

Assignment 3

Timer.java code changes

Repeat Function

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U>
function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: with " + n + " runs");
    T pre = null;
    // FIXME: note that the timer is running when this method is called and
    // should still be running when it returns. by replacing the following code
    for(int i = 0; i<n ; i++) {
        lap();
        pause();
        if(preFunction!= null) {
            pre = preFunction.apply(supplier.get());
        }
        resume();
        U fun = null;
        if(pre != null) {
            fun = function.apply(pre);
        }
        else {
            fun = function.apply(supplier.get());
        }
        pause();
        if(postFunction != null) {
            postFunction.accept(fun);
        }
        resume();
    }
    pause();
    return meanLapTime();
    // END
}
```

Get Clock Function

```
private static long getClock() {
    // FIXME by replacing the following code
    return System.nanoTime();
    // END
}
```

To Milli Secs Function

```
private static double toMillisecs(long ticks) {
```

```
// FIXME by replacing the following code
return TimeUnit.NANOSECONDS.toMillis(ticks);
// END
}
```

Unit test cases

Insertion.java

```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for(int i = from + 1; i < to; i++) {
        int k = i;
        while(k > from && helper.swapStableConditional(xs, k)) {
            k--;
        }
    }
    // FIXME
    // END
}
```

Unit test cases

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project structure with folders like 'graphs', 'greedy', 'lab_1', 'life', 'pq', 'randomwalk', 'reduction', 'sort', 'symbolTable', 'threesum', 'union_find', and 'util'. The 'util' folder is expanded, showing several test classes including 'TimerTest'.
- Editor:** Displays the source code for 'TimerTest.java'. The code includes package declarations, imports, and methods:


```
package edu.neu.coe.info6205.util;

import ...

public class TimerTest {

    @Before
    public void setup() {
        pre = 0;
        run = 0;
        post = 0;
        result = 0;
    }

    @Test
    public void testStop() {
        final Timer timer = new Timer();
    }
}
```
- Run Console:** Shows the execution results of the tests. It indicates that 11 out of 11 tests passed in 2 seconds and 628 milliseconds. The tests listed are:
 - testPauseAndLapResume0: 237 ms
 - testPauseAndLapResume1: 315 ms
 - testLap: 211 ms
 - testPause: 210 ms
 - testStop: 102 ms
 - testMillisecs: 105 ms
 - testRepeat1: 127 ms
 - testRepeat2: 249 ms
 - testRepeat3: 604 ms
 - testRepeat4: 361 ms
 - testPauseAndLap: 107 ms
- Status Bar:** Shows 'Tests passed: 11 (moments ago)' and 'Process finished with exit code 0'.

Different types of array filling logics:

Random Array:

```
public static void randomArray(Integer[] arr) {  
    for(int i=0;i<arr.length;i++) {  
        Random rand = new Random();  
        arr[i] = rand.nextInt(50);  
    }  
    arraySort(arr.length, arr);  
}
```

Ordered Array:

```
public static void orderedArray(Integer[] arr) {  
    for(int i=0;i<arr.length;i++) {  
        arr[i] = i;  
    }  
    arraySort(arr.length, arr);  
}
```

Reverse Ordered Array:

```
public static void reverseOrderedArray(Integer[] arr) {  
    for(int i=arr.length-1;i>=0;i--) {  
        arr[arr.length -1 - i] = i;  
    }  
    arraySort(arr.length, arr);  
}
```

Partially Ordered Array:

```
public static void partiallyOrderedArray(Integer[] arr) {  
  
    // Ordered  
    for(int i=0;i<arr.length/2;i++) {  
        arr[i] = i;  
    }  
  
    //Random  
    for(int i=arr.length/2;i<arr.length;i++) {  
        Random rand = new Random();  
        arr[i] = rand.nextInt(50);  
    }  
  
    arraySort(arr.length, arr);  
}
```

InsertionSortWithDifferentArrayValuesAndOrdering.java

