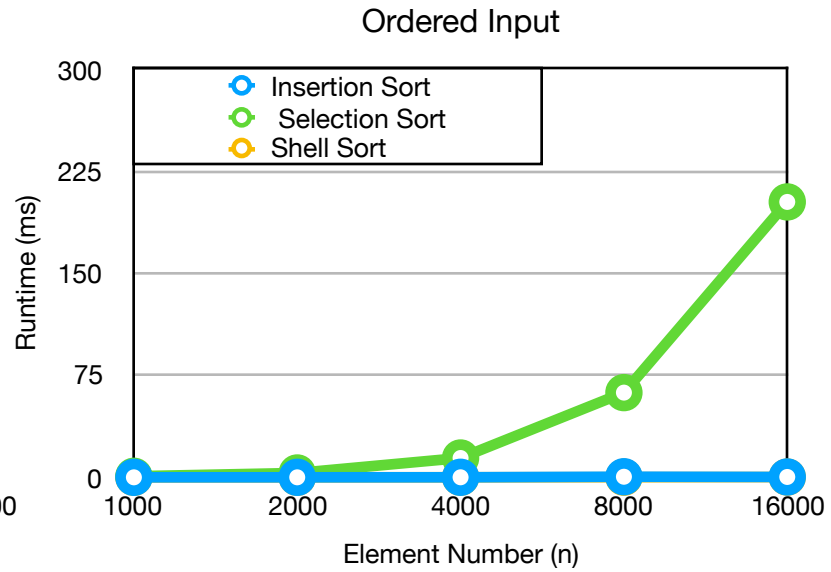
Conclusion:

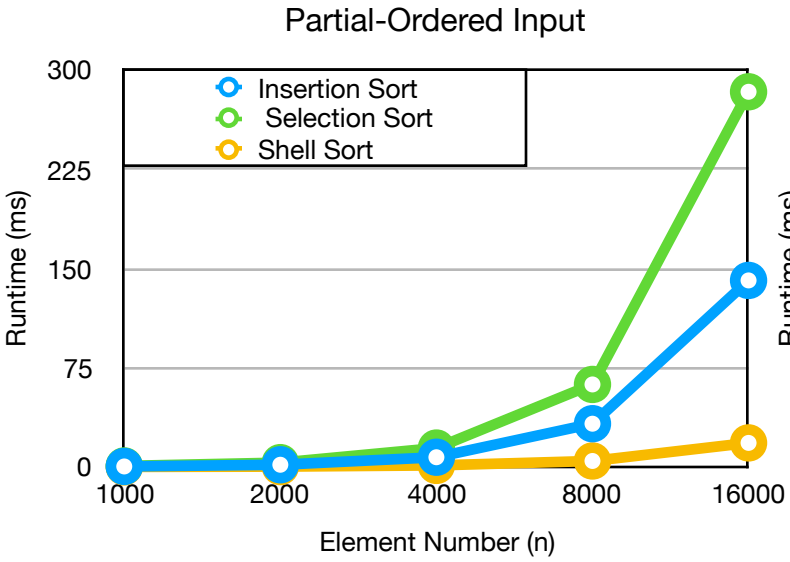
In the experiment section, shell sort is far **faster** than insertion sort and selection sort in any scenario.

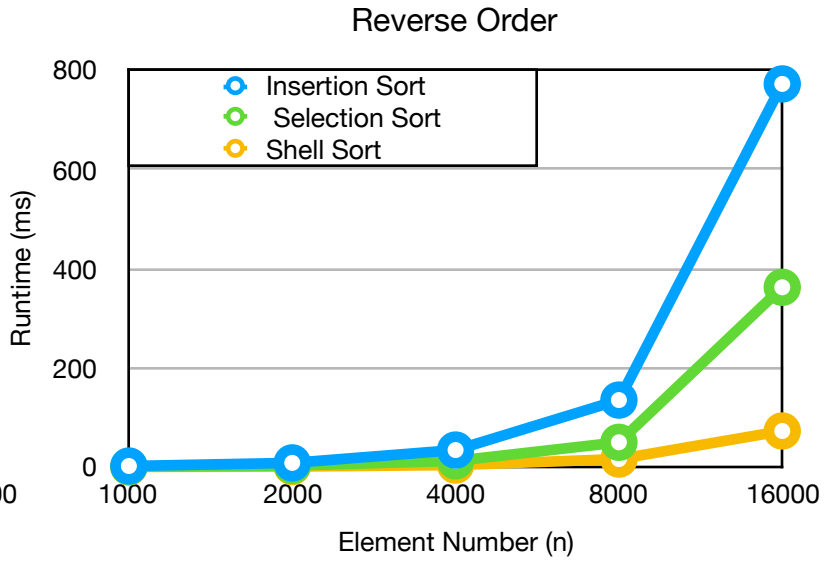
For random input array, the runtime of insertion sort and selection sort are approximately the **same**, but as the n increasing, the difference seems become significant.

Worthy to mention, when the input array is in ordered, **the runtime of selection sort is similar to random one.** It indicates that whether the input array is in order, selection sort still have to iterate the whole array to find the minimum element.









The runtime of each case shows above. As the random order input and partial order input array, insertion sort have the **same** growth.

In the partial-order case, insertion sort perform **faster** than selection sort.

In ordered case, the runtime of insertion sort becomes **linear time**.

In the reverse order case, the runtime of insertion sort seems grow **faster** than selection sort.