Program Structures and Algorithms
Fall 2024

NAME: Sanskruti Manoria
NUID: 002643300
GITHUB LINK: https://github.com/sannskruti/INFO6205

Assignment 3

Task:

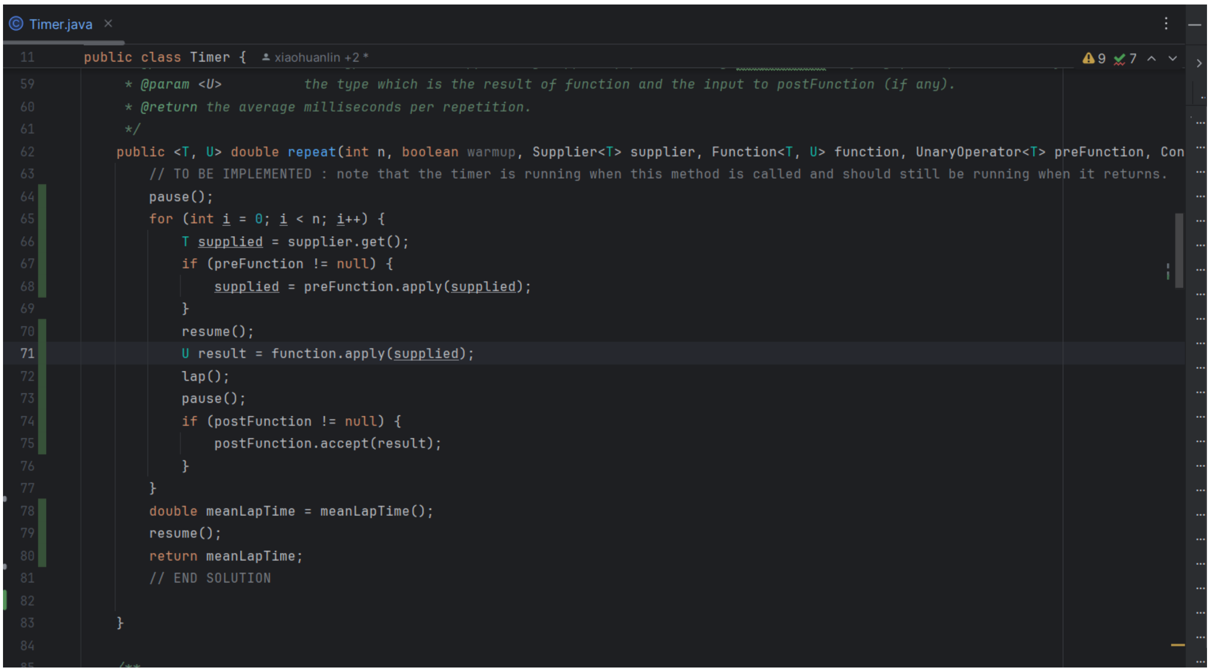
(Part 1) You are to implement three (3) methods (repeat, getClock, and toMillisecs) of a class called Timer

(Part 2) Implement insertion sort (in the InsertionSortBasic class)

