Program Structures and Algorithms
Fall 2023

NAME: Shangqing Hu
NUID: 001374342

Task:

(Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Please see the skeleton class that I created in the repository. *Timer* is invoked from a class called *Benchmark_Timer* which implements the *Benchmark* interface. The APIs of these class are as follows:

(Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort*. If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). The easiest is to use the *helper.swapStableConditional* method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in *InsertionSortTest*.

(Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to be sorted are of type *Integer*. Use the doubling method for choosing *n* and test for at least five values of *n*. Draw any conclusions from your observations regarding the order of growth.

Relationship Conclusion:
1. In the case of an array containing random elements, insertion sort exhibits an average and worst-case time complexity of O(n^2). This is due to the potential need for multiple shifts of each element before it finds its correct position.

2. When dealing with an array sorted in reverse order, the time complexity of insertion sort remains O(n^2). In this scenario, each element must be shifted to the beginning of the array during every iteration, leading to the worst-case performance for insertion sort.

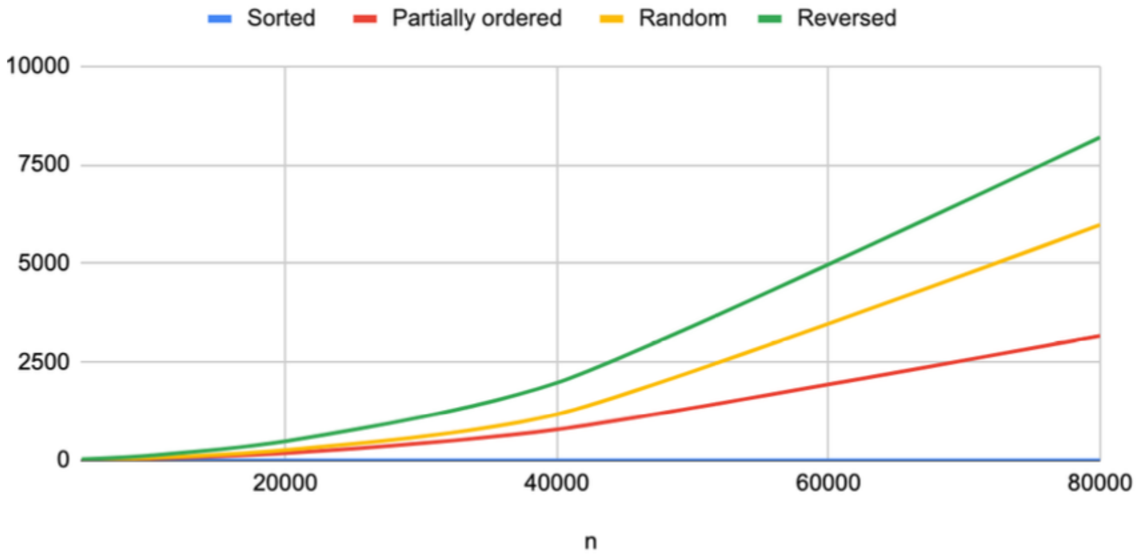
3. If the array is already sorted, insertion sort demonstrates its best-case time complexity of O(n). This favorable scenario occurs because elements only require checking without any shifting, resulting in efficient sorting.

4. For a partially sorted array, the time complexity falls somewhere between O(n) and O(n^2), dependent on the extent of order within the array. The greater the level of pre-sortedness, the closer the time complexity approaches O(n).

