Program Structures and Algorithms
Spring 2023(SEC – 01)

NAME: Neha Rastogi
NUID: 002709191
ASSIGNMENT: 3

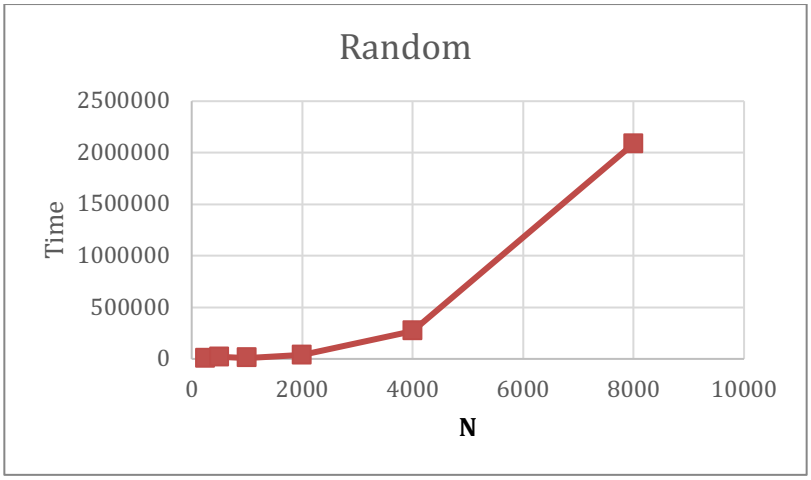
Task: For this assignment the tasks were divided into 3 parts
Part 1: To implement 3 methods namely *repeat*, *getClock*, and *toMillisecs* of class *Timer* and then run the unit tests in *BenchmarkTest and TimerTest*
Part 2: Implement *InsertionSort* in the *InsertionSort* class and run the unit tests in *InsertionSortTest*
Part 3: Implement a main program, or you could do it via your own unit tests, to 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. 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.

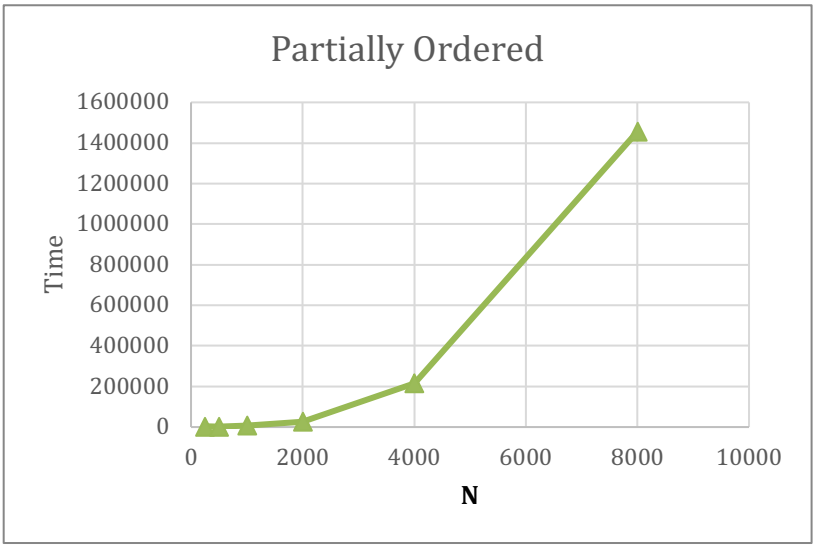
Relationship Conclusion:
For Parts 1 & 2, the necessary code was fixed and the test cases passed.

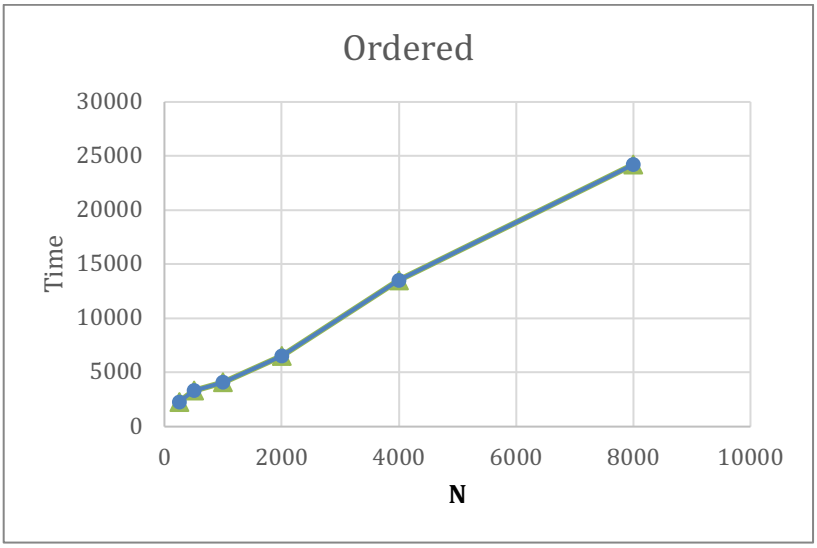
For Part 3, on plotting the times taken by differently ordered arrays, we observe that Reverse Ordered takes the most time followed by Random, then Partially Ordered while Ordered array takes the least time.