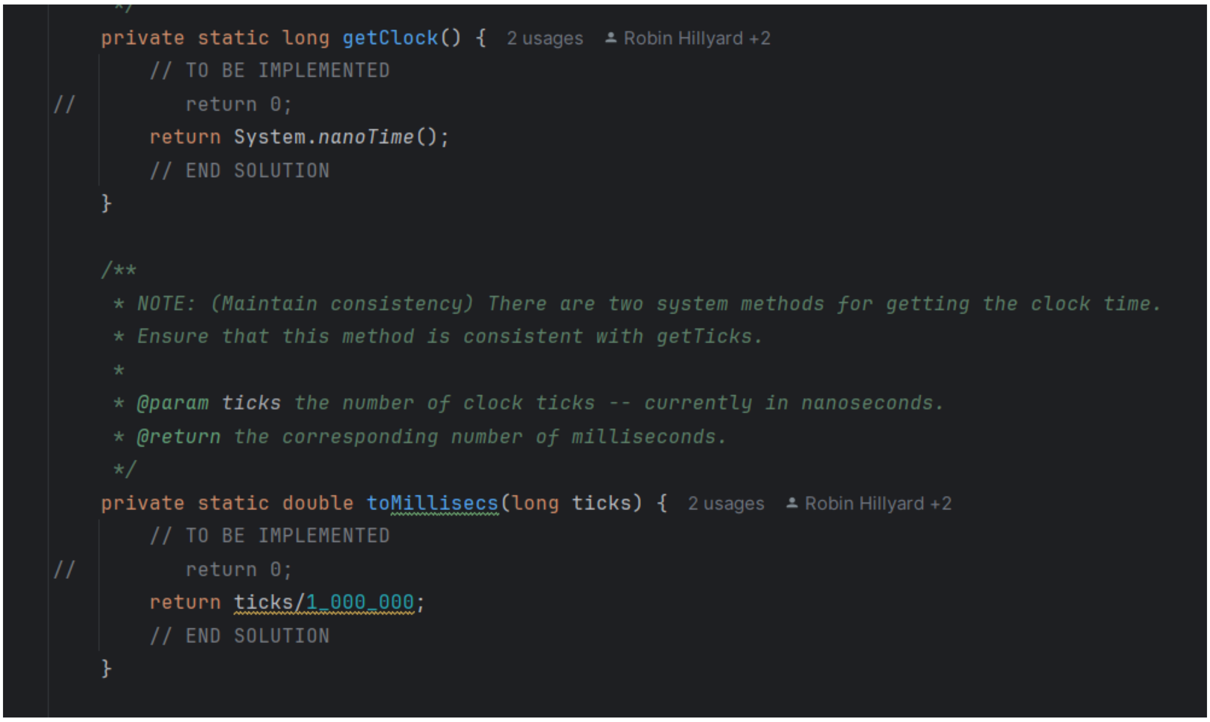
(Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered.

Code Screenshots:

Task 1 – Timer.java

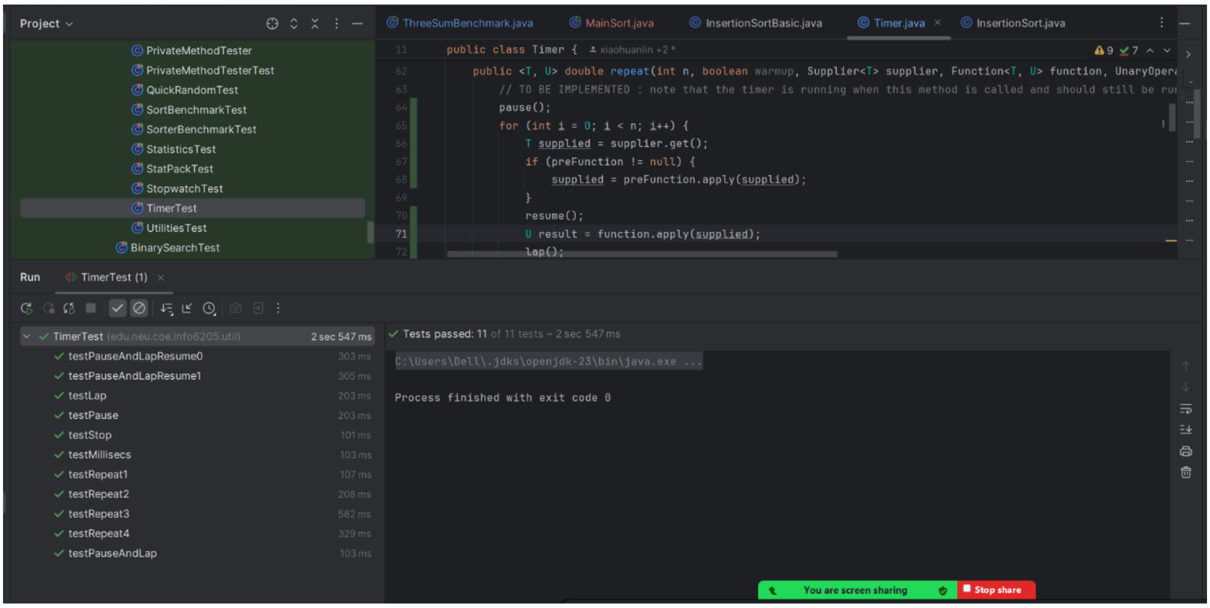


```
public class Timer {
     * @param <U>          the type which is the result of function and the input to postFunction (if any).
     * @return the average milliseconds per repetition.
     */
    public <T, U> double repeat(int n, boolean warmup, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Con
        // TO BE IMPLEMENTED : note that the timer is running when this method is called and should still be running when it returns.
        pause();
        for (int i = 0; i < n; i++) {
            T supplied = supplier.get();
            if (preFunction != null) {
                supplied = preFunction.apply(supplied);
            }
            resume();
            U result = function.apply(supplied);
            lap();
            pause();
            if (postFunction != null) {
                postFunction.accept(result);
            }
        }
        double meanLapTime = meanLapTime();
        resume();
        return meanLapTime;
        // END SOLUTION

    }

    /**
```