Evidence to support that conclusion:

n	Sorted	Partially ordered	Random	Reversed
5000	0	11.7	15.15	30.2
10000	0	46.3	61.9	122.7
20000	0	181.45	260.6	484.4
40000	0.05	784	1172.9	1974.95
80000	0.1	3165.75	5978	8196.95

Graphical Representation:



Observation:

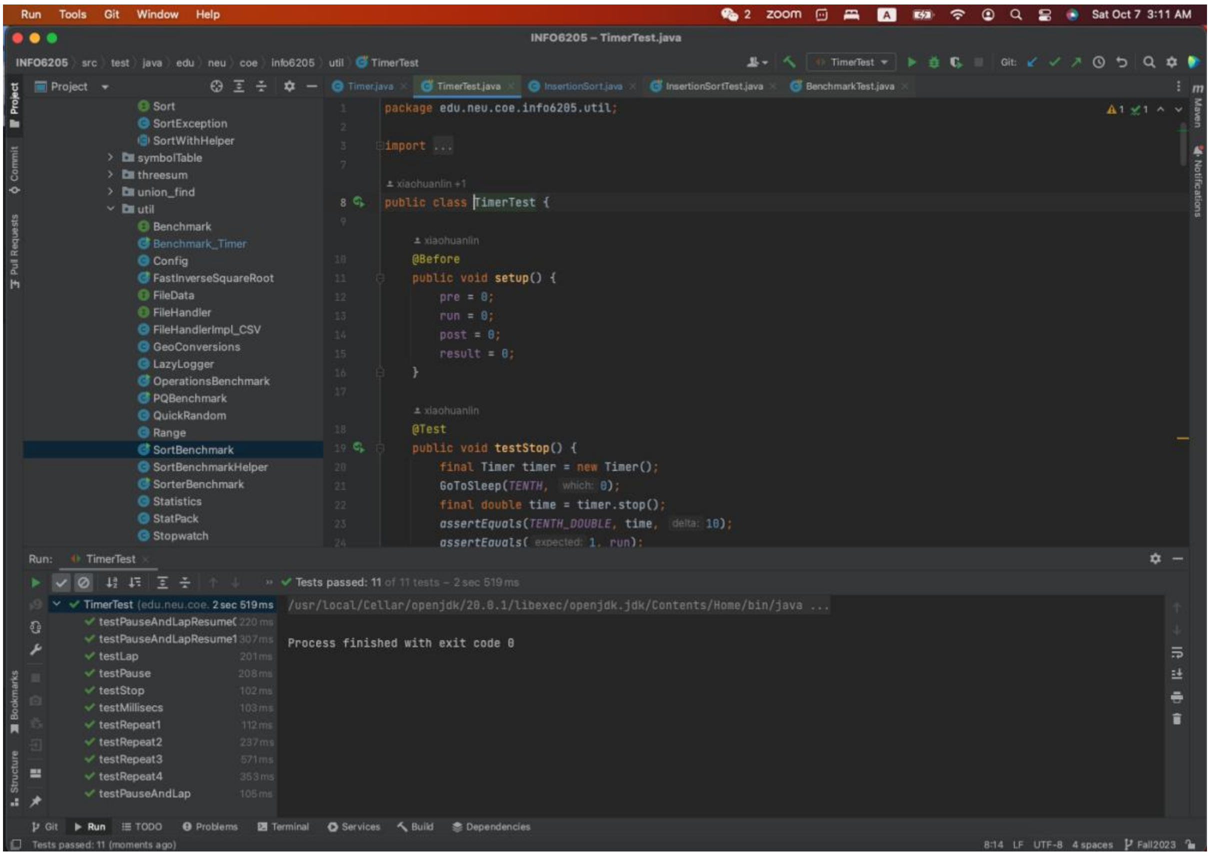
1. In the case of an array containing random elements, the graph's growth rate follows O(n^2), signifying a rapid increase as the array size expands. This growth pattern forms a distinct parabolic shape.

