

# INFO6205 Assignment 3 (Benchmark)

NAME: Bohan Feng

NUID: 001564249

Repository: <https://github.com/fengb3/INFO6205>

## Part 1 Timer & Benchmark

### Unit Test Screenshot

Timer:

```

125
126 @Test // Slow
127 public void testRepeat3() {
128     final Timer timer = new Timer();
129     final int zzz = 20;
130     final double mean = timer.repeat(n: 10, () → zzz, t → {
131         GoToSleep(t, which: 0);
132         return null;
133     }, t → {
134         GoToSleep(t, which: -1);
135         System.out.println("post");
136         return t;
137     }, t → GoToSleep(mSecs: 10, which: 1));
138     assertEquals(expected: 10, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
139     assertEquals(zzz, mean, delta: 11.5);
140     assertEquals(expected: 10, run);
141     assertEquals(expected: 10, pre);
142     assertEquals(expected: 10, post);
143 }
144

```

```

145 @Test // Slow
146 public void testRepeat4() {
147     final Timer timer = new Timer();
148     final int zzz = 20;
149     final double mean = timer.repeat(n: 10,
150         () → zzz, // supplier
151         t → { // function
152             result = t;
153             GoToSleep(mSecs: 10, which: 0);
154             return null;
155         }, t → { // pre-function
156             GoToSleep(mSecs: 10, which: -1);
157             return 2*t;
158         }, t → GoToSleep(mSecs: 10, which: 1) // post-function
159     );
160     assertEquals(expected: 10, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
161     assertEquals(zzz, actual: 20, delta: 6);
162     assertEquals(expected: 10, run);
163     assertEquals(expected: 10, pre);
164     assertEquals(expected: 10, post);
165     // This test is designed to ensure that the preFunction is properly implemented in repeat.
166     assertEquals(expected: 40, result);
167 }
168

```

```

169 int pre = 0;
170 int run = 0;
171 int post = 0;
172 int result = 0;
173

```

```

174 private void GoToSleep(long mSecs, int which) {
175     try {
176         Thread.sleep(mSecs);
177         if (which == 0) run++;
178         else if (which > 0) post++;
179         else pre++;
180     } catch (InterruptedException e) {
181         e.printStackTrace();
182     }
183 }
184

```



17 xiaohuanlin

18 @Test

```
19 public void testStop() {
20     final Timer timer = new Timer();
21     GoToSleep(TENTH, which: 0);
22     final double time = timer.stop();
23     assertEquals(TENTH_DOUBLE, time, delta: 10);
24     assertEquals(expected: 1, run);
25     assertEquals(expected: 1, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
26 }
```

27 xiaohuanlin

28 @Test

```
29 public void testPauseAndLap() {
30     final Timer timer = new Timer();
31     final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
32     GoToSleep(TENTH, which: 0);
33     timer.pauseAndLap();
34     final Long ticks = (Long) privateMethodTester.invokePrivate(name: "getTicks");
35     assertEquals(TENTH_DOUBLE, actual: ticks / 1e6, delta: 12);
36     assertFalse((Boolean) privateMethodTester.invokePrivate(name: "isRunning"));
37     assertEquals(expected: 1, privateMethodTester.invokePrivate(name: "getLaps"));
38 }
```

39 xiaohuanlin

40 @Test

```
41 public void testPauseAndLapResume0() {
42     final Timer timer = new Timer();
43     final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
44     GoToSleep(TENTH, which: 0);
45     timer.pauseAndLap();
46     timer.resume();
47     assertTrue((Boolean) privateMethodTester.invokePrivate(name: "isRunning"));
48     assertEquals(expected: 1, privateMethodTester.invokePrivate(name: "getLaps"));
49 }
```

50 xiaohuanlin

51 @Test

```
52 public void testPauseAndLapResume1() {
53     final Timer timer = new Timer();
54     GoToSleep(TENTH, which: 0);
55     timer.pauseAndLap();
56     GoToSleep(TENTH, which: 0);
57     timer.resume();
58     GoToSleep(TENTH, which: 0);
59     final double time = timer.stop();
60     assertEquals(TENTH_DOUBLE, time, delta: 10.0);
61     assertEquals(expected: 3, run);
62 }
```

