Program Structures and Algorithms
Spring 2024

NAME: Ashish Nevan Gade
NUID: 002889005
GITHUB LINK: https://github.com/AshishNevan/INFO6205

Task:

1. Implement 3 methods (repeat, getClock, toMillisecs) of Timer.java
2. Implement sort method of InsertionSort.java
3. Write a main program to test runtimes of InsertionSort.java on 4 different kinds of array orders (random, sorted order, partially sorted order, reverse sorted order)

Relationship Conclusion:

- **The worst-case runtime of Insertion sort is $O(n^2)$, where n is the input size.**
- **Insertion sort is adaptive, meaning that it is faster when input is sorted, either partially or completely.**

Evidence to support that conclusion:
For 5 different values of N, the experiment was repeated 100 times.
The observed values are listed in the spreadsheet below.

	Input size (N)					
	1K	2K	4k	8K	16K	
Input Type	Runtime in MilliSeconds					Growth
Random	0.84979	2.35433	8.92456	36.9187	158.202	2.316492997
Sorted	0.07771	0.05028	0.04877	0.05387	0.11516	-0.461194453
Reverse Sorted	1.37707	4.61172	18.0737	73.8805	303.581	2.603710961
Partially Sorted	0.50166	1.71638	6.74641	28.4773	111.669	2.173508208

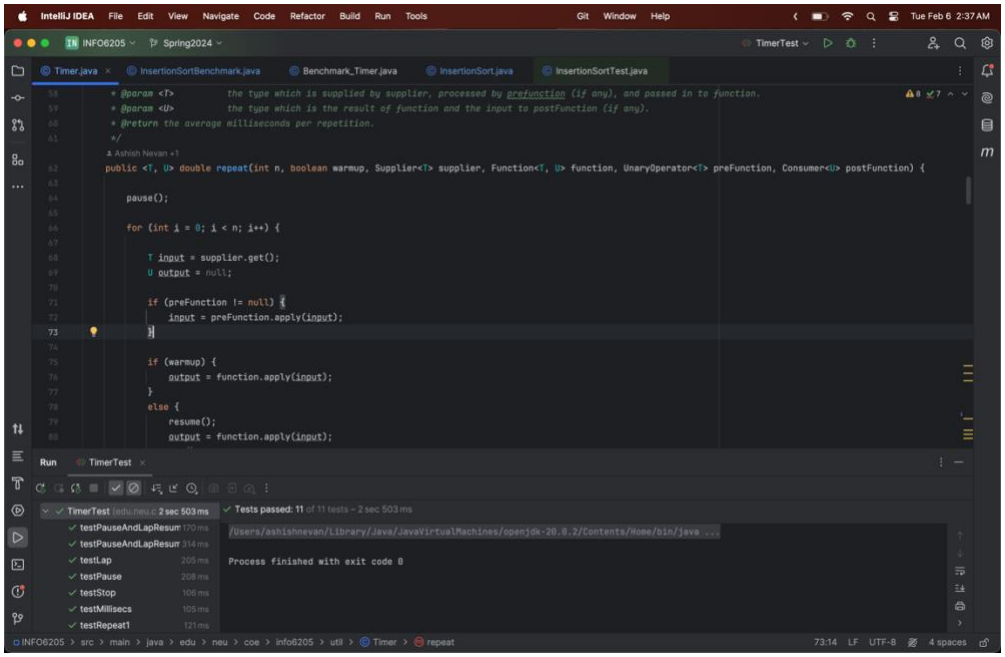
Figure 1: Runtimes and Growth

In worst-case scenario, meaning when the input array is reverse sorted, the growth (LOG(SUM(runtimes)) of this sorting algorithm is in the order of 2. Another important observation is that when a sorted array is given as input, the growth is close to 1, (-0.46 is shown because of the decrease between 1K to 4K, maybe how memory works?).