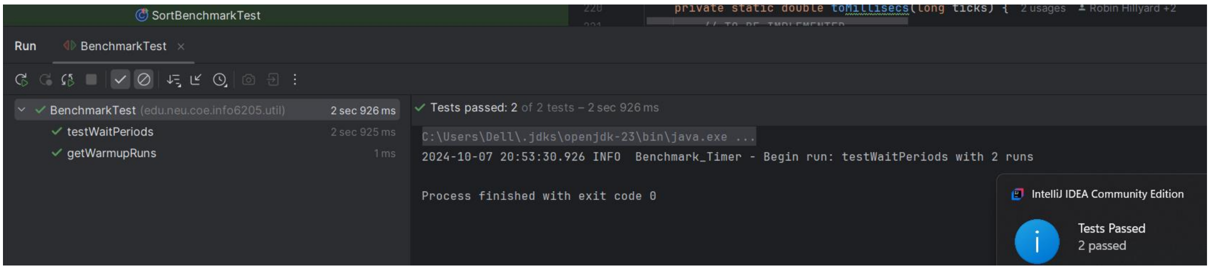
```java
     */
    private static long getClock() {  2 usages  Robin Hillyard +2
        // TO BE IMPLEMENTED
//          return 0;
        return System.nanoTime();
        // END SOLUTION
    }

    /**
     * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
     * Ensure that this method is consistent with getTicks.
     *
     * @param ticks the number of clock ticks -- currently in nanoseconds.
     * @return the corresponding number of milliseconds.
     */
    private static double toMillisecs(long ticks) {  2 usages  Robin Hillyard +2
        // TO BE IMPLEMENTED
//          return 0;
        return ticks/1_000_000;
        // END SOLUTION
    }
```

Output-Unit Test



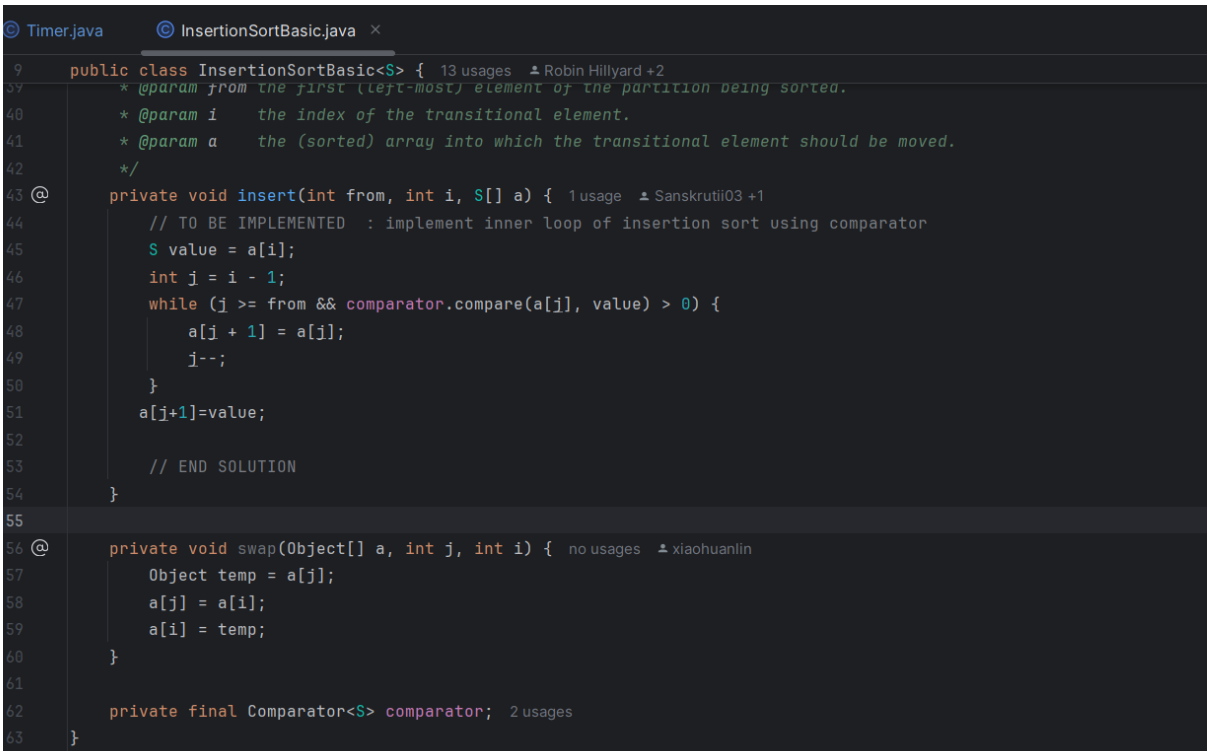
```java
public class Timer {  xiaohuanlin +2 *
    public <T, U> double repeat(int n, boolean warmup, Supplier<T> supplier, Function<T, U> function, UnaryOpera
        // TO BE IMPLEMENTED : note that the timer is running when this method is called and should still be run
        pause();
        for (int i = 0; i < n; i++) {
            T supplied = supplier.get();
            if (preFunction != null) {
                supplied = preFunction.apply(supplied);
            }
            resume();
            U result = function.apply(supplied);
            lap();
```

```
Run   TimerTest (1)

✓ TimerTest (edu.neu.coe.info6205.util)   2 sec 547 ms
    ✓ testPauseAndLapResume0   303 ms
    ✓ testPauseAndLapResume1   305 ms
    ✓ testLap                  203 ms
    ✓ testPause                203 ms
    ✓ testStop                 101 ms
    ✓ testMillisecs            103 ms
    ✓ testRepeat1              107 ms
    ✓ testRepeat2              208 ms
    ✓ testRepeat3              582 ms
    ✓ testRepeat4              329 ms
    ✓ testPauseAndLap          103 ms

✓ Tests passed: 11 of 11 tests – 2 sec 547 ms

C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...

Process finished with exit code 0
```



