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) {
```

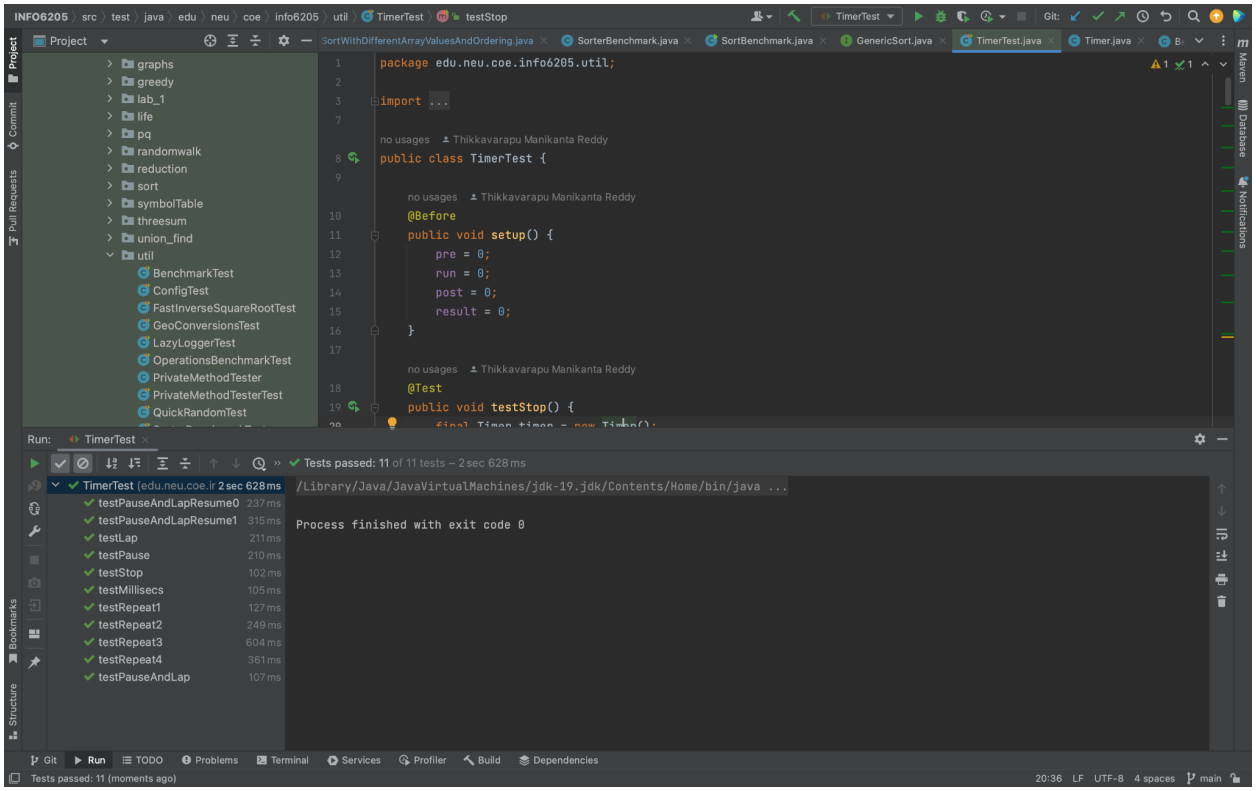
```
    // FIXME by replacing the following code
     return TimeUnit.NANOSECONDS.toMillis(ticks);
    // END
}
```

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



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);
}
```

Revere 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);
}
```

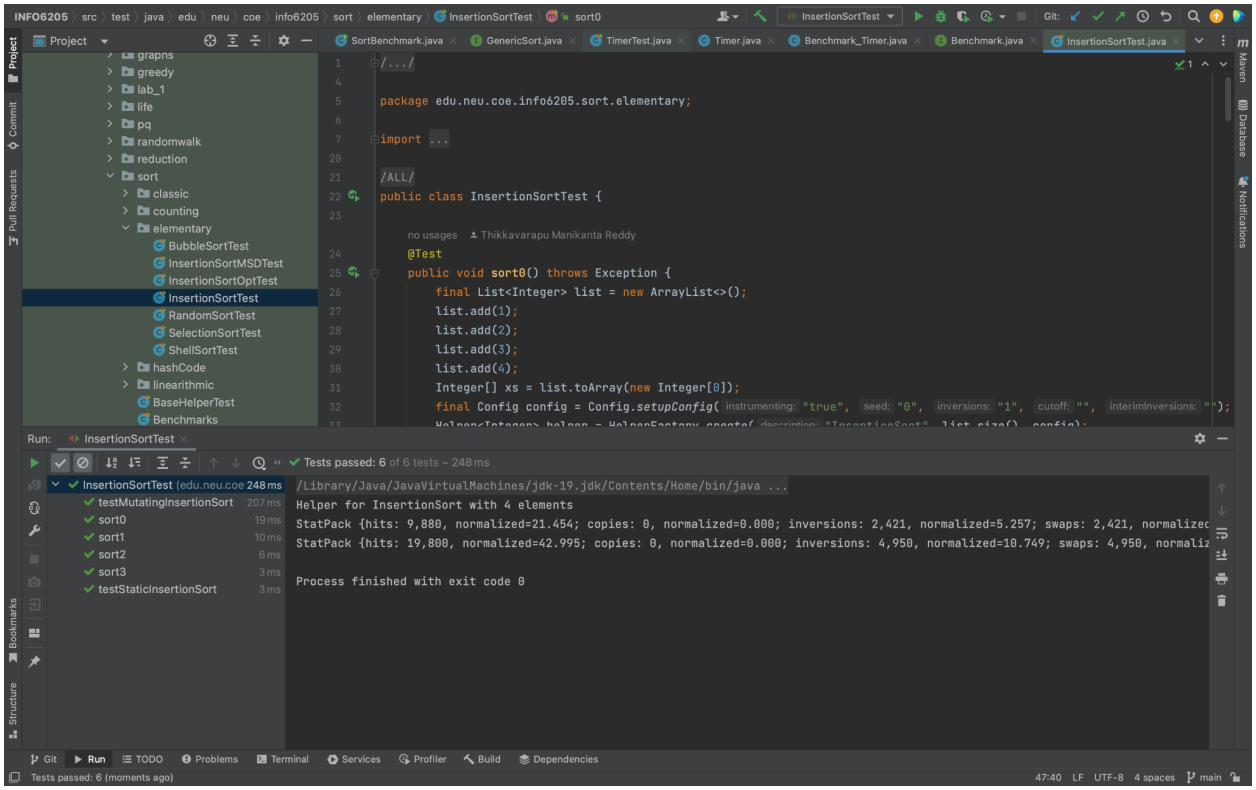
InsertionSortWithDifferentArrayValuesAndOrdering.java