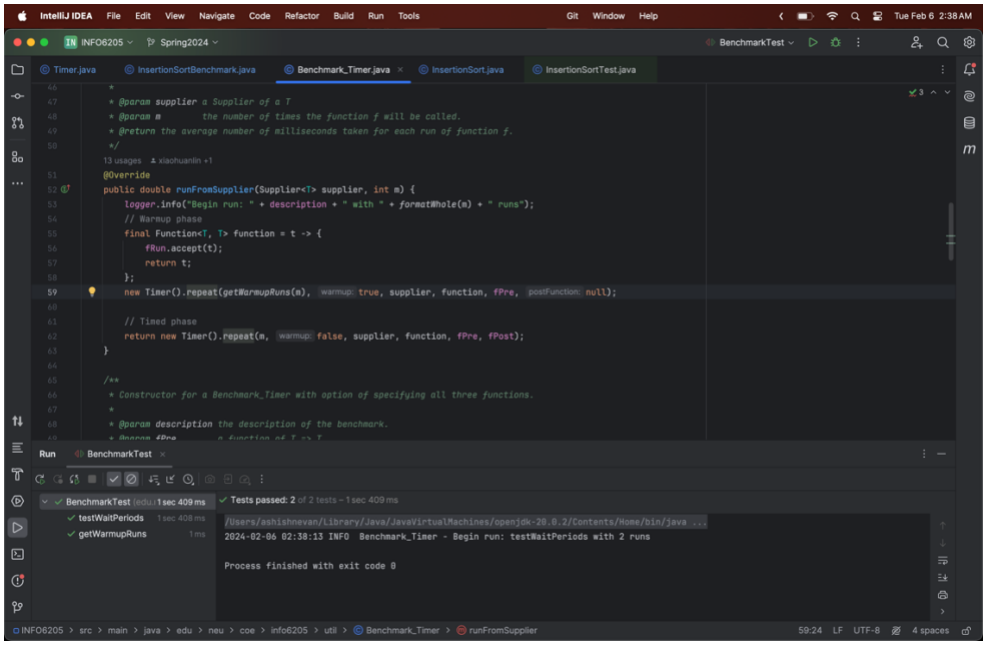
From this I conclude that the insertion sort algorithm has O(N) runtime.

Unit Test Screenshots:

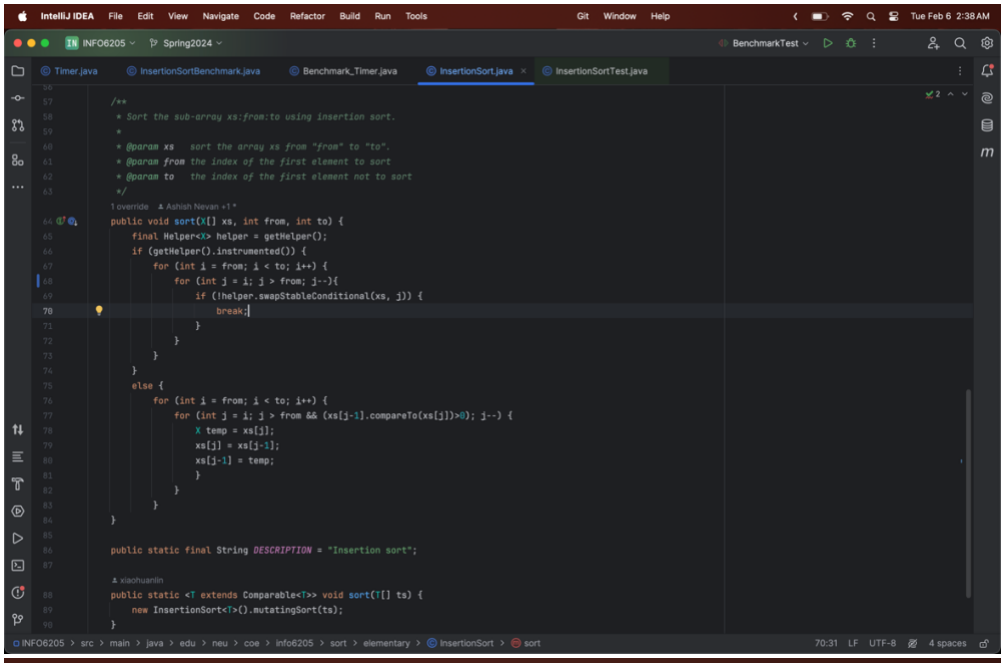
Task 1: Timer.java and TimerTest passed



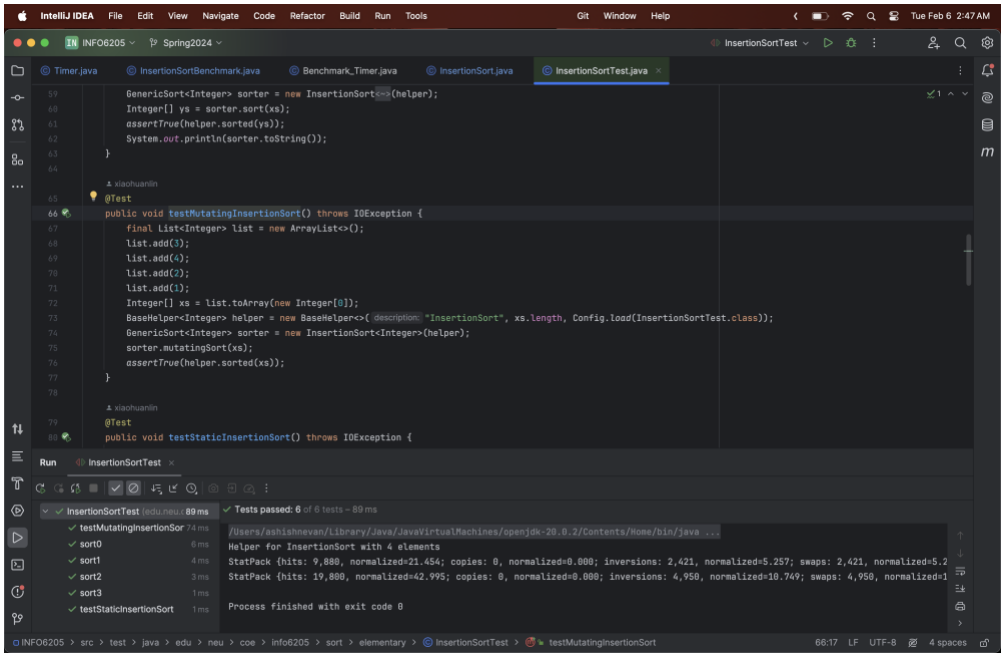
Benchmark_Timer.java and BenchmarkTest passed



InsertionSort.java and InsertionSortTest.java

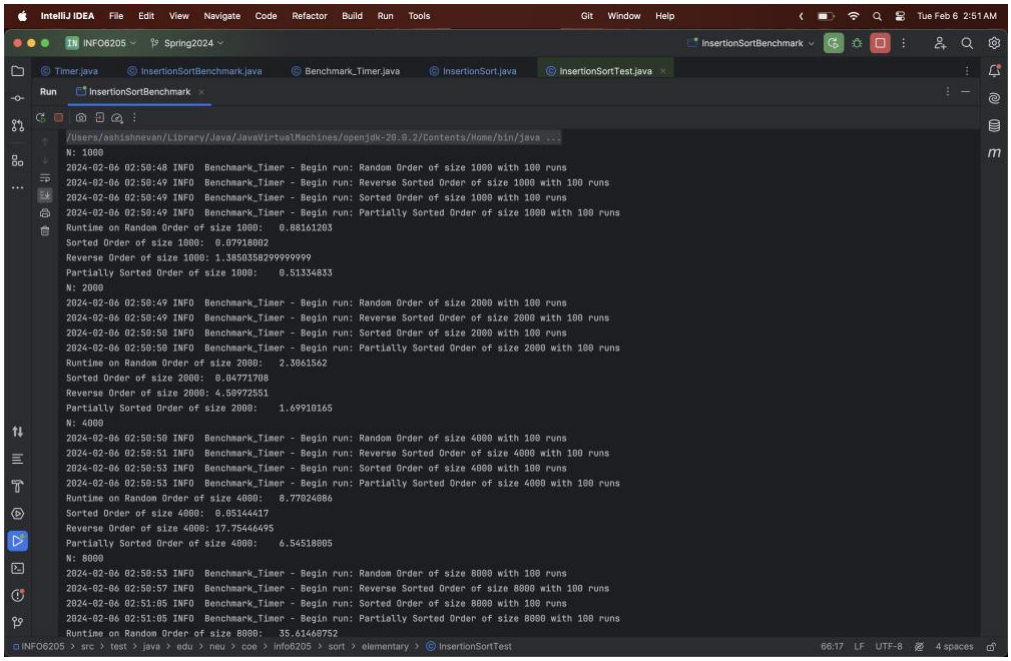


```java
/**
 * Sort the sub-array xs:from:to using insertion sort.
 *
 * @param xs   sort the array xs from "from" to "to".
 * @param from the index of the first element to sort
 * @param to   the index of the first element not to sort
 */
1 override  Ashish Nevan +1 *
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    if (getHelper().instrumented()) {
        for (int i = from; i < to; i++) {
            for (int j = i; j > from; j--){
                if (!helper.swapStableConditional(xs, j)) {
                    break;
                }
            }
        }
    }
    else {
        for (int i = from; i < to; i++) {
            for (int j = i; j > from && (xs[j-1].compareTo(xs[j])>0); j--) {
                X temp = xs[j];
                xs[j] = xs[j-1];
                xs[j-1] = temp;
                }
            }
        }
    }

    public static final String DESCRIPTION = "Insertion sort";

    xiaohuanlin
    public static <T extends Comparable<T>> void sort(T[] ts) {
        new InsertionSort<T>().mutatingSort(ts);
    }
```