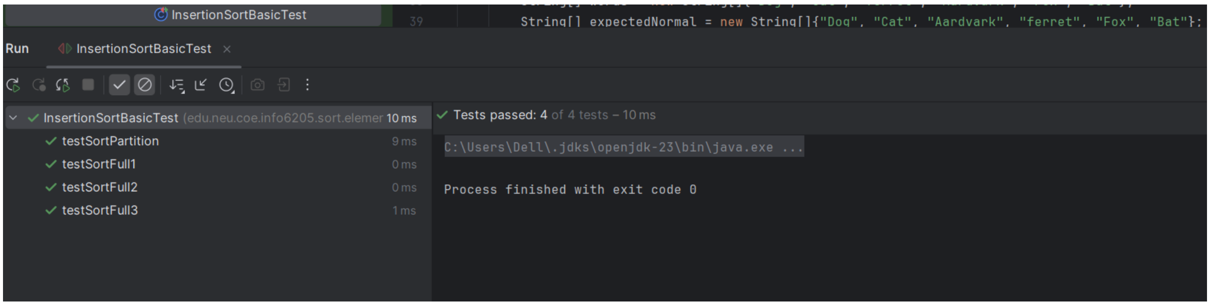
```
    private static double toMillisecs(long ticks) {  2 usages  Robin Hillyard +2

Run   BenchmarkTest

✓ BenchmarkTest (edu.neu.coe.info6205.util)   2 sec 926 ms
    ✓ testWaitPeriods   2 sec 925 ms
    ✓ getWarmupRuns     1 ms

✓ Tests passed: 2 of 2 tests – 2 sec 926 ms

C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
2024-10-07 20:53:30.926 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Task 2 – InsertionSortBasic.java



```java
public class InsertionSortBasic<S> {
     * @param from the first (left-most) element of the partition being sorted.
     * @param i    the index of the transitional element.
     * @param a    the (sorted) array into which the transitional element should be moved.
     */
    private void insert(int from, int i, S[] a) {
        // TO BE IMPLEMENTED  : implement inner loop of insertion sort using comparator
        S value = a[i];
        int j = i - 1;
        while (j >= from && comparator.compare(a[j], value) > 0) {
            a[j + 1] = a[j];
            j--;
        }
       a[j+1]=value;