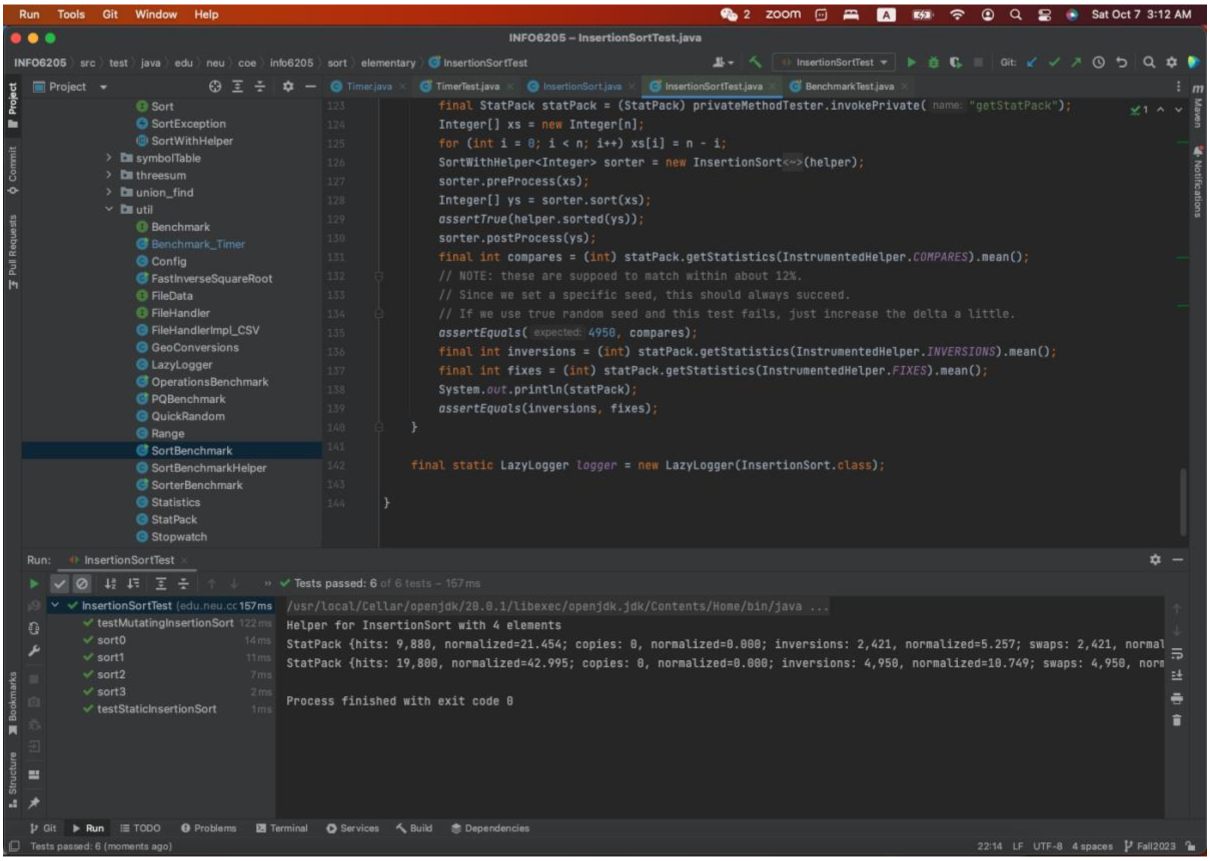
2. When dealing with an array sorted in reverse order, the graph exhibits the same O(n^2) growth rate as seen with random elements. Consequently, the graph also assumes a parabolic shape.