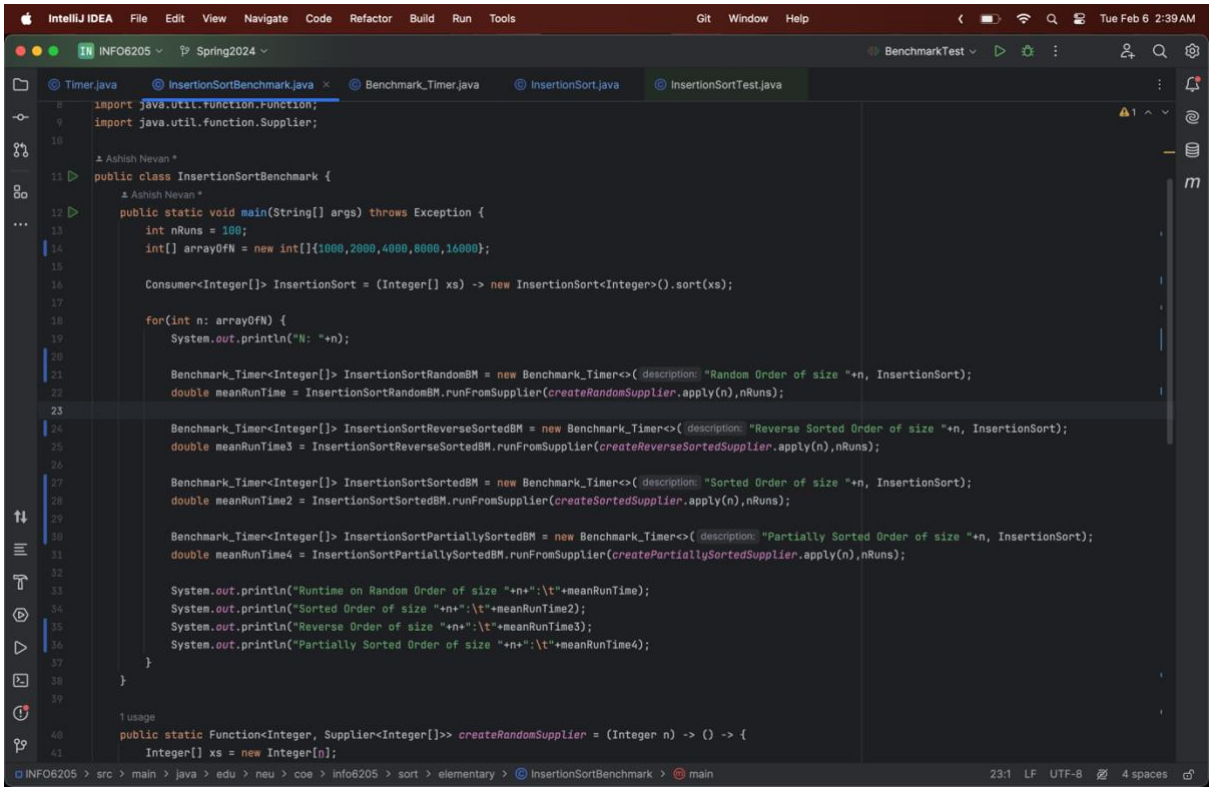


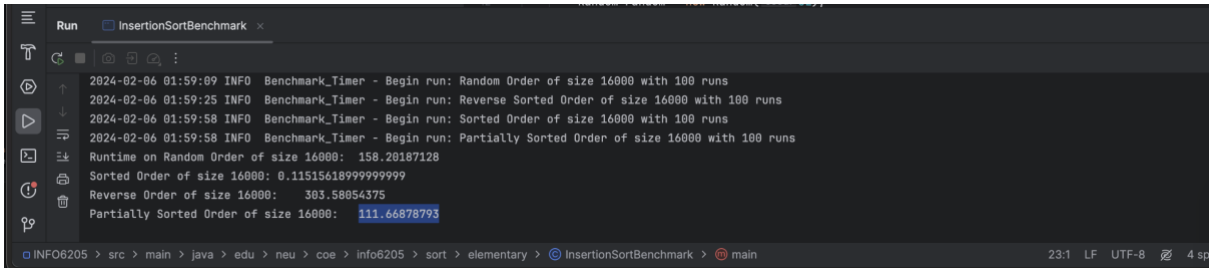
```java
        GenericSort<Integer> sorter = new InsertionSort<>(helper);
        Integer[] ys = sorter.sort(xs);
        assertTrue(helper.sorted(ys));
        System.out.println(sorter.toString());
    }

    xiaohuanlin
    @Test
    public void testMutatingInsertionSort() throws IOException {
        final List<Integer> list = new ArrayList<>();
        list.add(3);
        list.add(4);
        list.add(2);
        list.add(1);
        Integer[] xs = list.toArray(new Integer[0]);
        BaseHelper<Integer> helper = new BaseHelper<>( description: "InsertionSort", xs.length, Config.load(InsertionSortTest.class));
        GenericSort<Integer> sorter = new InsertionSort<Integer>(helper);
        sorter.mutatingSort(xs);
        assertTrue(helper.sorted(xs));
    }

    xiaohuanlin
    @Test
    public void testStaticInsertionSort() throws IOException {
```

```
Run  InsertionSortTest
InsertionSortTest (edu.neu.c 89 ms
  testMutatingInsertionSor 74 ms
  sort0 6 ms
  sort1 4 ms
  sort2 3 ms
  sort3 1 ms
  testStaticInsertionSort 1 ms
Tests passed: 6 of 6 tests – 89 ms
/Users/ashishnevan/Library/Java/JavaVirtualMachines/openjdk-20.0.2/Contents/Home/bin/java ...
Helper for InsertionSort with 4 elements
StatPack {hits: 9,880, normalized=21.454; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normalized=5.2
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=1

Process finished with exit code 0
```