        // END SOLUTION
    }

    private void swap(Object[] a, int j, int i) {
        Object temp = a[j];
        a[j] = a[i];
        a[i] = temp;
    }

    private final Comparator<S> comparator;
}
```

Output-UnitTest

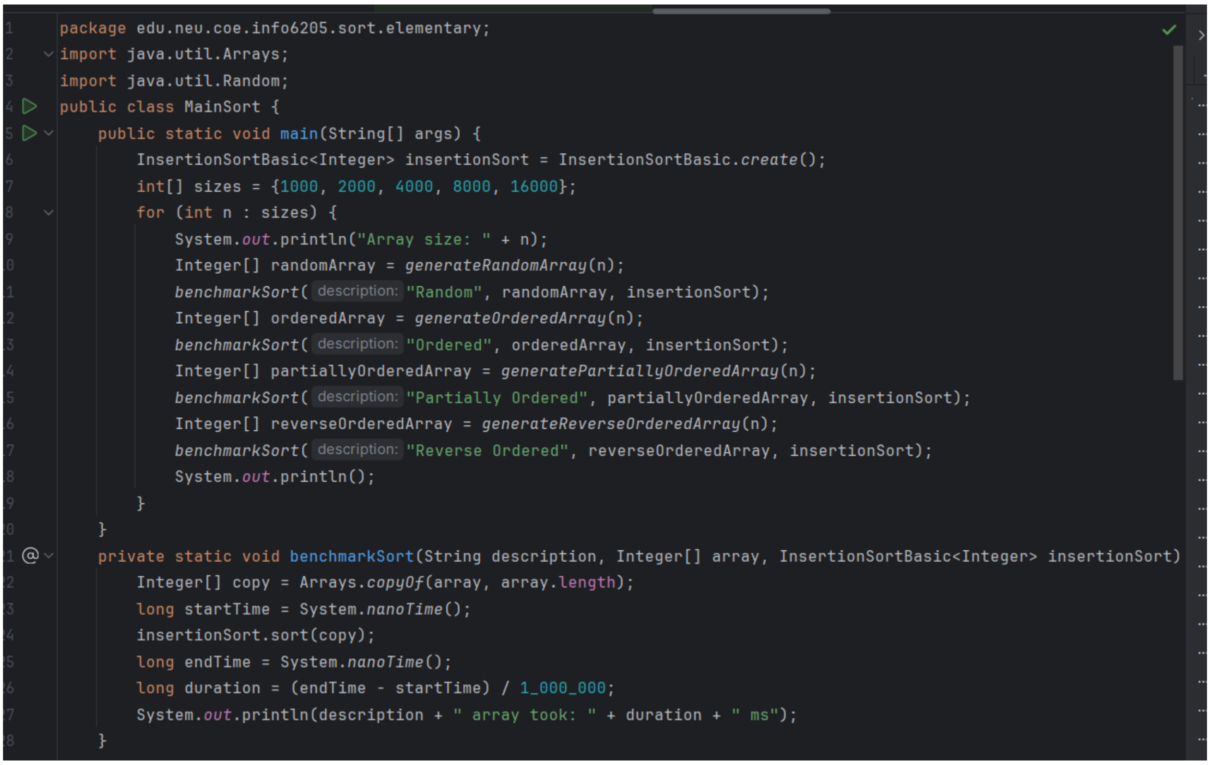