```
package edu.neu.coe.info6205.sort.elementary;

import edu.neu.coe.info6205.sort.Helper;
import edu.neu.coe.info6205.sort.HelperFactory;
import edu.neu.coe.info6205.sort.SortWithHelper;
import edu.neu.coe.info6205.util.Benchmark_Timer;
import edu.neu.coe.info6205.util.Config;

import java.util.Random;
import java.util.function.Consumer;
import java.util.function.Supplier;

public class InsertionSortWithDifferentArrayValuesAndOrdering {

    public static void arraySort(int n, Integer[] a) {
        final Config config = Config.setupConfig("true", "0", "1", "", "");
        Helper<Integer> helper = HelperFactory.create("InsertionSort", n,
config);
        SortWithHelper<Integer> sorter = new InsertionSort<Integer>(helper);
        Integer[] ys = sorter.sort(a);
    }

    public static void main(String[] args) {
        int n = 6400;
        Integer[] arr = new Integer[n];

        // Reverse ordered array
        reverseOrderedArray(arr);

        // Random array
        randomArray(arr);

        // Partially ordered array
        partiallyOrderedArray(arr);

        // Ordered array
        orderedArray(arr);
    }

    public static void randomArray(Integer[] arr) {
        Consumer<Integer[]> randomFunc = randomOrderedArr ->
arraySort(arr.length, randomOrderedArr);
        Benchmark_Timer<Integer[]> randomOrderTimer = new
Benchmark_Timer<>("Sort random ordered array of " + arr.length + " elements",
randomFunc);
    }
}
```

```

Supplier<Integer[]> random = () -> {
    Integer[] randomArr = new Integer[arr.length];
    for(int i=0;i<arr.length;i++) {
        Random rand = new Random();
        randomArr[i] = rand.nextInt(arr.length);
    }
    return randomArr;
};
randomFunc.accept(random.get());
double randomTime = randomOrderTimer.run(random.get(), 100);
System.out.println("Time to run random array of " + arr.length + "
elements is " + randomTime);
}

public static void orderedArray(Integer[] arr) {
    Consumer<Integer[]> orderedFunc = orderedArr -> arraySort(arr.length,
orderedArr);
    Benchmark_Timer<Integer[]> orderedTimer = new Benchmark_Timer<>("Sort
ordered array of " + arr.length + " elements", orderedFunc);
    Supplier<Integer[]> ordered = () -> {
        Integer[] orderedArr = new Integer[arr.length];
        for(int i=0;i<arr.length;i++) {
            orderedArr[i] = i;
        }
        return orderedArr;
    };
    orderedFunc.accept(ordered.get());
    double orderedTime = orderedTimer.run(ordered.get(), 100);
    System.out.println("Time to run ordered array of " + arr.length + "
elements is " + orderedTime);
}