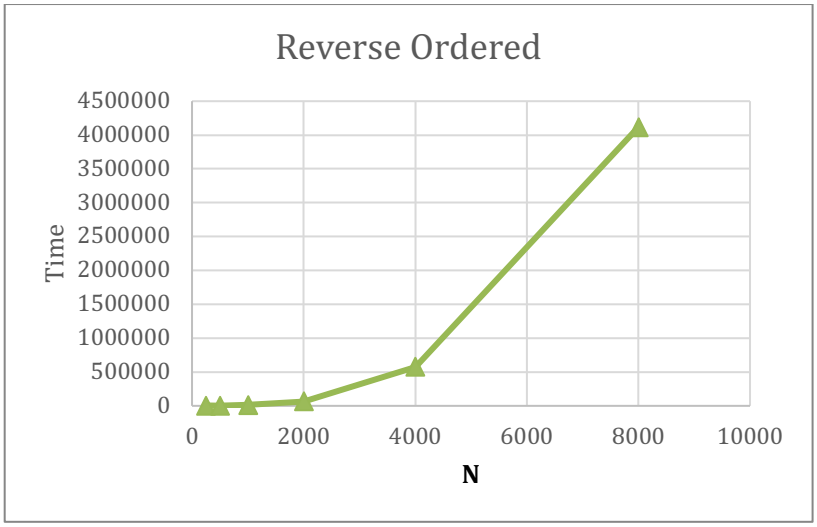
ReverseOrdered > RandomOrdered > PartiallyOrdered > Ordered

Taking N as size of the array, Insertion Sort has the best case for an already sorted array and requires only N-1 comparisons and no swaps thereby resulting in a linear graph.
The worst case is observed when the array is in the reverse order and for each element the entire sorted array needs to be traversed again resulting in N(N-1)/2 comparisons and N(N-1)/2 swaps which is of the order of N^2 thereby resulting in a quadratic graph.
For a random sorted array undergoing Insertion sort we can consider the average time complexity required which is nearly of the order of N^2 due to $N^2/4$ comparisons and $N^2/4$ swaps.
For a partially sorted array, the number of comparisons gets reduced drastically and that the complexity is reduced from the worst case, however, the number of comparisons and swaps will lie between the best-case and worst-case scenarios, depending on the degree of sorting necessary. On average, the number of comparisons and swaps will be closer to the best-case scenario than the worst-case scenario. So, we can say it lies in the range of O(N+d), where d is number of swaps.


Random
2500000
2000000
1500000
1000000
500000
0
Time
0
2000
4000
6000
8000
10000
N

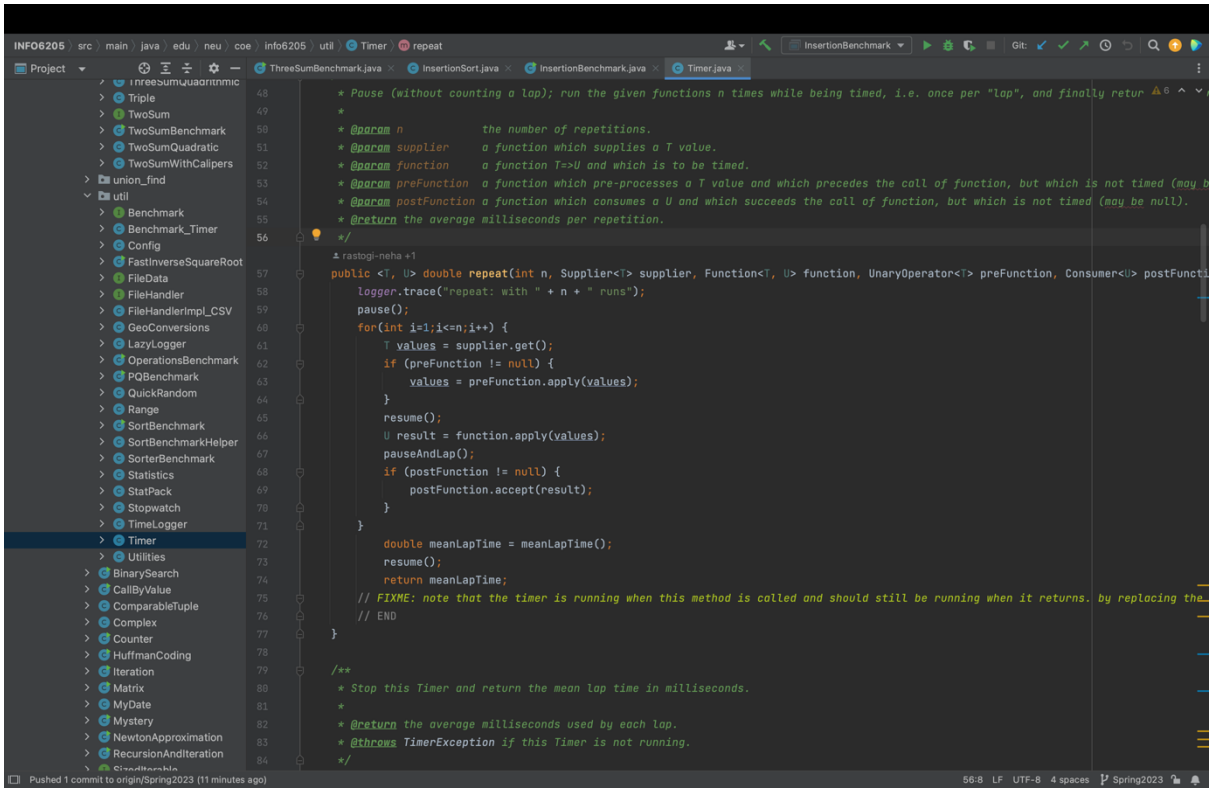

Partially Ordered
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Time
0
2000
4000
6000
8000
10000
N