```java
String[] expectedNormal = new String[]{"Dog", "Cat", "Aardvark", "ferret", "Fox", "Bat"};
```

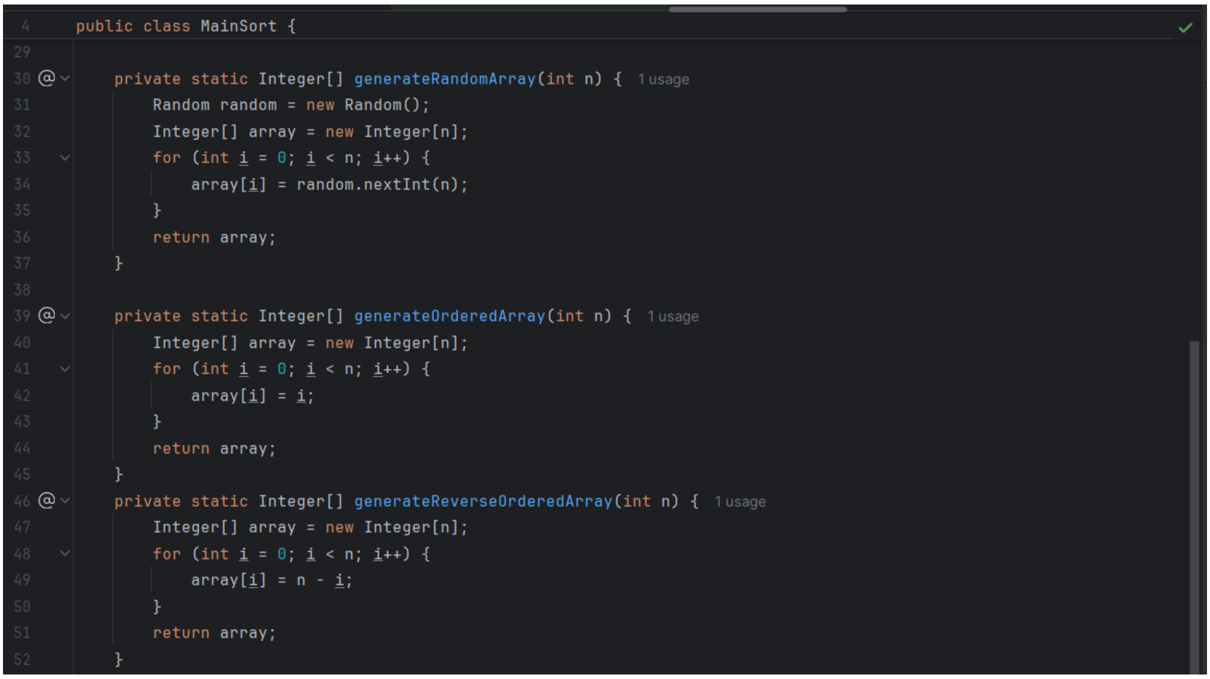
```
✓ InsertionSortBasicTest (edu.neu.coe.info6205.sort.elemer 10 ms
    ✓ testSortPartition    9 ms
    ✓ testSortFull1        0 ms
    ✓ testSortFull2        0 ms
    ✓ testSortFull3        1 ms
```

```
✓ Tests passed: 4 of 4 tests – 10 ms

C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...

Process finished with exit code 0
```

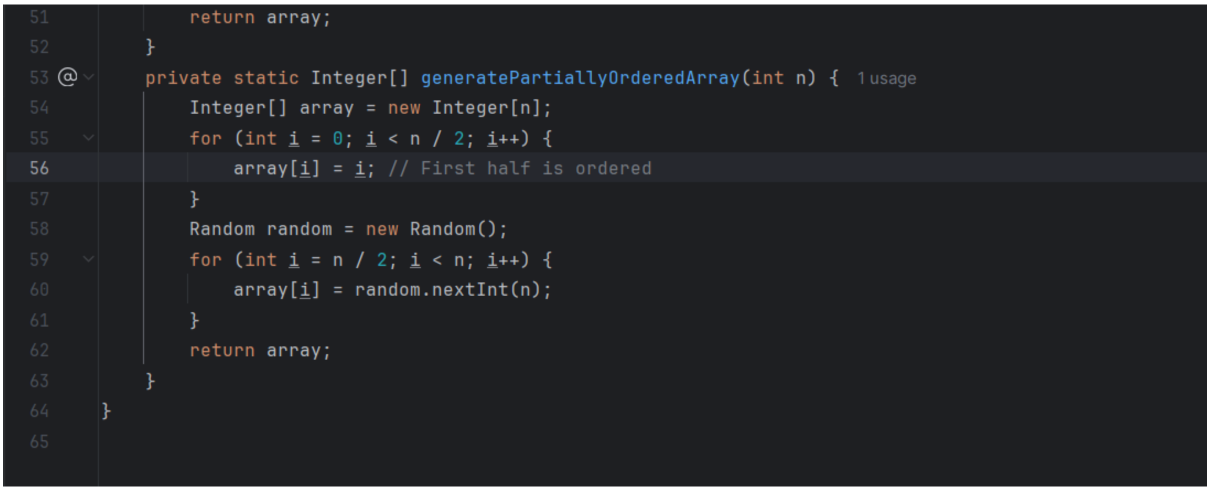
Task 3 – Main class implementation



