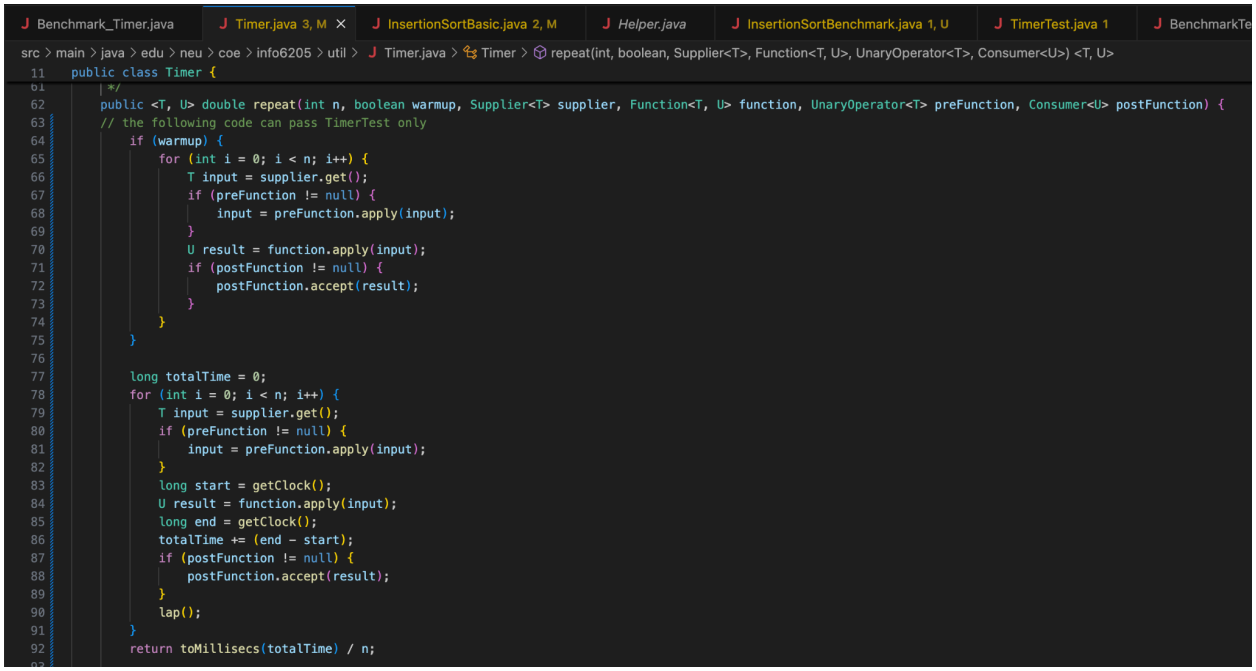
Program Structures and Algorithms

Fall 2024