63 xiaohuanlin

64 @Test

```
65 public void testLap() {
66     final Timer timer = new Timer();
67     GoToSleep(TENTH, which: 0);
68     timer.lap();
69     GoToSleep(TENTH, which: 0);
70     final double time = timer.stop();
71     assertEquals(TENTH_DOUBLE, time, delta: 10.0);
72     assertEquals(expected: 2, run);
73 }
```

74 xiaohuanlin

75 @Test

```
76 public void testPause() {
77     final Timer timer = new Timer();
78     GoToSleep(TENTH, which: 0);
```



```

75  @Test
76  public void testPause() {
77      final Timer timer = new Timer();
78      GoToSleep(TENTH, which: 0);
79      timer.pause();
80      GoToSleep(TENTH, which: 0);
81      timer.resume();
82      final double time = timer.stop();
83      assertEquals(TENTH_DOUBLE, time, delta: 10.0);
84      assertEquals(expected: 2, run);
85  }
86
87  @Test
88  public void testMillisecs() {
89      final Timer timer = new Timer();
90      GoToSleep(TENTH, which: 0);
91      timer.stop();
92      final double time = timer.millisecs();
93      assertEquals(TENTH_DOUBLE, time, delta: 10.0);
94      assertEquals(expected: 1, run);
95  }
96
97  @Test
98  public void testRepeat1() {
99      final Timer timer = new Timer();
100     final double mean = timer.repeat(n: 10, () → {
101         GoToSleep(HUNDREDTH, which: 0);
102         return null;
103     });
104     assertEquals(expected: 10, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
105     assertEquals(expected: TENTH_DOUBLE / 10, mean, delta: 6);
106     assertEquals(expected: 10, run);
107     assertEquals(expected: 0, pre);
108     assertEquals(expected: 0, post);
109 }
110
111 @Test
112 public void testRepeat2() {
113     final Timer timer = new Timer();
114     final int zzz = 20;
115     final double mean = timer.repeat(n: 10, () → zzz, t → {
116         GoToSleep(t, which: 0);
117         return null;
118     });
119     assertEquals(expected: 10, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
120     assertEquals(zzz, mean, delta: 11);
121     assertEquals(expected: 10, run);
122     assertEquals(expected: 0, pre);
123     assertEquals(expected: 0, post);
124 }
125
126 @Test // Slow
127 public void testRepeat3() {
128     final Timer timer = new Timer();
129     final int zzz = 20;
130     final double mean = timer.repeat(n: 10, () → zzz, t → {
131         GoToSleep(t, which: 0);
132         return null;
133     }, t → {
134         GoToSleep(t, which: -1);
135         System.out.println("post");
136         return t;
137     }, t → GoToSleep(mSecs: 10, which: 1));

```

Run: TimerTest X

✓ Tests passed: 11 of 11 tests – 3 sec 397 ms

Test Name	Duration
✓ testPauseAndLapResume0	608 ms
✓ testPauseAndLapResume1	324 ms
✓ testLap	217 ms
✓ testPause	213 ms
✓ testStop	104 ms
✓ testMillisecs	108 ms
✓ testRepeat1	154 ms
✓ testRepeat2	309 ms
✓ testRepeat3	773 ms
✓ testRepeat4	477 ms
✓ testPauseAndLap	110 ms

C:\Users\冯博藻\.jdk\openjdk-18.0.2.1\bin\java.exe ...