```java
package edu.neu.coe.info6205.sort.elementary;
import java.util.Arrays;
import java.util.Random;
public class MainSort {
    public static void main(String[] args) {
        InsertionSortBasic<Integer> insertionSort = InsertionSortBasic.create();
        int[] sizes = {1000, 2000, 4000, 8000, 16000};
        for (int n : sizes) {
            System.out.println("Array size: " + n);
            Integer[] randomArray = generateRandomArray(n);
            benchmarkSort( description: "Random", randomArray, insertionSort);
            Integer[] orderedArray = generateOrderedArray(n);
            benchmarkSort( description: "Ordered", orderedArray, insertionSort);
            Integer[] partiallyOrderedArray = generatePartiallyOrderedArray(n);
            benchmarkSort( description: "Partially Ordered", partiallyOrderedArray, insertionSort);
            Integer[] reverseOrderedArray = generateReverseOrderedArray(n);
            benchmarkSort( description: "Reverse Ordered", reverseOrderedArray, insertionSort);
            System.out.println();
        }
    }
    private static void benchmarkSort(String description, Integer[] array, InsertionSortBasic<Integer> insertionSort)
        Integer[] copy = Arrays.copyOf(array, array.length);
        long startTime = System.nanoTime();
        insertionSort.sort(copy);
        long endTime = System.nanoTime();
        long duration = (endTime - startTime) / 1_000_000;
        System.out.println(description + " array took: " + duration + " ms");
    }
```

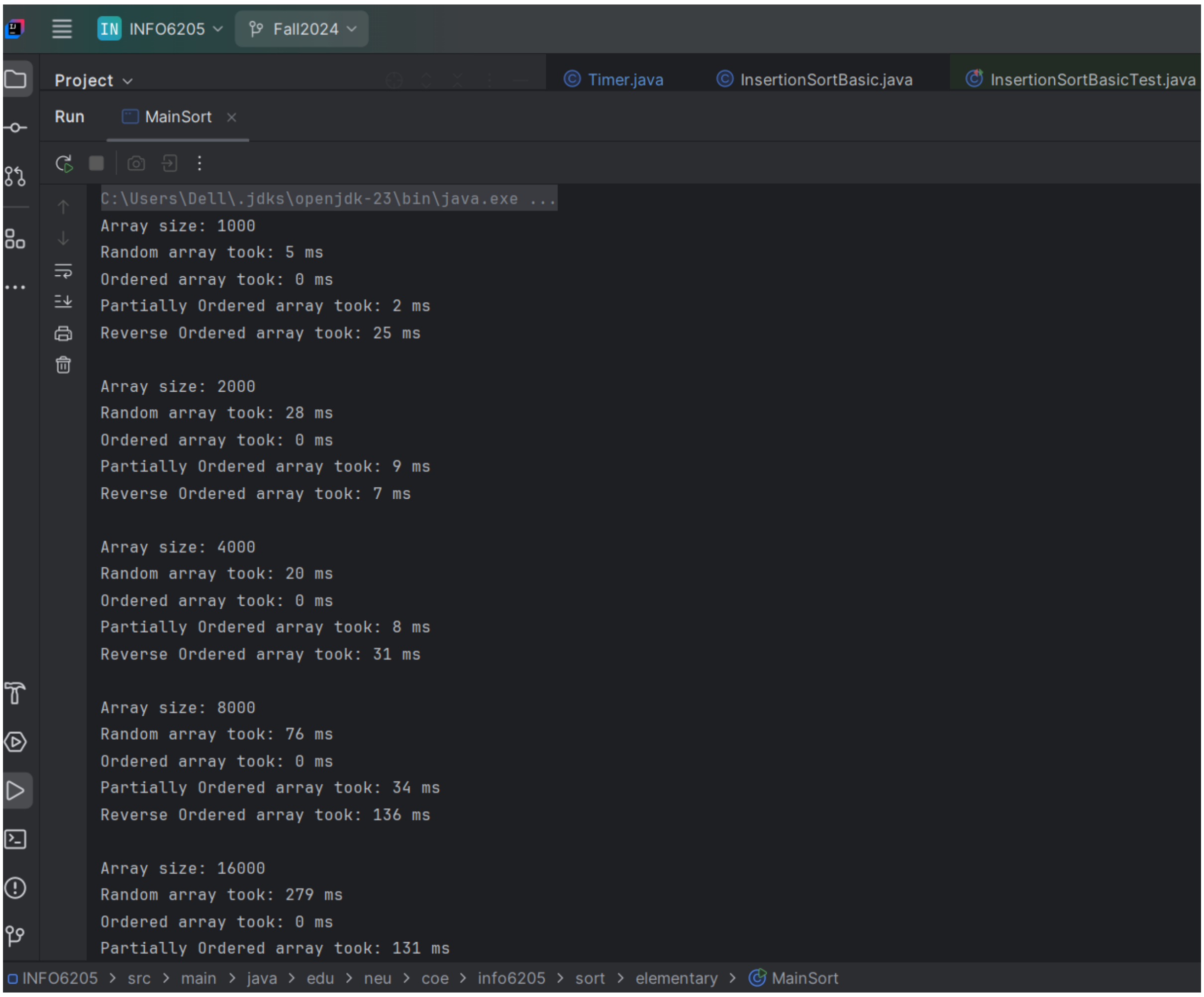