public static void reverseOrderedArray(Integer[] arr) {
    Consumer<Integer[]> reverseFunc = reverseArr -> arraySort(arr.length,
reverseArr);
    Benchmark_Timer<Integer[]> reverseTimer = new Benchmark_Timer<>("Sort
reverse array of " + arr.length + " elements", reverseFunc);
    Supplier<Integer[]> reverse = () -> {
        Integer[] reverseArr = new Integer[arr.length];
        for(int i=arr.length-1;i>=0;i--) {
            reverseArr[arr.length -1 - i] = i;
        }
        return reverseArr;
    };
    reverseFunc.accept(reverse.get());
    double reverseTime = reverseTimer.run(reverse.get(), 100);
    System.out.println("Time to run reverse array of " + arr.length + "
elements is " + reverseTime);
}

```

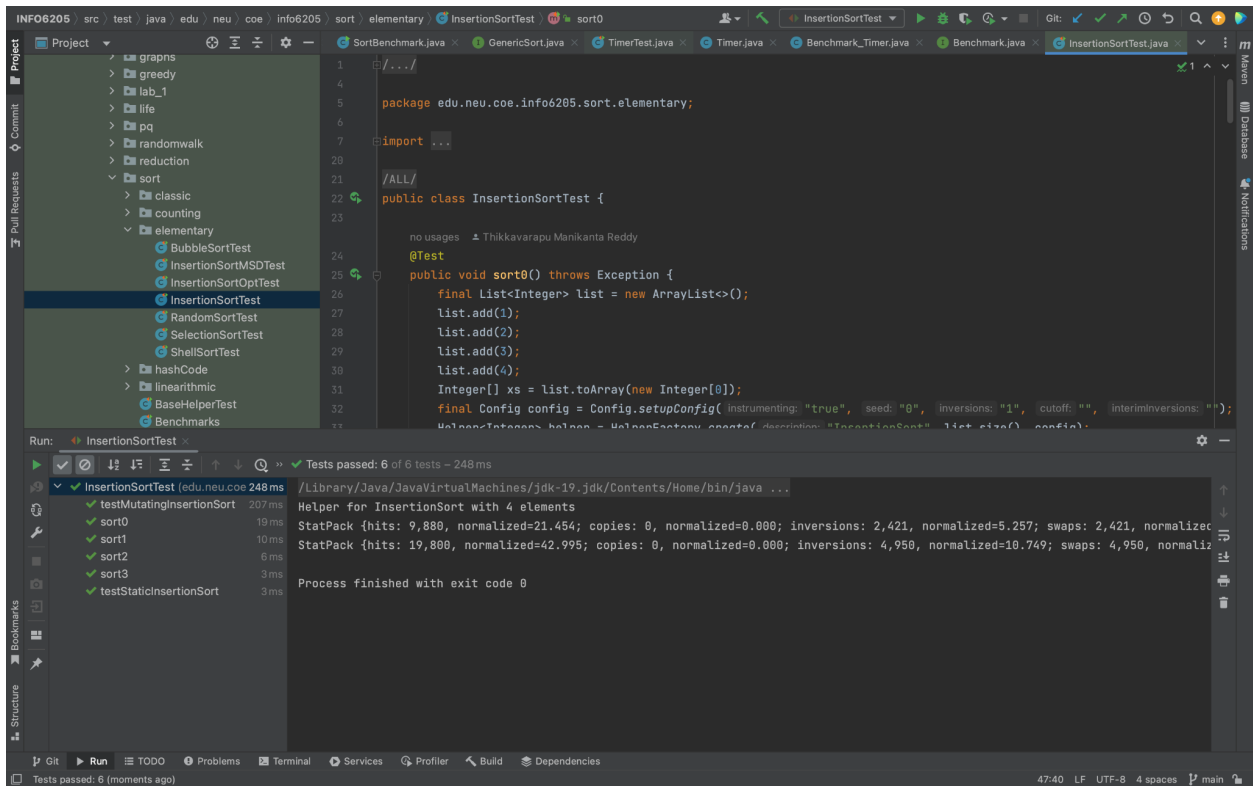
```

    public static void partiallyOrderedArray(Integer[] arr) {
        Consumer<Integer[]> partialOrderedFunc = partialOrderedArr ->
arraySort(arr.length, partialOrderedArr);
        Benchmark_Timer<Integer[]> partialOrderedTimer = new
Benchmark_Timer<>("Sort partial ordered array of " + arr.length + " elements",
partialOrderedFunc);
        Supplier<Integer[]> partialOrdered = () -> {
            Integer[] partialOrderedArr = new Integer[arr.length];
            // Ordered
            for(int i=0;i<arr.length/2;i++) {
                partialOrderedArr[i] = i;
            }

