

Program Structures and Algorithms  
Spring 2023(SEC –3)

NAME: Akshay Parab  
NUID: 002766150

**Task:**

Run benchmark to measure the running time of Insertion Sort using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. By using the doubling method for choosing input size and test for at least five values of input size, draw any conclusions from your observations regarding the order of growth of the Insertion Sort algorithm.

**Relationship Conclusion:**

Please find detailed observation based on the benchmark test on Insertion Sort:

1. **Best-Case Scenario:** The best-case scenario for the Insertion Sort algorithm is an ordered array, meaning the elements in the array are already sorted. In this scenario, the algorithm has a linear time complexity of  $O(n)$ . This is because each element only needs to be compared and inserted once, resulting in a running time proportional to the size of the input.
2. **Average-Case Scenario:** The average-case scenario for the Insertion Sort algorithm is a random array, meaning the elements in the array are not sorted. In this scenario, the algorithm has a quadratic time complexity of  $O(n^2)$ . This is because each element needs to be compared and inserted several times, resulting in a running time proportional to the square of the size of the input.
3. **Worst-Case Scenario:** The worst-case scenario for the Insertion Sort algorithm is a reverse-ordered array, meaning the elements in the array are sorted in the opposite direction. In this scenario, the algorithm also has a quadratic time complexity of  $O(n^2)$ . This is because each element needs to be compared and inserted several times, resulting in a running time proportional to the square of the size of the input.
4. **Partially Ordered Scenario:** The partially ordered scenario for the Insertion Sort algorithm refers to an array where some elements are sorted, but not all. In this scenario, the time complexity of the algorithm may be between  $O(n)$  and  $O(n^2)$ , depending on the degree of partial order. If the degree of partial order is closer to the ordered scenario, the time complexity will be closer to  $O(n)$ , and if the degree of partial order is closer to the reverse-ordered scenario, the time complexity will be closer to  $O(n^2)$ .

Overall, the Insertion Sort algorithm has a quadratic time complexity in most cases and is generally not efficient for large datasets.

**Evidence to support that conclusion:**

**Graphical Representation:**

*Input Type: Randomly Sorted*

Sr. No.	Input Size	Iterations	Average Time (ms)
1	1000	100	1.2529600099999998
2	2000	100	4.8126466599999995
3	4000	100	18.962787919999997
4	8000	100	74.54104998
5	16000	100	444.38567997999996

*Input Type: Partially Sorted*

Sr. No.	Input Size	Iterations	Average Time (ms)
1	1000	100	0.9302303999999999
2	2000	100	3.6490608599999996
3	4000	100	13.7623567
4	8000	100	57.44689715
5	16000	100	240.82581793999998

### Input Type: Ordered Input

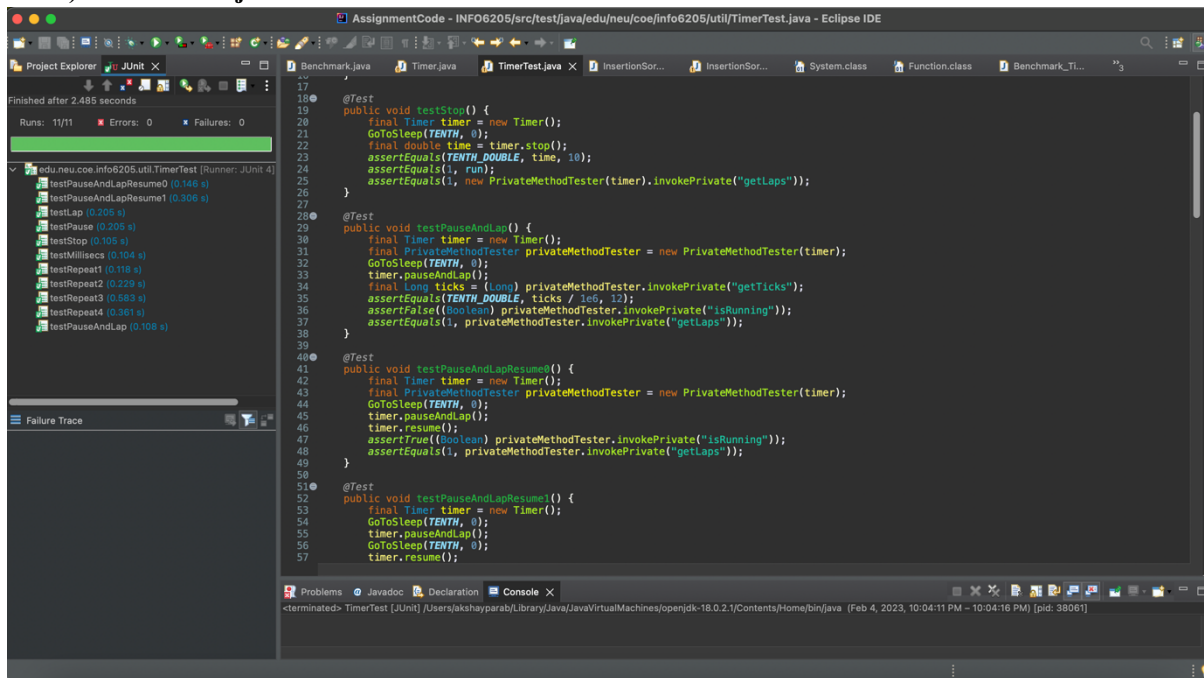
Sr. No.	Input Size	Iterations	Average Time (ms)
1	1000	100	0.00649289
2	2000	100	0.01214583
3	4000	100	0.025107499999999998
4	8000	100	0.051137489999999994
5	16000	100	0.10269331