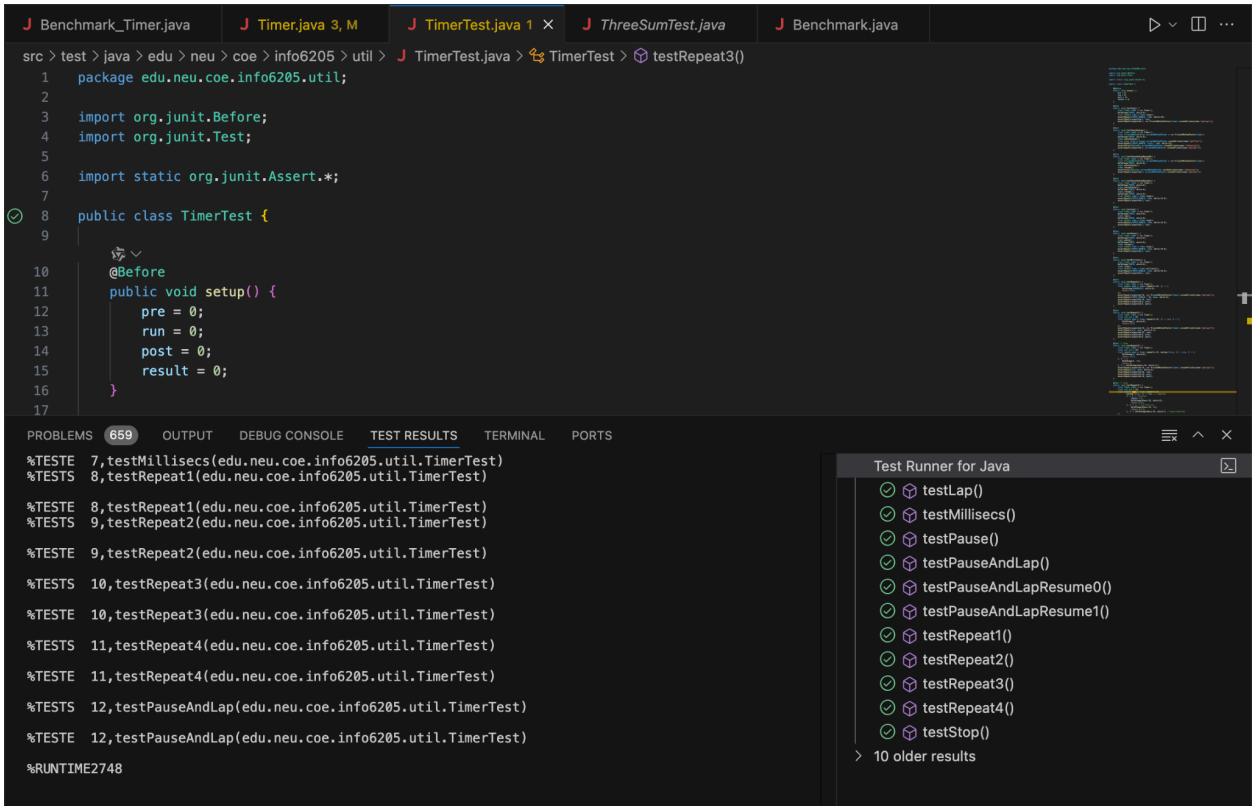
NAME: Xinyi Xu

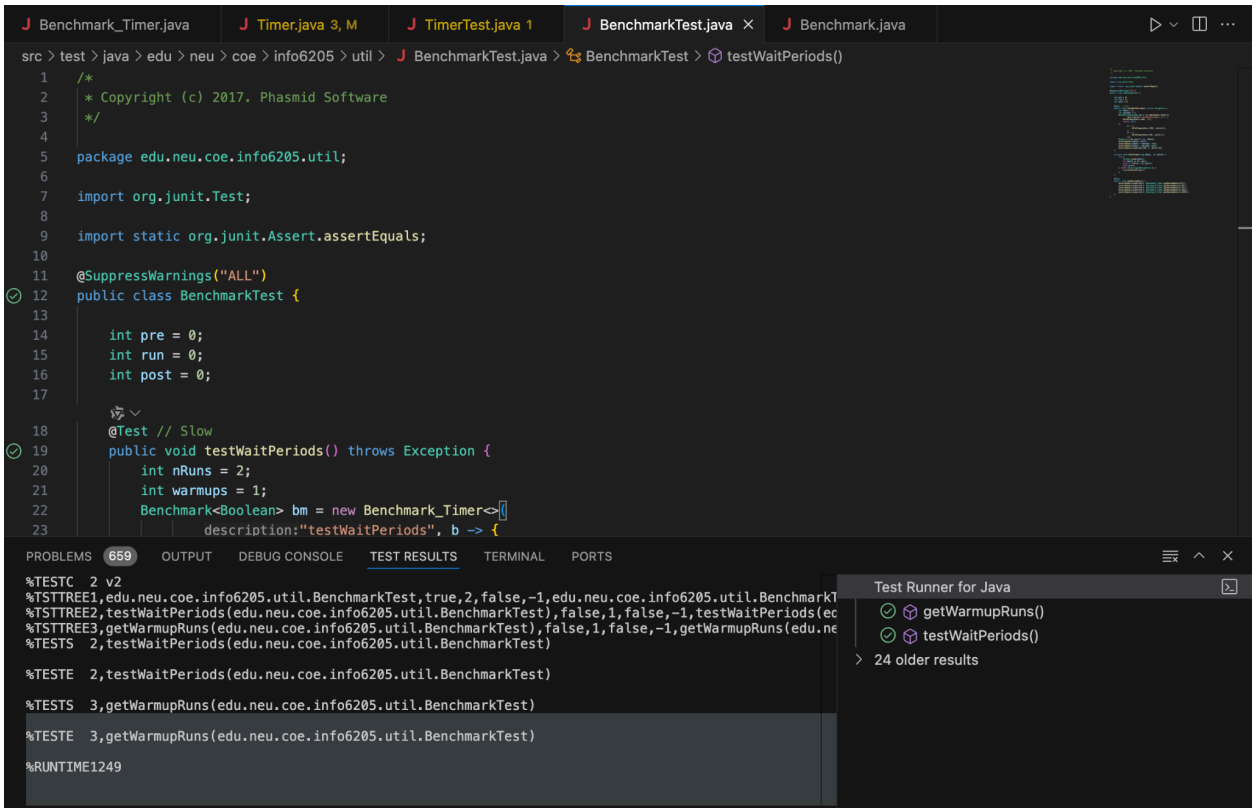