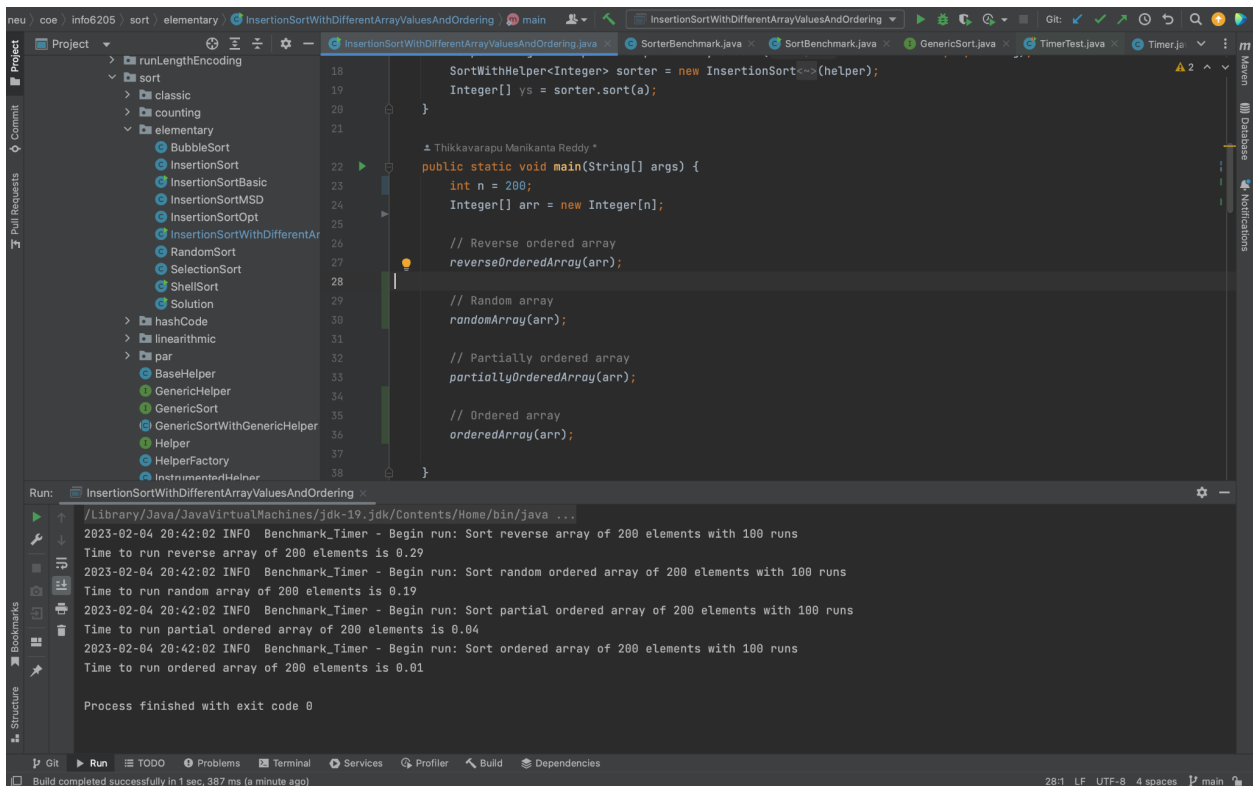
            //Random
            for(int i=arr.length/2;i<arr.length;i++) {
                Random rand = new Random();
                partialOrderedArr[i] = rand.nextInt(arr.length/2) +
arr.length/2;
            }
            return partialOrderedArr;
        };
        partialOrderedFunc.accept(partialOrdered.get());
        double partialOrderedTime =
partialOrderedTimer.run(partialOrdered.get(), 100);
        System.out.println("Time to run partial ordered array of " + arr.length
+ " elements is " + partialOrderedTime);
    }
}

```

Unit test cases



Output Screenshots



The screenshot shows an IDE with a project named 'elementary' containing a 'sort' package. The 'InsertionSortWithDifferentArrayValuesAndOrdering.java' file is open, showing a benchmarking class. The code defines a `SortWithHelper<Integer>` interface and implements it for InsertionSort. The `main` method tests the sort on arrays of size 400, including reverse, random, partially ordered, and ordered arrays, each with 100 runs.

```
SortWithHelper<Integer> sorter = new InsertionSort<>>(helper);
Integer[] ys = sorter.sort(a);

public static void main(String[] args) {
    int n = 400;
    Integer[] arr = new Integer[n];

    // Reverse ordered array
    reverseOrderedArray(arr);

    // Random array
    randomArray(arr);

    // Partially ordered array
    partiallyOrderedArray(arr);

    // Ordered array
    orderedArray(arr);
}
```

The Run console shows the following output:

```
2023-02-04 20:43:08 INFO Benchmark_Timer - Begin run: Sort reverse array of 400 elements with 100 runs
Time to run reverse array of 400 elements is 0.6
2023-02-04 20:43:08 INFO Benchmark_Timer - Begin run: Sort random ordered array of 400 elements with 100 runs
Time to run random array of 400 elements is 0.38
2023-02-04 20:43:08 INFO Benchmark_Timer - Begin run: Sort partial ordered array of 400 elements with 100 runs
Time to run partial ordered array of 400 elements is 0.13
2023-02-04 20:43:08 INFO Benchmark_Timer - Begin run: Sort ordered array of 400 elements with 100 runs
Time to run ordered array of 400 elements is 0.02
Process finished with exit code 0
```

The screenshot shows the same IDE setup as the first image, but with the array size `n` set to 800 in the `main` method. The benchmark results in the Run console are as follows:

```
2023-02-04 20:43:37 INFO Benchmark_Timer - Begin run: Sort reverse array of 800 elements with 100 runs
Time to run reverse array of 800 elements is 2.82
2023-02-04 20:43:37 INFO Benchmark_Timer - Begin run: Sort random ordered array of 800 elements with 100 runs
Time to run random array of 800 elements is 1.6
2023-02-04 20:43:37 INFO Benchmark_Timer - Begin run: Sort partial ordered array of 800 elements with 100 runs
Time to run partial ordered array of 800 elements is 0.39
2023-02-04 20:43:37 INFO Benchmark_Timer - Begin run: Sort ordered array of 800 elements with 100 runs
Time to run ordered array of 800 elements is 0.02
Process finished with exit code 0
```

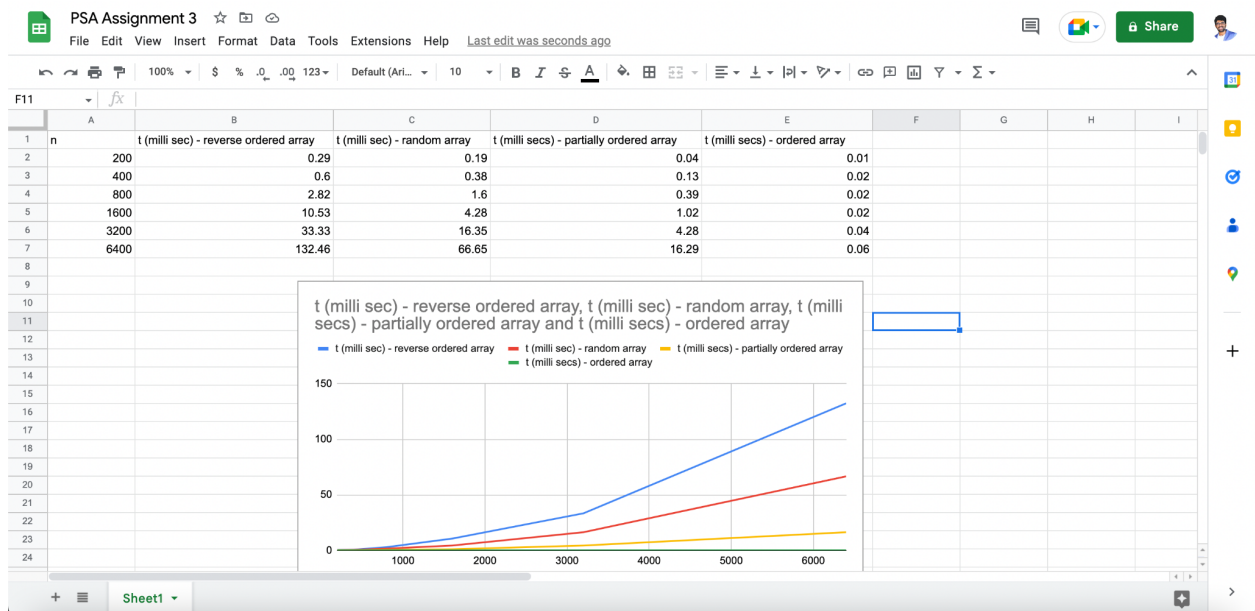


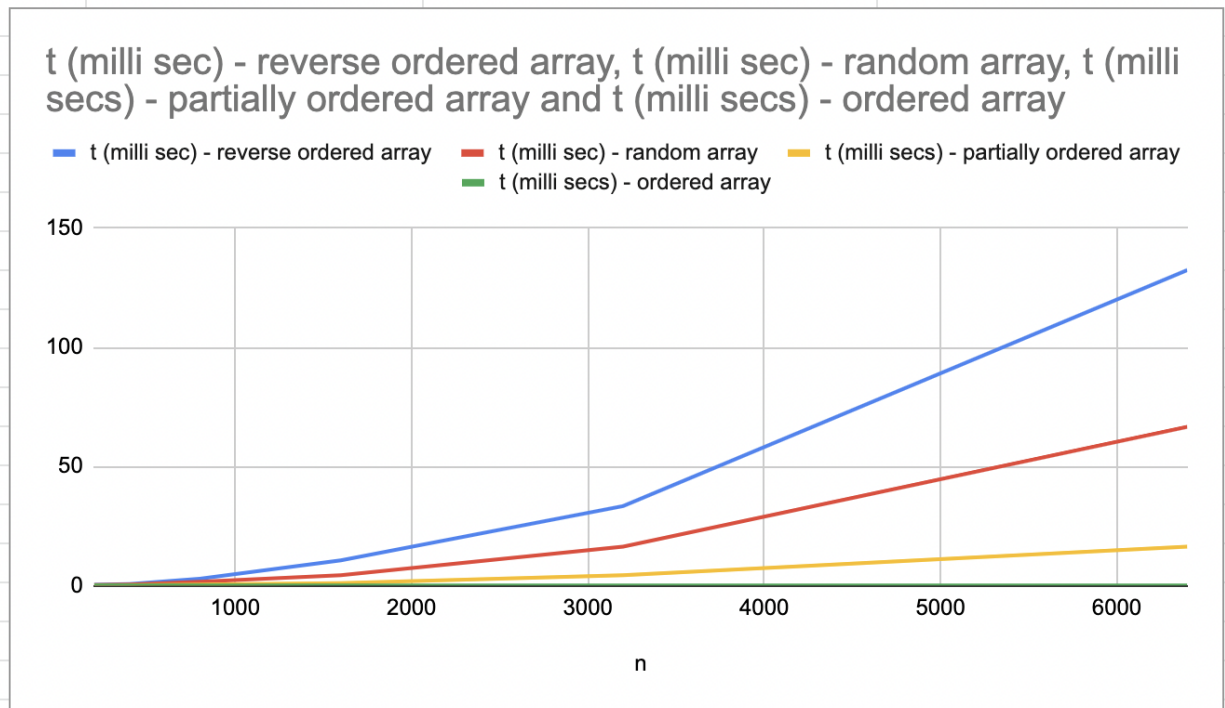
```
Project
├── runLengthEncoding
├── sort
│   ├── classic
│   ├── counting
│   └── elementary
│       ├── BubbleSort
│       ├── InsertionSort
│       ├── InsertionSortBasic
│       ├── InsertionSortMSD
│       ├── InsertionSortOpt
│       ├── InsertionSortWithDifferentAr
│       ├── RandomSort
│       ├── SelectionSort
│       ├── ShellSort
│       └── Solution
├── hashCode
├── linearithmic
├── par
│   ├── BaseHelper
│   ├── GenericHelper
│   ├── GenericSort
│   ├── GenericSortWithGenericHelper
│   ├── Helper
│   ├── HelperFactory
│   └── InstrumentedHelper
└── ...