### Input Type: Reverse Sorted

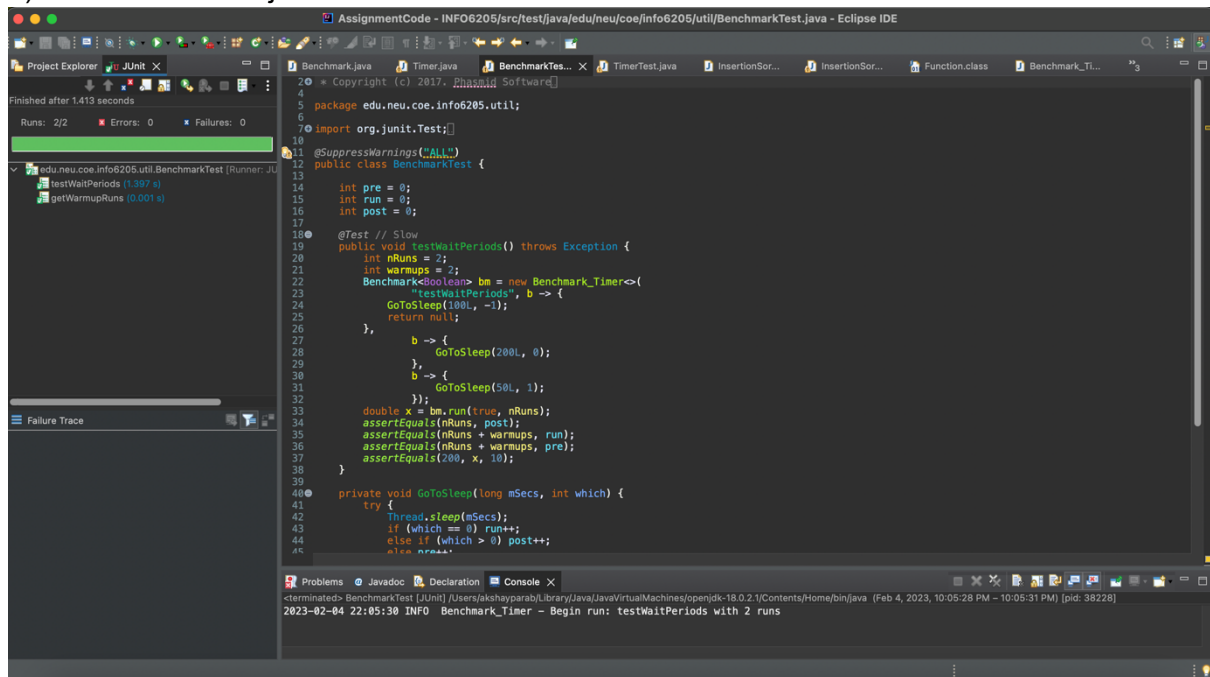
Sr. No.	Input Size	Iterations	Average Time (ms)
1	1000	100	2.39149791
2	2000	100	9.56261752
3	4000	100	36.42599541
4	8000	100	146.34479753
5	16000	100	662.29248377

## Unit Test Screenshots:

### 1) TimerTest.java



## 2) BenchmarkTest.java



The screenshot shows the Eclipse IDE with the file `BenchmarkTest.java` open. The code is a JUnit test class that benchmarks a `Benchmark_Timer` class. It includes a `@Test` method `testWaitPeriods` and a `private void goToSleep` method. The left sidebar shows the Project Explorer with the test results for `edu.neu.coe.info6205.util.BenchmarkTest`. The bottom console shows the output of the test run.