Process finished with exit code 0

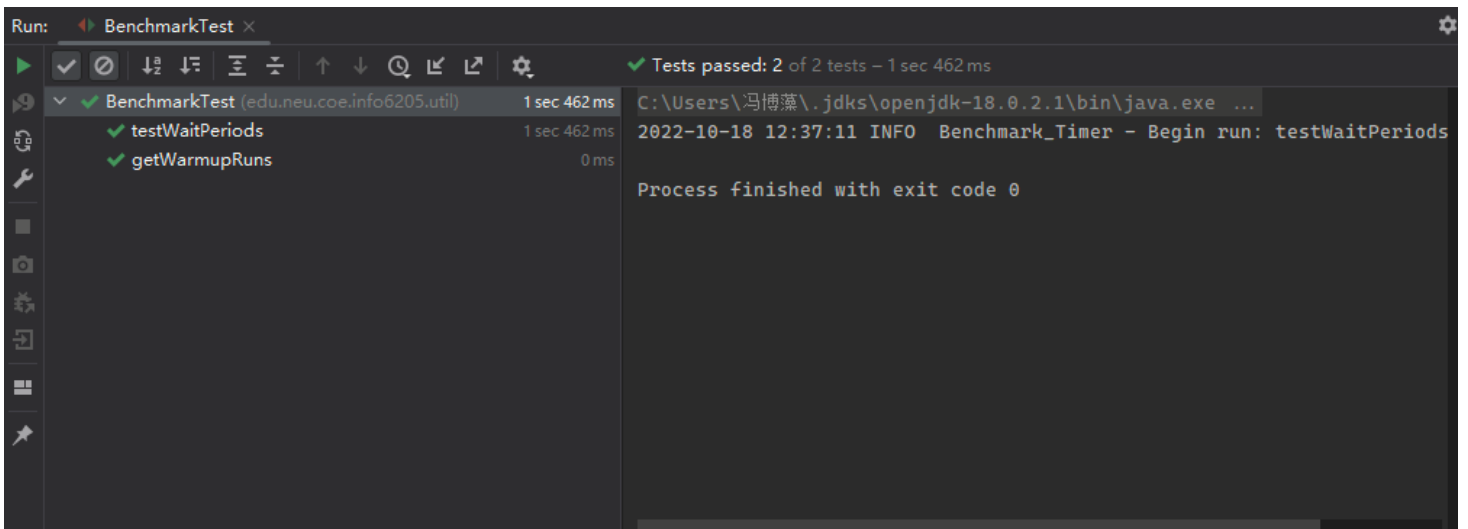
Benchmark:

```

1  / ... /
4
5  package edu.neu.coe.info6205.util;
6
7  import ...
10
11  /ALL/
12  public class BenchmarkTest {
13
14      2 usages
15      int pre = 0;
16      2 usages
17      int run = 0;
18      2 usages
19      int post = 0;
20
21      xiaohuanlin
22      @Test // Slow
23      public void testWaitPeriods() throws Exception {
24          int nRuns = 2;
25          int warmups = 2;
26          Benchmark<Boolean> bm = new Benchmark_Timer<>{
27              description: "testWaitPeriods", b → {
28                  GoToSleep( mSecs: 100L, which: -1);
29                  return null;
30              },
31              b → {
32                  GoToSleep( mSecs: 200L, which: 0);
33              },
34              b → {
35                  GoToSleep( mSecs: 50L, which: 1);
36              }
37          };
38          double x = bm.run( true, nRuns);
39          assertEquals(nRuns, post);
40          assertEquals( expected: nRuns + warmups, run);
41          assertEquals( expected: nRuns + warmups, pre);
42          assertEquals( expected: 200, x, delta: 10);
43      }
44
45      3 usages xiaohuanlin
46      private void GoToSleep(long mSecs, int which) {
47          try {
48              Thread.sleep(mSecs);
49              if (which == 0) run++;
50              else if (which > 0) post++;
51              else pre++;
52          } catch (InterruptedException e) {
53              e.printStackTrace();
54          }
55      }
56
57      xiaohuanlin
58      @Test
59      public void getWarmupRuns() {
60          assertEquals( expected: 2, Benchmark_Timer.getWarmupRuns( m: 0));
61          assertEquals( expected: 2, Benchmark_Timer.getWarmupRuns( m: 20));
62          assertEquals( expected: 3, Benchmark_Timer.getWarmupRuns( m: 30));
63          assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 100));
64          assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 1000));
65      }
66  }