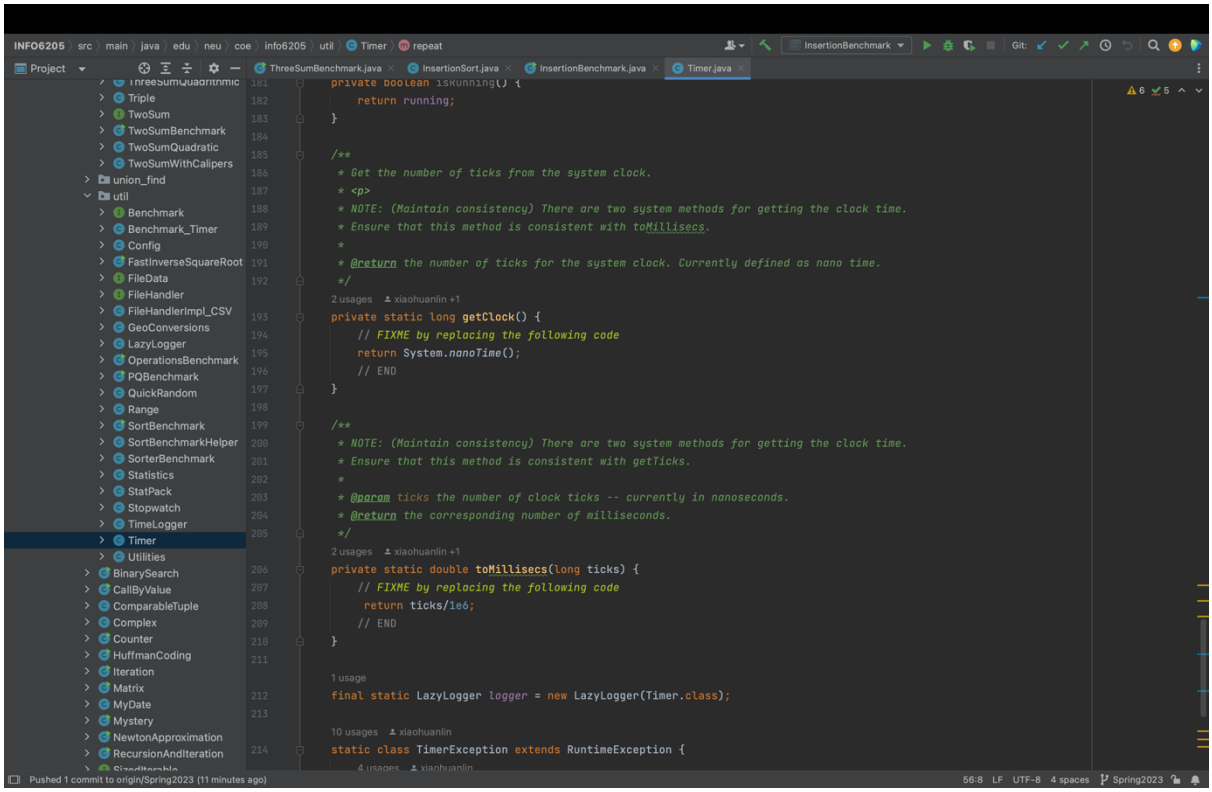

Ordered
30000
25000
20000
15000
10000
5000
0
Time
0
2000
4000
6000
8000
10000
N



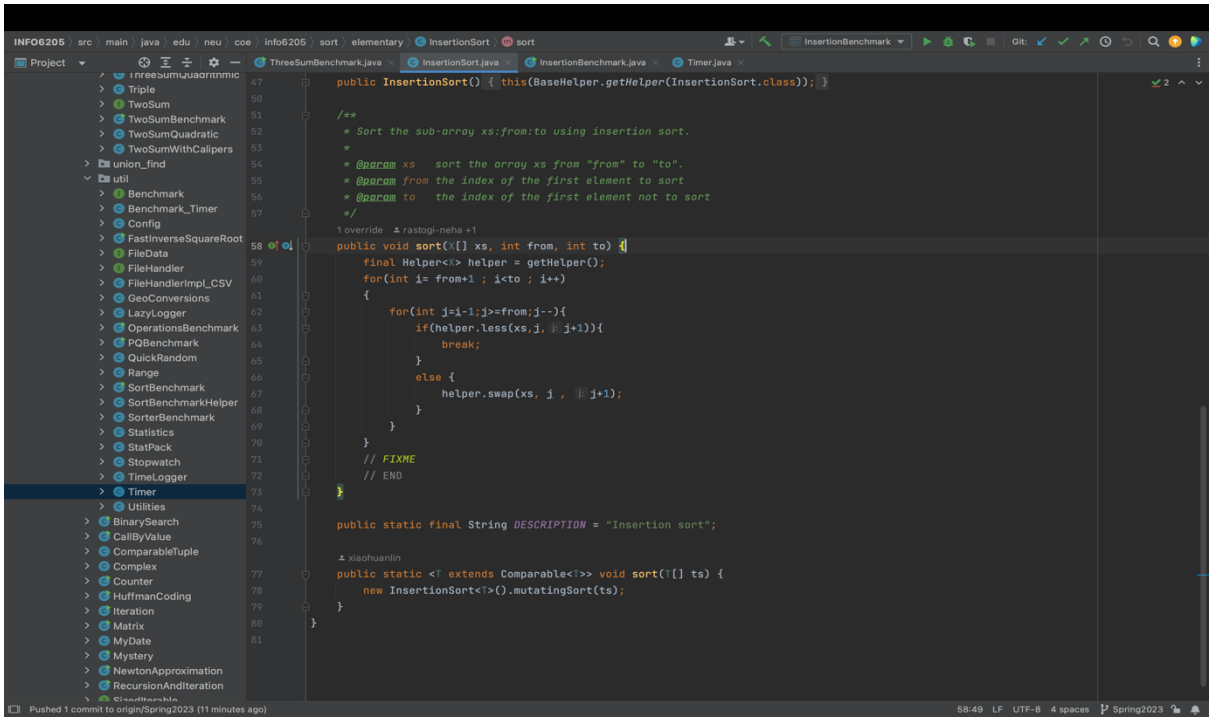
Evidence to support that conclusion:

Part 1:

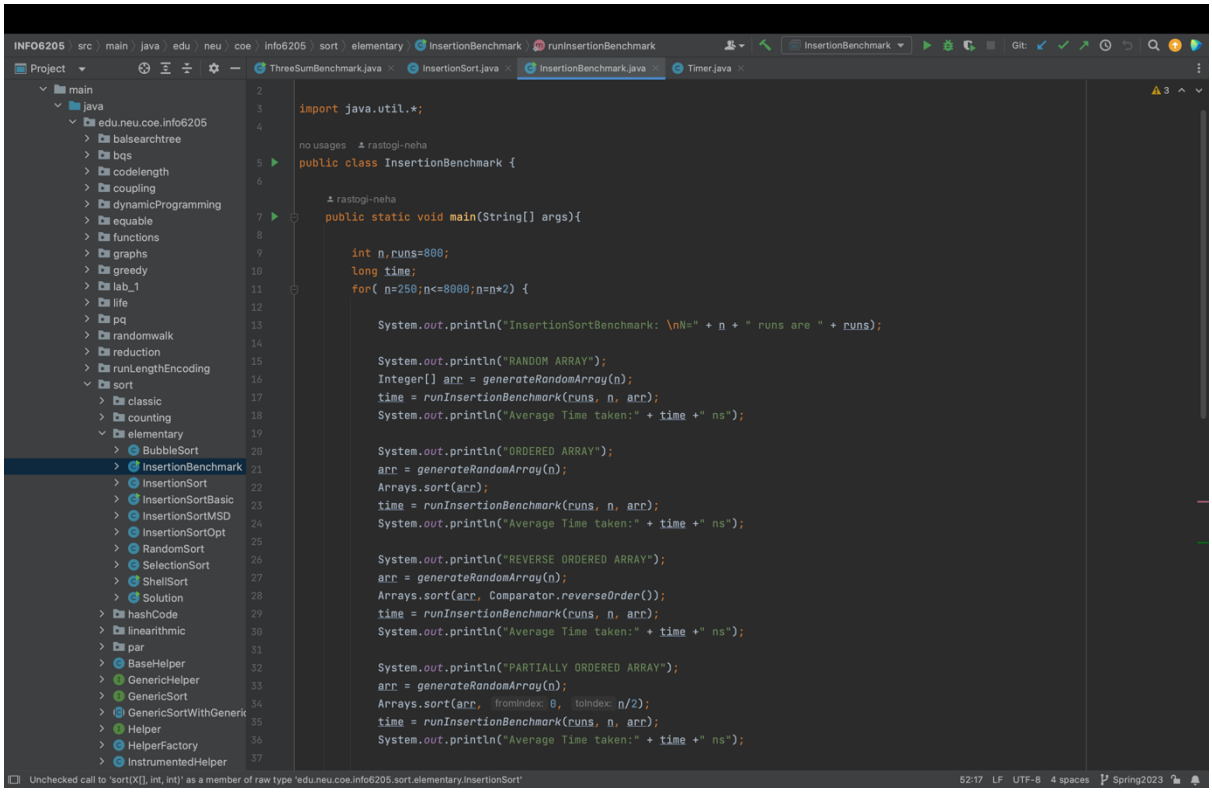




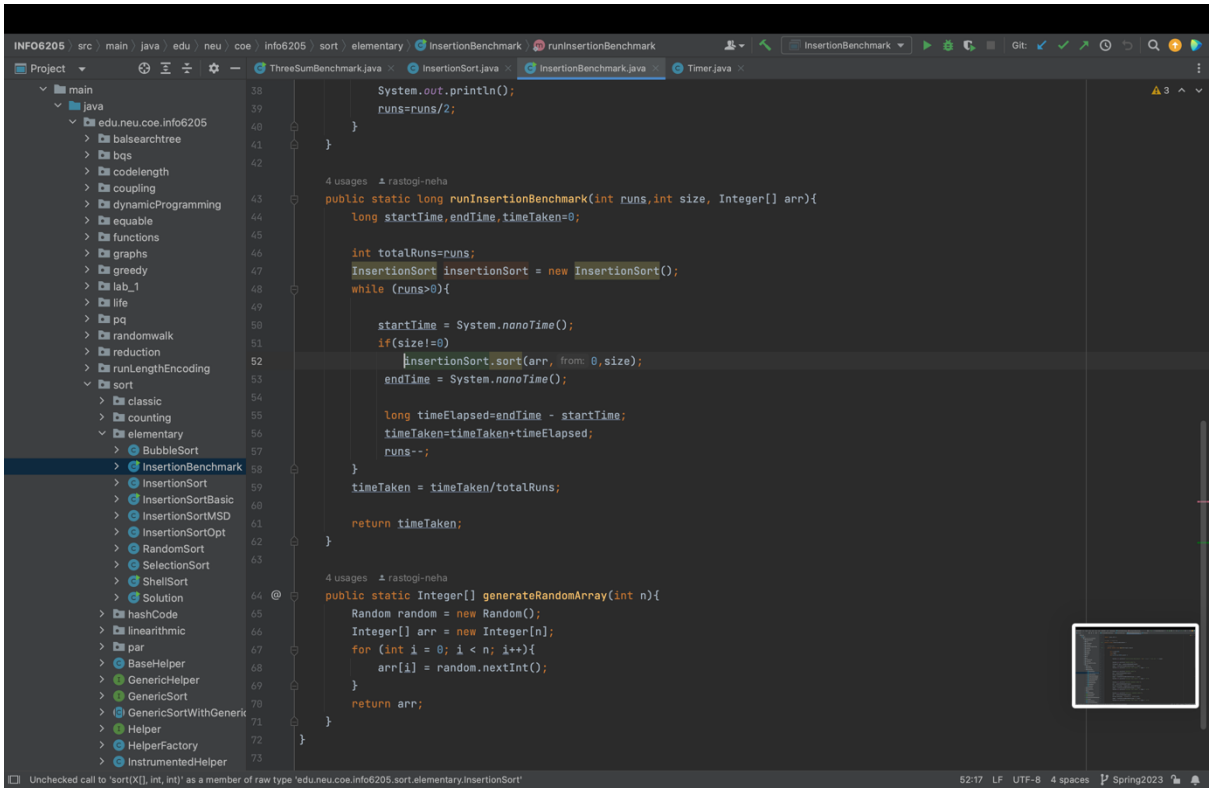
Part 2:



Part 3:



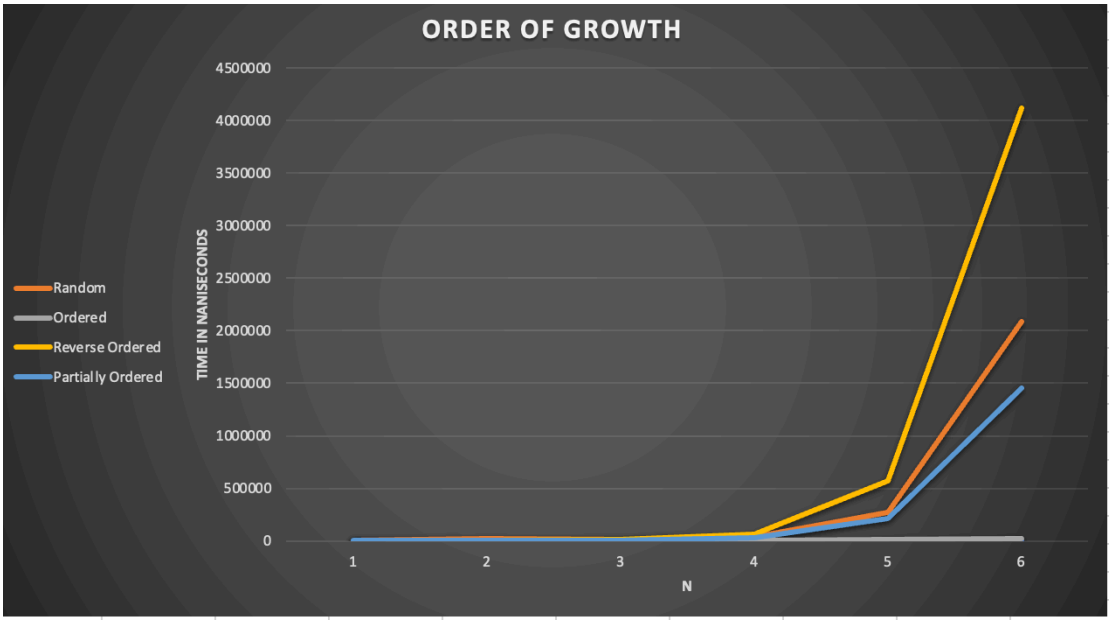
```java
import java.util.*;

no usages  rastogi-neha
public class InsertionBenchmark {

    rastogi-neha
    public static void main(String[] args){

        int n,runs=800;
        long time;
        for( n=250;n<=8000;n=n*2) {

            System.out.println("InsertionSortBenchmark: \nN=" + n + " runs are " + runs);

            System.out.println("RANDOM ARRAY");
            Integer[] arr = generateRandomArray(n);
            time = runInsertionBenchmark(runs, n, arr);
            System.out.println("Average Time taken:" + time +" ns");

            System.out.println("ORDERED ARRAY");
            arr = generateRandomArray(n);
            Arrays.sort(arr);
            time = runInsertionBenchmark(runs, n, arr);
            System.out.println("Average Time taken:" + time +" ns");

            System.out.println("REVERSE ORDERED ARRAY");
            arr = generateRandomArray(n);
            Arrays.sort(arr, Comparator.reverseOrder());
            time = runInsertionBenchmark(runs, n, arr);
            System.out.println("Average Time taken:" + time +" ns");

            System.out.println("PARTIALLY ORDERED ARRAY");
            arr = generateRandomArray(n);
            Arrays.sort(arr,  fromIndex: 0,  toIndex: n/2);
            time = runInsertionBenchmark(runs, n, arr);
            System.out.println("Average Time taken:" + time +" ns");
```



```java
            System.out.println();
            runs=runs/2;
        }
    }

    4 usages  rastogi-neha
    public static long runInsertionBenchmark(int runs,int size, Integer[] arr){
        long startTime,endTime,timeTaken=0;

        int totalRuns=runs;
        InsertionSort insertionSort = new InsertionSort();
        while (runs>0){

            startTime = System.nanoTime();
            if(size!=0)
                insertionSort.sort(arr, from: 0,size);
            endTime = System.nanoTime();

            long timeElapsed=endTime - startTime;
            timeTaken=timeTaken+timeElapsed;
            runs--;
        }
        timeTaken = timeTaken/totalRuns;

        return timeTaken;
    }

    4 usages  rastogi-neha
    public static Integer[] generateRandomArray(int n){
        Random random = new Random();
        Integer[] arr = new Integer[n];
        for (int i = 0; i < n; i++){
            arr[i] = random.nextInt();
        }
        return arr;
    }
}
```

Graphical Representation:

N	Random	Ordered	Reverse Ordered	Partially Ordered
250	7447	2282	4743	3371
500	20080	3316	2760	2176
1000	9828	4104	14800	7365
2000	36779	6514	65669	27530
4000	268870	13521	575401	216250
8000	2087775	24203	4116978	1455609



The order of growth for Reverse(in yellow) is tremendous. It is QUADRATIC in nature.