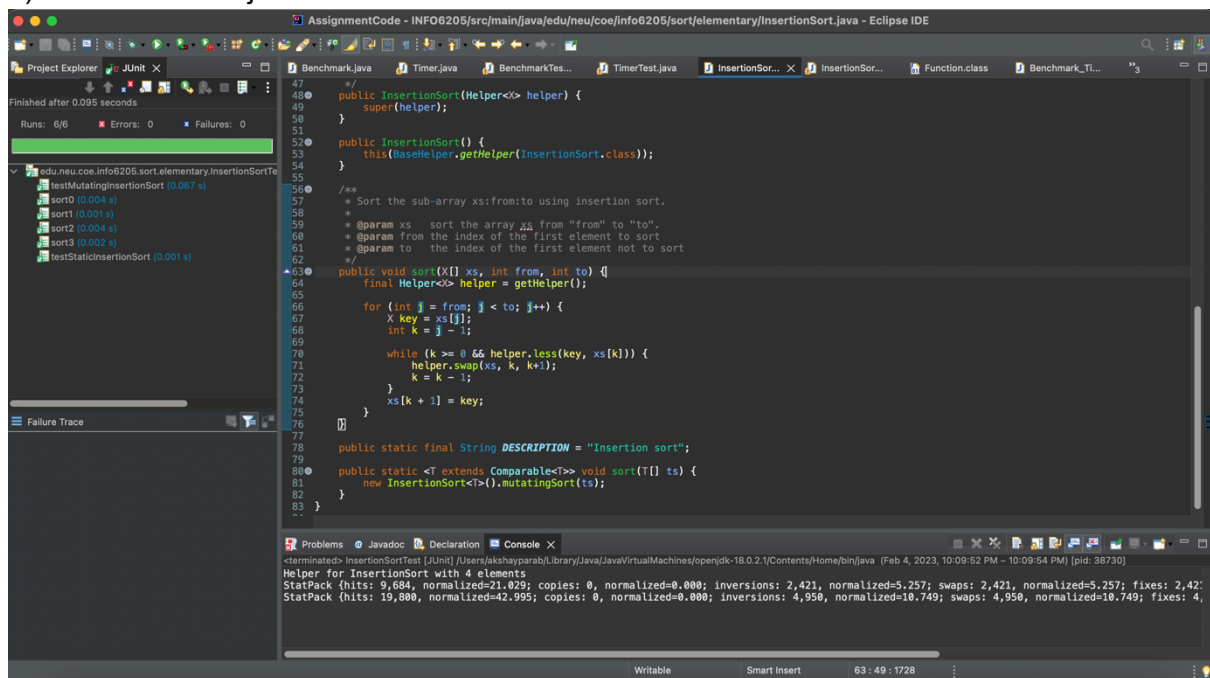
```
20 * Copyright (c) 2017. Phasmat Software
4
5 package edu.neu.coe.info6205.util;
6
7 import org.junit.Test;
8
9
10
11 @SuppressWarnings("ALL")
12 public class BenchmarkTest {
13
14     int pre = 0;
15     int run = 0;
16     int post = 0;
17
18     @Test // Slow
19     public void testWaitPeriods() throws Exception {
20         int nRuns = 2;
21         int warmups = 2;
22         Benchmark_Timer<> bm = new Benchmark_Timer<>("testWaitPeriods", b -> {
23             goToSleep(100L, -1);
24             return null;
25         },
26             b -> {
27                 goToSleep(200L, 0);
28             },
29             b -> {
30                 goToSleep(50L, 1);
31             });
32         double x = bm.run(true, nRuns);
33         assertEquals(nRuns, post);
34         assertEquals(nRuns + warmups, run);
35         assertEquals(nRuns + warmups, pre);
36         assertEquals(200, x, 10);
37     }
38
39     private void goToSleep(long mSecs, int which) {
40         try {
41             Thread.sleep(mSecs);
42             if (which == 0) run++;
43             else if (which > 0) post++;
44             // else pre++;
45         } catch (InterruptedException e) {}
46     }
47 }
```

Failure Trace

Problems Javadoc Declaration Console

terminated: BenchmarkTest [JUnit] /Users/akshayparab/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java (Feb 4, 2023, 10:05:28 PM ~ 10:05:31 PM) [pid: 38228]  
2023-02-04 22:05:30 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 2 runs

## 3) InsertSortTest.java



The screenshot shows the Eclipse IDE with the file `InsertSortTest.java` open. The code is a JUnit test class that benchmarks an `InsertSort` class. It includes a `@Test` method `testStaticInsertSort` and a `public void sort` method. The left sidebar shows the Project Explorer with the test results for `edu.neu.coe.info6205.sort.elementary.InsertSortTest`. The bottom console shows the output of the test run.

```
47
48 public InsertSort(Helper<> helper) {
49     super(helper);
50 }
51
52 public InsertSort() {
53     this(helper.getHelper(InsertSort.class));
54 }
55
56 /**
57  * Sort the sub-array xs:from:to using insertion sort.
58  * @param xs sort the array xs from "from" to "to".
59  * @param from the index of the first element to sort
60  * @param to the index of the first element not to sort
61  */
62
63 public void sort(X[] xs, int from, int to) {
64     final Helper<> helper = getHelper();
65
66     for (int j = from; j < to; j++) {
67         X key = xs[j];
68         int k = j - 1;
69
70         while (k >= 0 && helper.less(key, xs[k])) {
71             helper.swap(xs, k, k+1);
72             k = k - 1;
73         }
74         xs[k + 1] = key;
75     }
76 }
77
78 public static final String DESCRIPTION = "Insertion sort";
79
80 public static <T extends Comparable<T>> void sort(T[] ts) {
81     new InsertSort<>().mutatingSort(ts);
82 }
83
84 }
```

Failure Trace

Problems Javadoc Declaration Console

terminated: InsertSortTest [JUnit] /Users/akshayparab/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java (Feb 4, 2023, 10:09:52 PM ~ 10:09:54 PM) [pid: 38730]  
Helper for InsertSort with 4 elements  
StatPack {hits: 9,684, normalized=21.029; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normalized=5.257; fixes: 2,42; StatPack {hits: 19,880, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=10.749; fixes: 4;