```





## Part 2 Insertion Sort

Unit Test Screenshot:

```

21  @Test
22  public void sort0() throws Exception {
23      final List<Integer> list = new ArrayList<>();
24      list.add(1);
25      list.add(2);
26      list.add(3);
27      list.add(4);
28      Integer[] xs = list.toArray(new Integer[0]);
29      final Config config = ConfigTest.setupConfig( instrumenting: "true", seed: "0", inversions: "1", cutoff: "", interimInversion: "0.5");
30      Helper<Integer> helper = HelperFactory.create( description: "InsertionSort", list.size(), config);
31      helper.init(list.size());
32      final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
33      final StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");
34      SortWithHelper<Integer> sorter = new InsertionSort<>>(helper);
35      sorter.preProcess(xs);
36      Integer[] ys = sorter.sort(xs);
37      assertTrue(helper.sorted(ys));
38      sorter.postProcess(ys);
39      final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
40      assertEquals( expected: list.size() - 1, compares);
41      final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
42      assertEquals( expected: 0L, inversions);
43      final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
44      assertEquals(inversions, fixes);
45  }
46
47  @Test
48  public void sort1() throws Exception {
49      final List<Integer> list = new ArrayList<>();
50      list.add(3);
51      list.add(4);
52      list.add(2);
53      list.add(1);
54      Integer[] xs = list.toArray(new Integer[0]);
55      BaseHelper<Integer> helper = new BaseHelper<>( description: "InsertionSort", xs.length, Config.load(InsertionSortTest.class));
56      GenericSort<Integer> sorter = new InsertionSort<>>(helper);
57      Integer[] ys = sorter.sort(xs);
58      assertTrue(helper.sorted(ys));
59      System.out.println(sorter.toString());
60  }
61
62  @Test
63  public void testMutatingInsertionSort() throws IOException {
64      final List<Integer> list = new ArrayList<>();
65      list.add(3);
66      list.add(4);
67      list.add(2);
68      list.add(1);
69      Integer[] xs = list.toArray(new Integer[0]);
70      BaseHelper<Integer> helper = new BaseHelper<>( description: "InsertionSort", xs.length, Config.load(InsertionSortTest.class));
71      GenericSort<Integer> sorter = new InsertionSort<>>(helper);
72      sorter.mutatingSort(xs);
73      assertTrue(helper.sorted(xs));
74  }
75
76  @Test
77  public void testStaticInsertionSort() throws IOException {
78      final List<Integer> list = new ArrayList<>();
79      list.add(3);
80      list.add(4);
81      list.add(2);
82      list.add(1);
83      Integer[] xs = list.toArray(new Integer[0]);
84  }

```

Run: InsertionSortTest

Tests passed: 6 of 6 tests - 150 ms

✓ InsertionSortTest (edu.neu.coe.info6205.sort.elementa 150 ms

✓ testMutatingInsertionSort 107 ms

✓ sort0 33 ms

✓ sort1 1 ms

✓ sort2 5 ms

✓ sort3 2 ms

✓ testStaticInsertionSort 2 ms

C:\Users\冯博瀚\.jdk\openjdk-18.0.2.1\bin\java.exe ...

2022-10-18 11:46:14 DEBUG Config - Config.get(helper, instrument) = tr

2022-10-18 11:46:14 DEBUG Config - Config.get(helper, seed) = 0

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, copies) =

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, swaps) =

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, compares) =

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, inversion

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, fixes) =

2022-10-18 11:46:14 DEBUG Config - Config.get(instrumenting, hits) = t

2022-10-18 11:46:14 DEBUG Config - Config.get(helper, cutoff) =

Helper for InsertionSort with 4 elements

StatPack {hits: 9,880, normalized=21.454; copies: 0, normalized=0.000;

StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000

Process finished with exit code 0

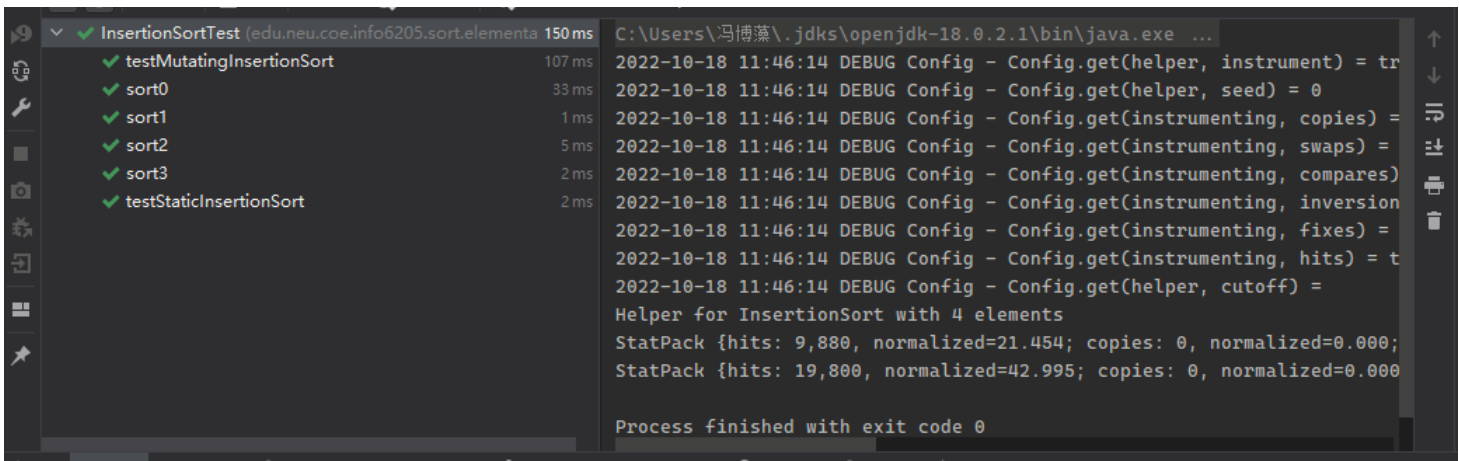
```

82     list.add(1);
83     Integer[] xs = list.toArray(new Integer[0]);
84     InsertionSort.sort(xs);
85     assertTrue( condition: xs[0] < xs[1] && xs[1] < xs[2] && xs[2] < xs[3]);
86 }
87
88  xiaohuanlin
89 @Test
90 public void sort2() throws Exception {
91     final Config config = ConfigTest.setupConfig( instrumenting: "true", seed: "0", inversions: "1", cutoff: "", interimInversion:
92     int n = 100;
93     Helper<Integer> helper = HelperFactory.create( description: "InsertionSort", n, config);
94     helper.init(n);
95     final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
96     final StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");
97     Integer[] xs = helper.random(Integer.class, r → r.nextInt( bound: 1000));
98     SortWithHelper<Integer> sorter = new InsertionSort<Integer>(helper);
99     sorter.preProcess(xs);
100    Integer[] ys = sorter.sort(xs);
101    assertTrue(helper.sorted(ys));
102    sorter.postProcess(ys);
103    final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
104    // NOTE: these are supposed to match within about 12%.
105    // Since we set a specific seed, this should always succeed.
106    // If we use true random seed and this test fails, just increase the delta a little.
107    assertEquals( expected: 1.0, actual: 4.0 * compares / n / (n - 1), delta: 0.12);
108    final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
109    final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
110    System.out.println(statPack);
111    assertEquals(inversions, fixes);
112 }
113
114  xiaohuanlin
115 @Test
116 public void sort3() throws Exception {
117     final Config config = ConfigTest.setupConfig( instrumenting: "true", seed: "0", inversions: "1", cutoff: "", interimInversion:
118     int n = 100;
119     Helper<Integer> helper = HelperFactory.create( description: "InsertionSort", n, config);
120     helper.init(n);
121     final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
122     final StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");
123     Integer[] xs = new Integer[n];
124     for (int i = 0; i < n; i++) xs[i] = n - i;
125     SortWithHelper<Integer> sorter = new InsertionSort<>(helper);
126     sorter.preProcess(xs);
127     Integer[] ys = sorter.sort(xs);
128     assertTrue(helper.sorted(ys));
129     sorter.postProcess(ys);
130     final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
131     // NOTE: these are supposed to match within about 12%.
132     // Since we set a specific seed, this should always succeed.
133     // If we use true random seed and this test fails, just increase the delta a little.
134     assertEquals( expected: 4950, compares);
135     final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
136     final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
137     System.out.println(statPack);
138     assertEquals(inversions, fixes);
139 }
140
141     final static LazyLogger logger = new LazyLogger(InsertionSort.class);
142 }