```java
public class MainSort {

    private static Integer[] generateRandomArray(int n) {  1 usage
        Random random = new Random();
        Integer[] array = new Integer[n];
        for (int i = 0; i < n; i++) {
            array[i] = random.nextInt(n);
        }
        return array;
    }

    private static Integer[] generateOrderedArray(int n) {  1 usage
        Integer[] array = new Integer[n];
        for (int i = 0; i < n; i++) {
            array[i] = i;
        }
        return array;
    }
    private static Integer[] generateReverseOrderedArray(int n) {  1 usage
        Integer[] array = new Integer[n];
        for (int i = 0; i < n; i++) {
            array[i] = n - i;
        }
        return array;
    }
```

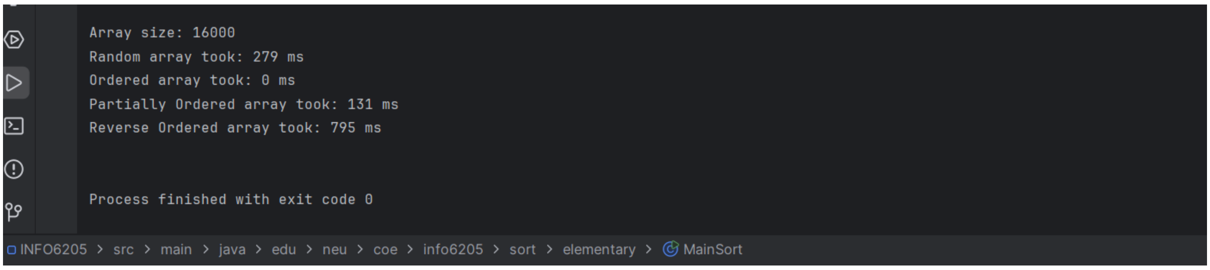


```java
        return array;
    }
    private static Integer[] generatePartiallyOrderedArray(int n) {  1 usage
        Integer[] array = new Integer[n];
        for (int i = 0; i < n / 2; i++) {
            array[i] = i; // First half is ordered
        }
        Random random = new Random();
        for (int i = n / 2; i < n; i++) {
            array[i] = random.nextInt(n);
        }
        return array;
    }
}

```

Output-


INFO6205
Fall2024
Project
Timer.java
InsertionSortBasic.java
InsertionSortBasicTest.java
Run
MainSort
INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > elementary > MainSort

```
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Array size: 1000
Random array took: 5 ms
Ordered array took: 0 ms
Partially Ordered array took: 2 ms
Reverse Ordered array took: 25 ms

Array size: 2000
Random array took: 28 ms
Ordered array took: 0 ms
Partially Ordered array took: 9 ms
Reverse Ordered array took: 7 ms

Array size: 4000
Random array took: 20 ms
Ordered array took: 0 ms
Partially Ordered array took: 8 ms
Reverse Ordered array took: 31 ms

Array size: 8000
Random array took: 76 ms
Ordered array took: 0 ms
Partially Ordered array took: 34 ms
Reverse Ordered array took: 136 ms

Array size: 16000
Random array took: 279 ms
Ordered array took: 0 ms
Partially Ordered array took: 131 ms
```


INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > elementary > MainSort

```
Array size: 16000
Random array took: 279 ms
Ordered array took: 0 ms
Partially Ordered array took: 131 ms
```

```
Reverse Ordered array took: 795 ms


Process finished with exit code 0
```

Conclusion-

1. Insertion sort is most efficient for already sorted arrays, and performance deteriorates significantly for reverse-ordered arrays, especially as size increases.

2. The performance on random and partially ordered arrays is generally better than on reverse-ordered arrays. Still, it scales poorly with larger input sizes, reflecting its O(n²) nature in less-than-optimal conditions.