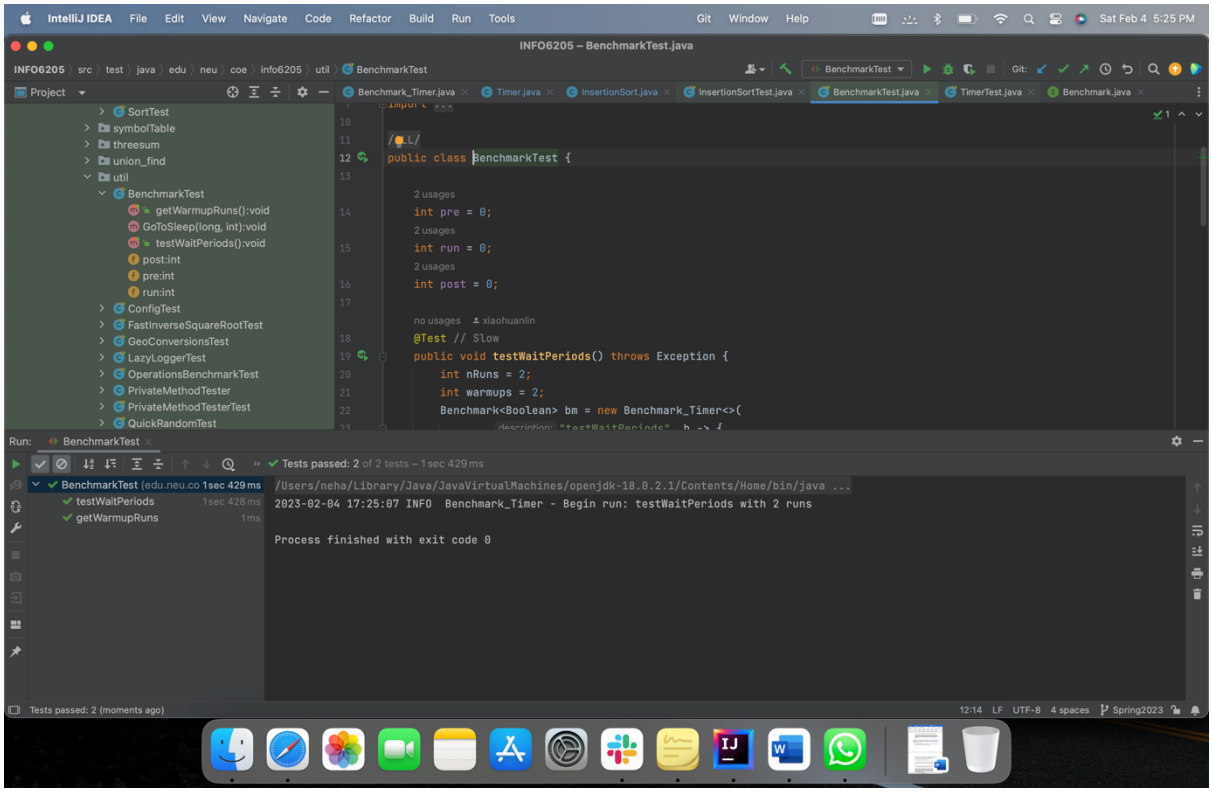
The order of growth for Random(in orange) is not as steep as Reverse. It is approaching quadratic in nature.

Partially ordered(in blue) performs better due to lesser comparisons and swaps.
Meanwhile, Ordered (in white) is linear and requires the least swaps and no comparisons.

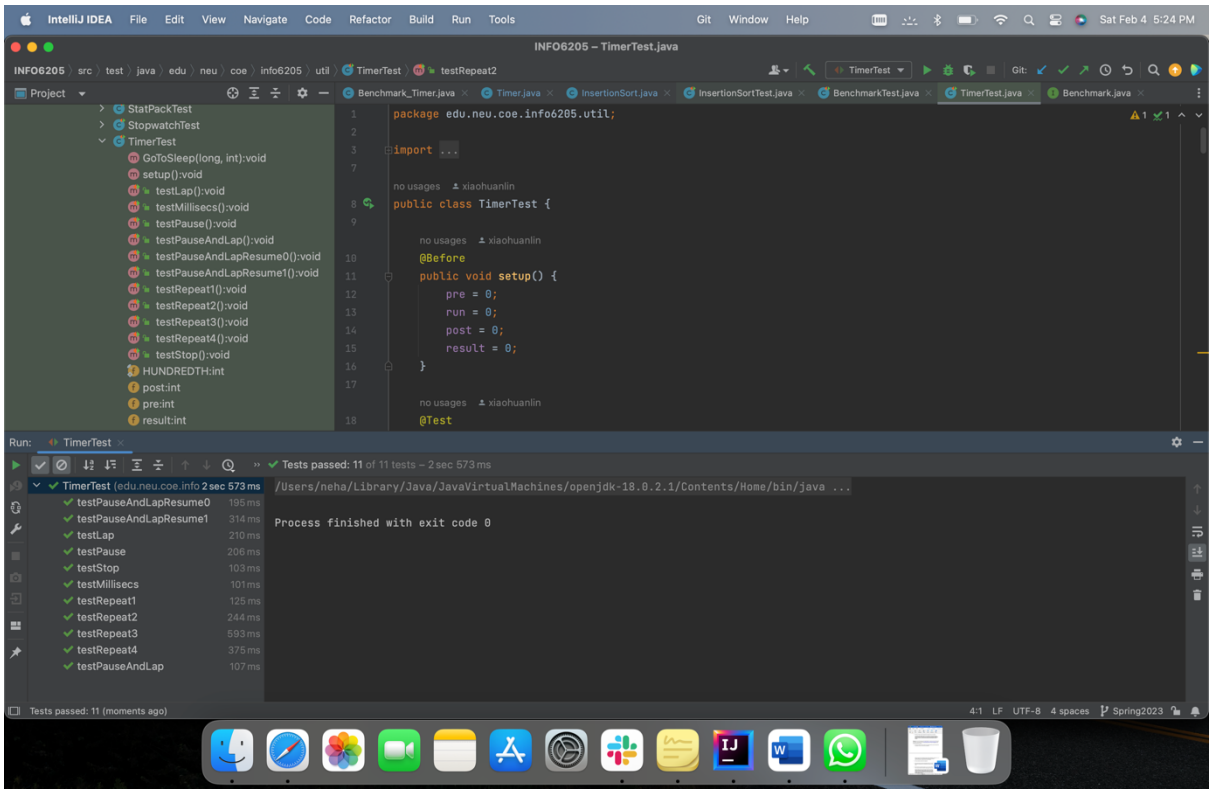
Unit Test Screenshots:

Part 1:

Benchmark Test

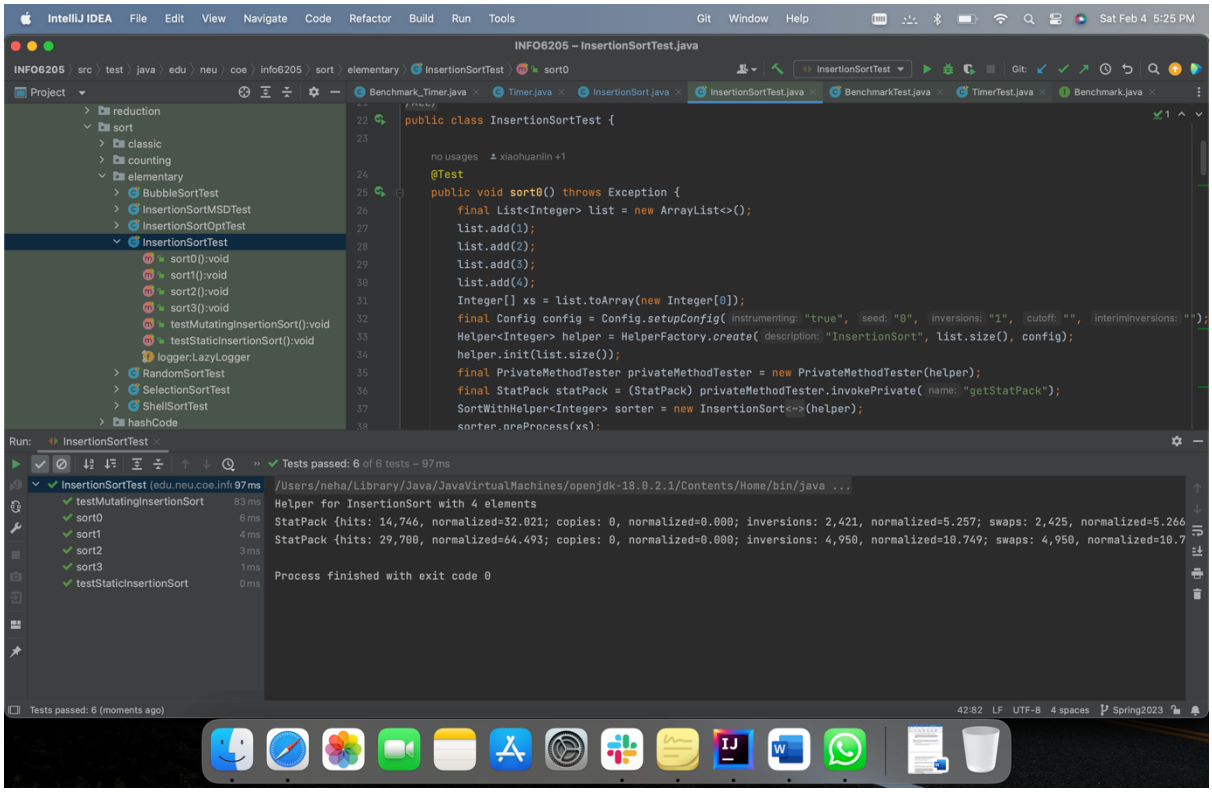


Timer Test

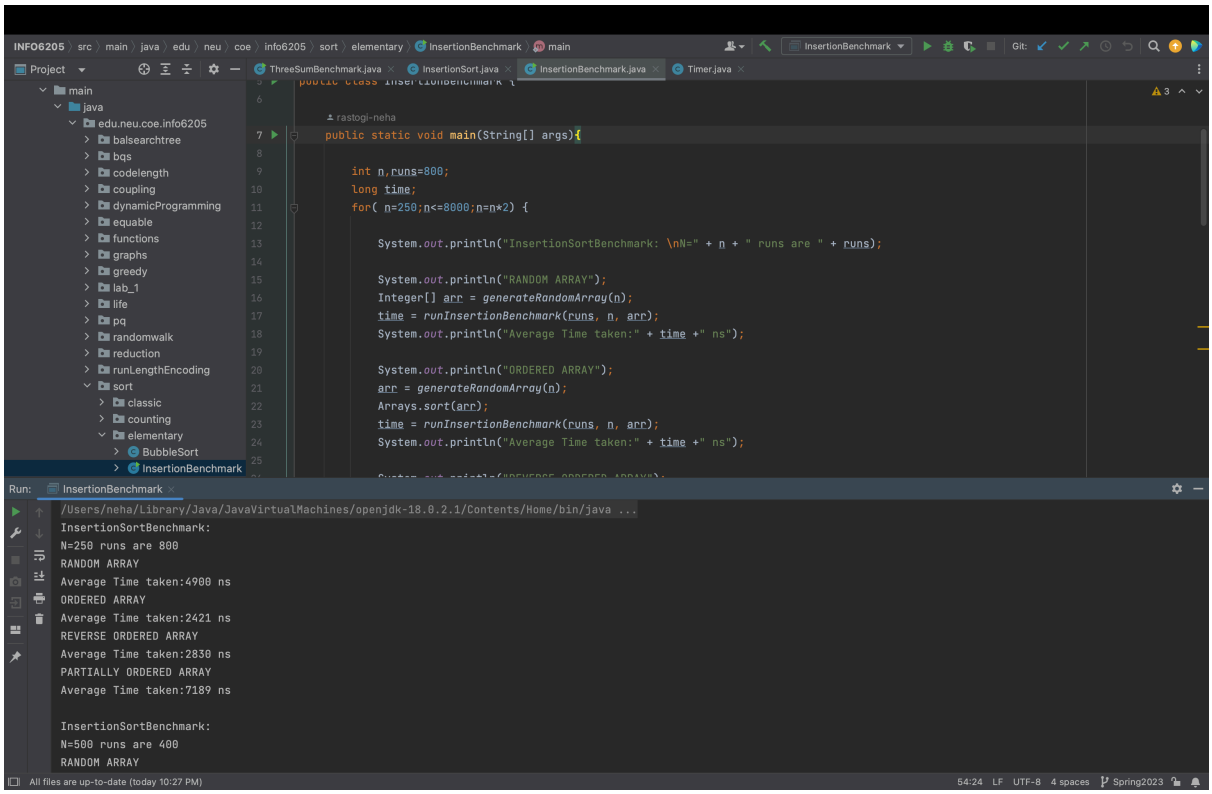


Part 2:

Insertion Sort Test


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help Sat Feb 4 5:25 PM
INFO6205 – InsertionSortTest.java
Project
reduction
sort
classic
counting
elementary
BubbleSortTest
InsertionSortMSDTest
InsertionSortOptTest
InsertionSortTest
sort0():void
sort1():void
sort2():void
sort3():void
testMutatingInsertionSort():void
testStaticInsertionSort():void
logger:LazyLogger
RandomSortTest
SelectionSortTest
ShellSortTest
hashCode
Benchmark_Timer.java
Timer.java
InsertionSort.java
InsertionSortTest.java
BenchmarkTest.java
TimerTest.java
Benchmark.java
public class InsertionSortTest {
no usages xiaohuanlin +1
@Test
public void sort0() throws Exception {
final List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.add(4);
Integer[] xs = list.toArray(new Integer[0]);
final Config config = Config.setupConfig(instrumenting: "true", seed: "0", inversions: "1", cutoff: "", interimInversions: "");
Helper<Integer> helper = HelperFactory.create(description: "InsertionSort", list.size(), config);
helper.init(list.size());
final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
final StatPack statPack = (StatPack) privateMethodTester.invokePrivate(name: "getStatPack");
SortWithHelper<Integer> sorter = new InsertionSort<>(helper);
Run: InsertionSortTest
Tests passed: 6 of 6 tests – 97 ms
InsertionSortTest (edu.neu.coe.info 97 ms
testMutatingInsertionSort 83 ms
sort0 6 ms
sort1 4 ms
sort2 3 ms
sort3 1 ms
testStaticInsertionSort 0 ms
/Users/neha/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java ...
Helper for InsertionSort with 4 elements