InsertionSortBenchmark.java


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help Tue Feb 6 2:39 AM
INFO6205 Spring2024 BenchmarkTest
Timer.java InsertionSortBenchmark.java Benchmark_Timer.java InsertionSort.java InsertionSortTest.java
import java.util.function.Function;
import java.util.function.Supplier;
Ashish Nevan *
public class InsertionSortBenchmark {
Ashish Nevan *
public static void main(String[] args) throws Exception {
int nRuns = 100;
int[] arrayOfN = new int[]{1000,2000,4000,8000,16000};
Consumer<Integer[]> InsertionSort = (Integer[] xs) -> new InsertionSort<Integer>().sort(xs);
for(int n: arrayOfN) {
System.out.println("N: "+n);
Benchmark_Timer<Integer[]> InsertionSortRandomBM = new Benchmark_Timer<>(description: "Random Order of size "+n, InsertionSort);
double meanRunTime = InsertionSortRandomBM.runFromSupplier(createRandomSupplier.apply(n),nRuns);
Benchmark_Timer<Integer[]> InsertionSortReverseSortedBM = new Benchmark_Timer<>(description: "Reverse Sorted Order of size "+n, InsertionSort);
double meanRunTime3 = InsertionSortReverseSortedBM.runFromSupplier(createReverseSortedSupplier.apply(n),nRuns);
Benchmark_Timer<Integer[]> InsertionSortSortedBM = new Benchmark_Timer<>(description: "Sorted Order of size "+n, InsertionSort);
double meanRunTime2 = InsertionSortSortedBM.runFromSupplier(createSortedSupplier.apply(n),nRuns);
Benchmark_Timer<Integer[]> InsertionSortPartiallySortedBM = new Benchmark_Timer<>(description: "Partially Sorted Order of size "+n, InsertionSort);
double meanRunTime4 = InsertionSortPartiallySortedBM.runFromSupplier(createPartiallySortedSupplier.apply(n),nRuns);
System.out.println("Runtime on Random Order of size "+n+":\t"+meanRunTime);
System.out.println("Sorted Order of size "+n+":\t"+meanRunTime2);
System.out.println("Reverse Order of size "+n+":\t"+meanRunTime3);
System.out.println("Partially Sorted Order of size "+n+":\t"+meanRunTime4);
}
}
1 usage
public static Function<Integer, Supplier<Integer[]>> createRandomSupplier = (Integer n) -> () -> {
Integer[] xs = new Integer[n];
INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > elementary > InsertionSortBenchmark > main
23:1 LF UTF-8 4 spaces


Run InsertionSortBenchmark
2024-02-06 01:59:09 INFO Benchmark_Timer - Begin run: Random Order of size 16000 with 100 runs
2024-02-06 01:59:25 INFO Benchmark_Timer - Begin run: Reverse Sorted Order of size 16000 with 100 runs
2024-02-06 01:59:58 INFO Benchmark_Timer - Begin run: Sorted Order of size 16000 with 100 runs
2024-02-06 01:59:58 INFO Benchmark_Timer - Begin run: Partially Sorted Order of size 16000 with 100 runs
Runtime on Random Order of size 16000: 158.20187128
Sorted Order of size 16000: 0.11515618999999999
Reverse Order of size 16000: 303.58054375
Partially Sorted Order of size 16000: 111.66878793
INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > elementary > InsertionSortBenchmark > main
23:1 LF UTF-8