NUID: 002856992

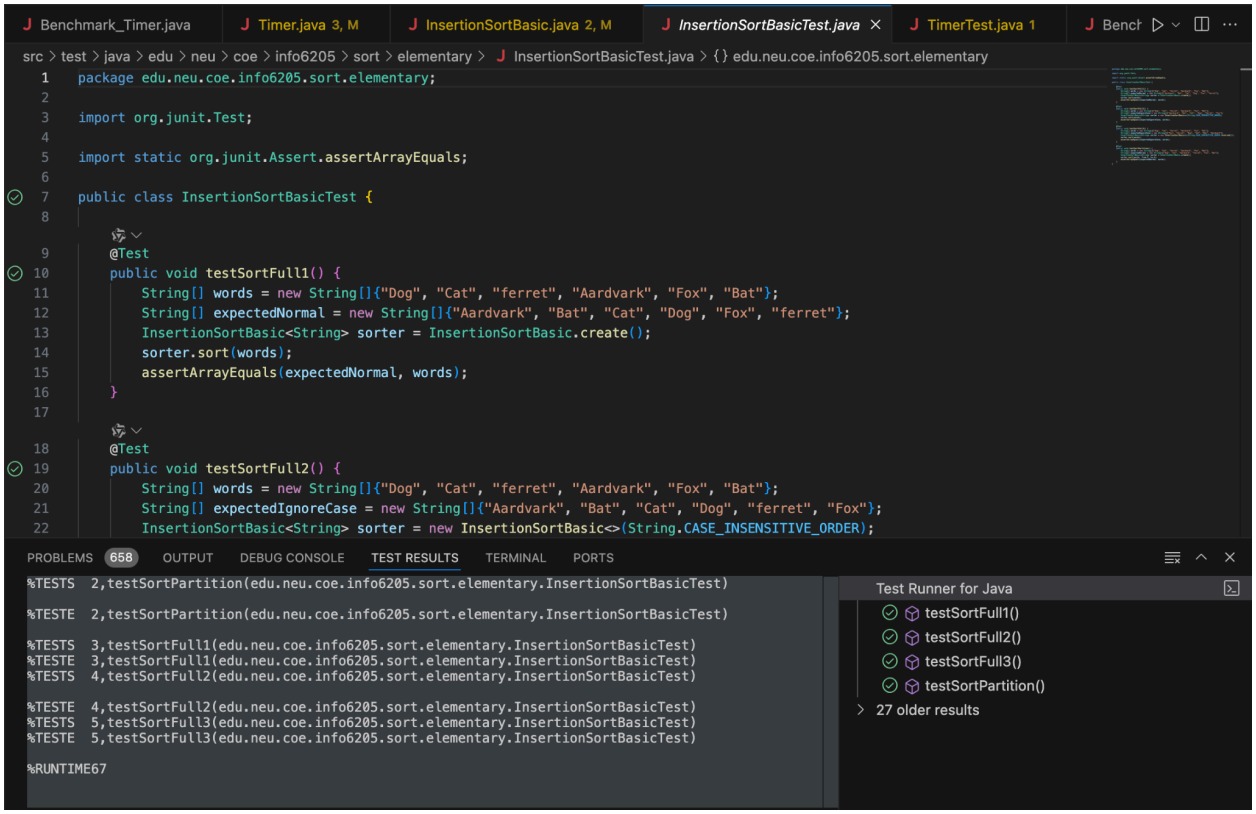
Code Screenshots:



Unit Test Screenshots:


Benchmark_Timer.java
Timer.java 3, M
TimerTest.java 1
ThreeSumTest.java
Benchmark.java
src > test > java > edu > neu > coe > info6205 > util > TimerTest.java > TimerTest > testRepeat3()
package edu.neu.coe.info6205.util;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
public class TimerTest {
@Before
public void setup() {
pre = 0;
run = 0;
post = 0;
result = 0;
}
PROBLEMS 659
OUTPUT
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
%TESTE 7,testMillisecs(edu.neu.coe.info6205.util.TimerTest)
%TESTS 8,testRepeat1(edu.neu.coe.info6205.util.TimerTest)
%TESTE 8,testRepeat1(edu.neu.coe.info6205.util.TimerTest)
%TESTS 9,testRepeat2(edu.neu.coe.info6205.util.TimerTest)
%TESTE 9,testRepeat2(edu.neu.coe.info6205.util.TimerTest)
%TESTS 10,testRepeat3(edu.neu.coe.info6205.util.TimerTest)
%TESTE 10,testRepeat3(edu.neu.coe.info6205.util.TimerTest)
%TESTS 11,testRepeat4(edu.neu.coe.info6205.util.TimerTest)
%TESTE 11,testRepeat4(edu.neu.coe.info6205.util.TimerTest)
%TESTS 12,testPauseAndLap(edu.neu.coe.info6205.util.TimerTest)
%TESTE 12,testPauseAndLap(edu.neu.coe.info6205.util.TimerTest)
%RUNTIME2748
Test Runner for Java
testLap()
testMillisecs()
testPause()
testPauseAndLap()
testPauseAndLapResume0()
testPauseAndLapResume1()
testRepeat1()
testRepeat2()
testRepeat3()
testRepeat4()
testStop()
10 older results


Benchmark_Timer.java
Timer.java 3, M
TimerTest.java 1
BenchmarkTest.java
Benchmark.java
src > test > java > edu > neu > coe > info6205 > util > BenchmarkTest.java > BenchmarkTest > testWaitPeriods()
/*
* Copyright (c) 2017. Phasmid Software
*/
package edu.neu.coe.info6205.util;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
@SuppressWarnings("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 = 1;
Benchmark<Boolean> bm = new Benchmark_Timer<>(
description:"testWaitPeriods", b -> {
PROBLEMS 659
OUTPUT
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
%TESTC 2 v2
%TSTTREE1,edu.neu.coe.info6205.util.BenchmarkTest,true,2,false,-1,edu.neu.coe.info6205.util.BenchmarkT
%TSTTREE2,testWaitPeriods(edu.neu.coe.info6205.util.BenchmarkTest),false,1,false,-1,testWaitPeriods(ec
%TSTTREE3,getWarmupRuns(edu.neu.coe.info6205.util.BenchmarkTest),false,1,false,-1,getWarmupRuns(edu.ne
%TESTS 2,testWaitPeriods(edu.neu.coe.info6205.util.BenchmarkTest)
%TESTE 2,testWaitPeriods(edu.neu.coe.info6205.util.BenchmarkTest)
%TESTS 3,getWarmupRuns(edu.neu.coe.info6205.util.BenchmarkTest)
%TESTE 3,getWarmupRuns(edu.neu.coe.info6205.util.BenchmarkTest)
%RUNTIME1249
Test Runner for Java
getWarmupRuns()
testWaitPeriods()
24 older results



Observations:

Array Size	Random(ns)	Ordered(ns)	Partially Ordered(ns)	Reverse Ordered(ns)
100	110971	8141	81173	234761
200	468000	17512	179474	1979499
400	1742699	41887	518093	431169
800	742119	15238	196180	1550092
1600	3494032	13034	881024	7091524

Explanation:

Time consumption: ordered array < partially ordered array < reverse ordered array < random array

Since the array was already ordered, sorting takes the least time. Insertion sort is effective for sorting ordered arrays.

Partially ordered arrays perform more operation and use more time than ordered arrays. Since it has some elements in the correct position, insertion sort performs better than sorting random and reverse-ordered arrays.

A reverse-ordered array is the worst case for insertion sort, as all elements have to change their position to the correct place.

Sorting a random array takes a lot of time, especially when the array size increases.

As the array size increases, the time used for sorting grows. For an already ordered array, it still takes the least time to sort.