StatPack {hits: 14,746, normalized=32.021; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,425, normalized=5.266
StatPack {hits: 29,700, normalized=64.493; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=10.7
Process finished with exit code 0
Tests passed: 6 (moments ago)
42:82 LF UTF-8 4 spaces Spring2023

Part 3:


INFO6205 src main java edu neu coe info6205 sort elementary InsertionBenchmark main
Project
main
java
edu.neu.coe.info6205
balsearchtree
bqs
codelength
coupling
dynamicProgramming
equable
functions
graphs
greedy
lab_1
life
pq
randomwalk
reduction
runLengthEncoding
sort
classic
counting
elementary
BubbleSort
InsertionBenchmark
ThreeSumBenchmark.java
InsertionSort.java
InsertionBenchmark.java
Timer.java
rastogi-neha
public static void main(String[] args){
int n,runs=800;
long time;
for(n=250;n<=8000;n=n*2) {
System.out.println("InsertionSortBenchmark: \nN=" + n + " runs are " + runs);
System.out.println("RANDOM ARRAY");
Integer[] arr = generateRandomArray(n);
time = runInsertionBenchmark(runs, n, arr);
System.out.println("Average Time taken:" + time +" ns");
System.out.println("ORDERED ARRAY");
arr = generateRandomArray(n);
Arrays.sort(arr);
time = runInsertionBenchmark(runs, n, arr);
System.out.println("Average Time taken:" + time +" ns");
Run: InsertionBenchmark
/Users/neha/Library/Java/JavaVirtualMachines/openjdk-18.0.2.1/Contents/Home/bin/java ...
InsertionSortBenchmark:
N=250 runs are 800
RANDOM ARRAY
Average Time taken:4900 ns
ORDERED ARRAY
Average Time taken:2421 ns
REVERSE ORDERED ARRAY
Average Time taken:2830 ns
PARTIALLY ORDERED ARRAY
Average Time taken:7189 ns
InsertionSortBenchmark:
N=500 runs are 400
RANDOM ARRAY
All files are up-to-date (today 10:27 PM)
54:24 LF UTF-8 4 spaces Spring2023