InsertionSortWithDifferentArrayValuesAndOrdering.java
18 SortWithHelper<Integer> sorter = new InsertionSort<>(helper);
19 Integer[] ys = sorter.sort(a);
20 }
21
22 public static void main(String[] args) {
23     int n = 1600;
24     Integer[] arr = new Integer[n];
25
26     // Reverse ordered array
27     reverseOrderedArray(arr);
28
29     // Random array
30     randomArray(arr);
31
32     // Partially ordered array
33     partiallyOrderedArray(arr);
34
35     // Ordered array
36     orderedArray(arr);
37 }

Run: InsertionSortWithDifferentArrayValuesAndOrdering
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
2023-02-04 20:44:07 INFO Benchmark_Timer - Begin run: Sort reverse array of 1600 elements with 100 runs
Time to run reverse array of 1600 elements is 10.53
2023-02-04 20:44:08 INFO Benchmark_Timer - Begin run: Sort random ordered array of 1600 elements with 100 runs
Time to run random array of 1600 elements is 4.28
2023-02-04 20:44:09 INFO Benchmark_Timer - Begin run: Sort partial ordered array of 1600 elements with 100 runs
Time to run partial ordered array of 1600 elements is 1.02
2023-02-04 20:44:09 INFO Benchmark_Timer - Begin run: Sort ordered array of 1600 elements with 100 runs
Time to run ordered array of 1600 elements is 0.02
Process finished with exit code 0
```

Excel sheet screenshots





Observations:

- The graph shows that, for bigger array sizes, the reverse ordered array requires the longest time to sort using insertion sort since all of the elements must be moved to their proper locations.
- Since there is a good probability that some of the random numbers created by the random function contain some of the numbers already in the right location, random array takes somewhat less time than reverse ordered array.
- Since half of the array would already be in the sorted position, partially sorted order requires less time than the random array.
- The optimal scenario is an array that has already been sorted, which takes the shortest amount of time and is expected given that the array has already been sorted.