```

Run: InsertionSortTest ×

Tests passed: 6 of 6 tests – 150 ms



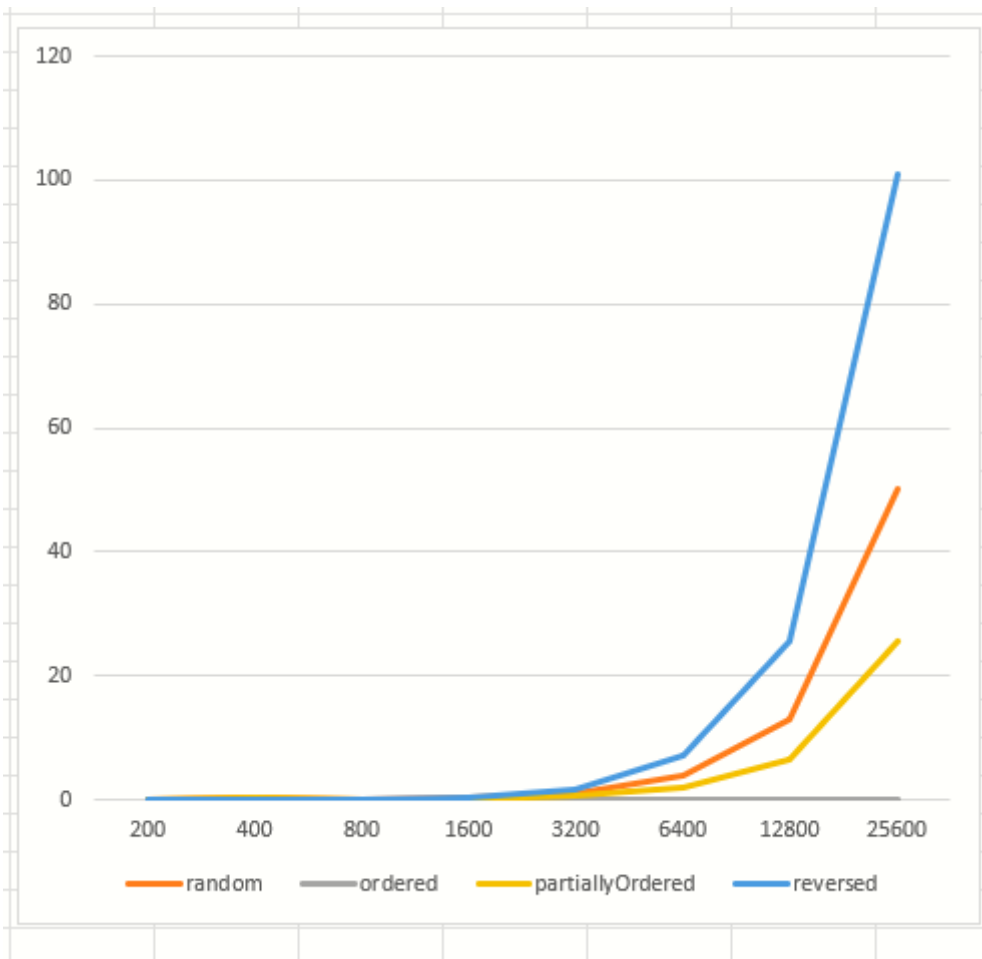
## Part 3 Measure Running Times

### Measured Data

Table:

n	random	ordered	partiallyOrdered	reversed
200	0.0374	0.00516	0.01614	0.04288
400	0.32443	0.00359	0.19526	0.05365
800	0.10849	0.0065	0.03351	0.11643
1600	0.25289	0.00811	0.12406	0.45562
3200	0.93144	0.01623	0.56324	1.79383
6400	3.84342	0.03391	1.83933	7.2329
12800	12.86309	0.05785	6.54417	25.50461
25600	50.28089	0.0905	25.56846	100.8463

Graph:



## Obervation

I used doubling method to test the time usage of InsertionSort. For each different array length ( $n$ ), I tried four array sorting situations (random, ordered, inverted, and partially ordered). Based on the output data and charts, we can briefly know that the speed of sorting ordered array is the fastest, sorting random and partially ordered array is slower, and sorting for reversed is the slowest.

For best situation (ordered), InsertionSort does not require a swap operation, it needs  $(n - 1)$  times comparison. For worst situation(reversed), InsertionSort needs  $n * (n - 1) / 2$  times comparison, because when we insert  $n$ th element, we need to compare previous  $(n - 1)$  elements. InsertionSort's swap times is number of comparison operations minus  $(n - 1)$ . On the average, the time complexity of the InsertionSort is  $O(n^2)$ .