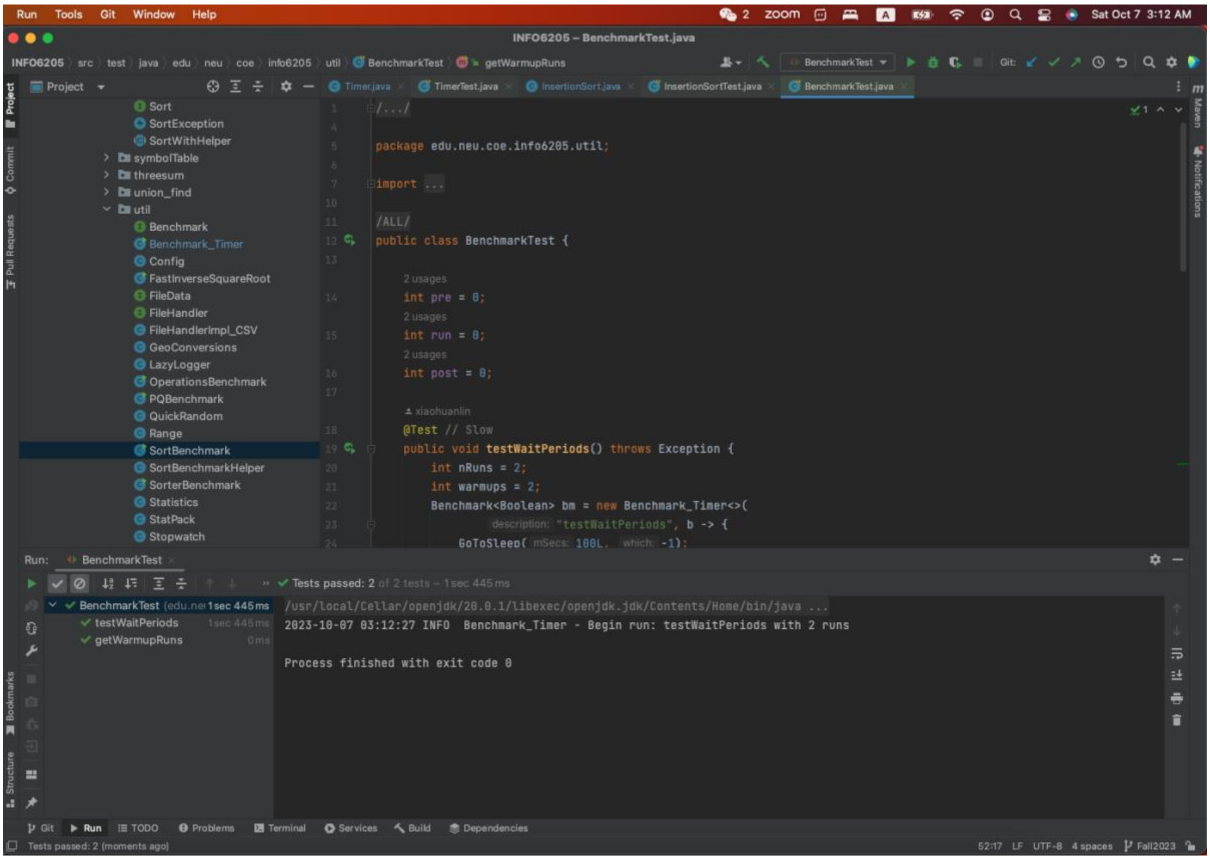
3. If the array is already sorted, the graph's growth rate adheres to O(n), resulting in a linear progression as the array size enlarges. This growth pattern takes the form of a straight line.

4. For a partially sorted array, the graph's growth rate falls between O(n) and O(n^2). When the array is mostly sorted, the graph approaches O(n), and as sorting decreases, it approaches O(n^2). This produces a curved shape, intermediary between a straight line and a parabolic curve.

Screenshots of run and/or Unit Test:






INFO6205 – BenchmarkTest.java
package edu.neu.coe.info6205.util;
import ...
/ALL/
public class BenchmarkTest {
int pre = 0;
int run = 0;
int post = 0;
@Test // Slow
public void testWaitPeriods() throws Exception {
int nRuns = 2;
int warmups = 2;
Benchmark<Boolean> bm = new Benchmark_Timer<>(
description: "testWaitPeriods", b -> {
Tests passed: 2 of 2 tests – 1 sec 445 ms
BenchmarkTest
testWaitPeriods
getWarmupRuns
/usr/local/Cellar/openjdk/20.0.1/libexec/openjdk.jdk/Contents/Home/bin/java ...
2023-10-07 03:12:27 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
Process finished with exit code 0
Tests passed: 2 (moments ago)
Sat Oct 7 3:12 AM