```
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);
```

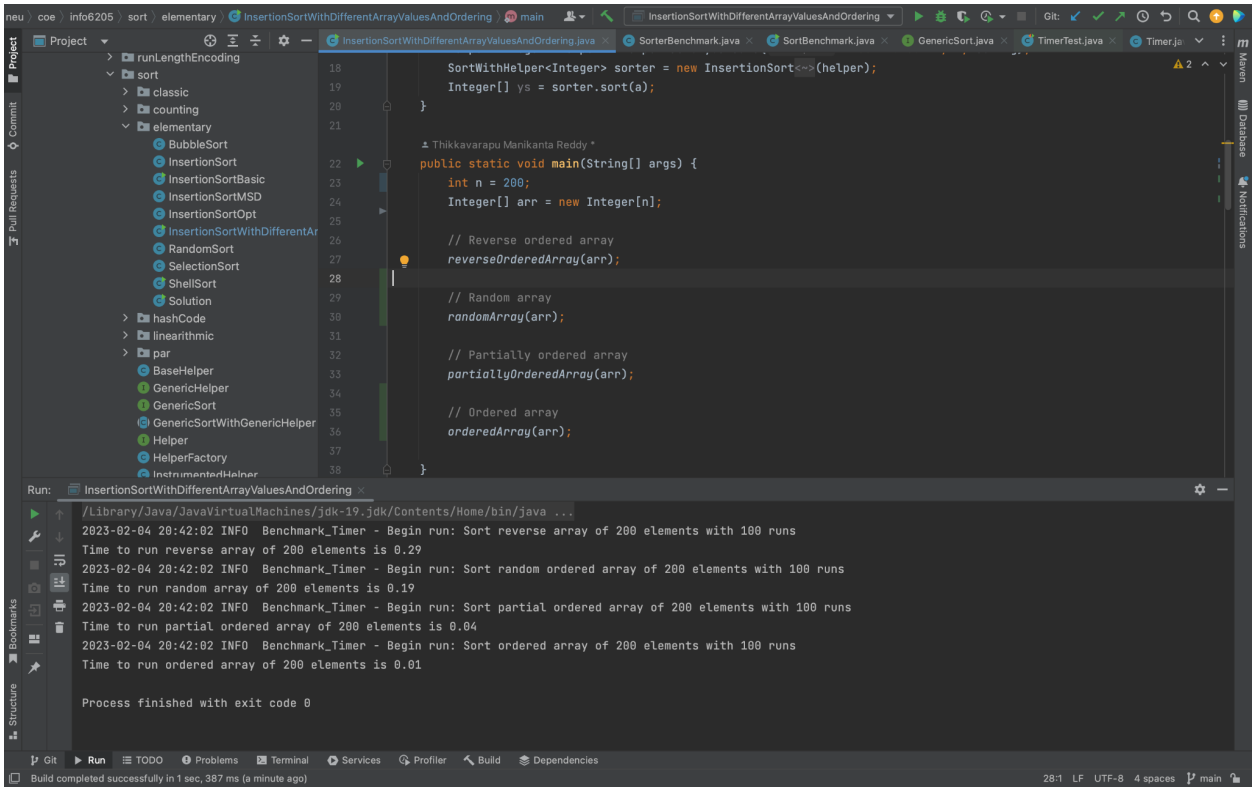
```
        Supplier<Integer[]> random = () -> {
            Integer[] randomArr = new Integer[arr.length];
            for(int i=0;i<arr.length;i++) {
                Random rand = new Random();
                randomArr[i] = rand.nextInt(arr.length);
            }
            return randomArr;
        };
        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);
    }
```

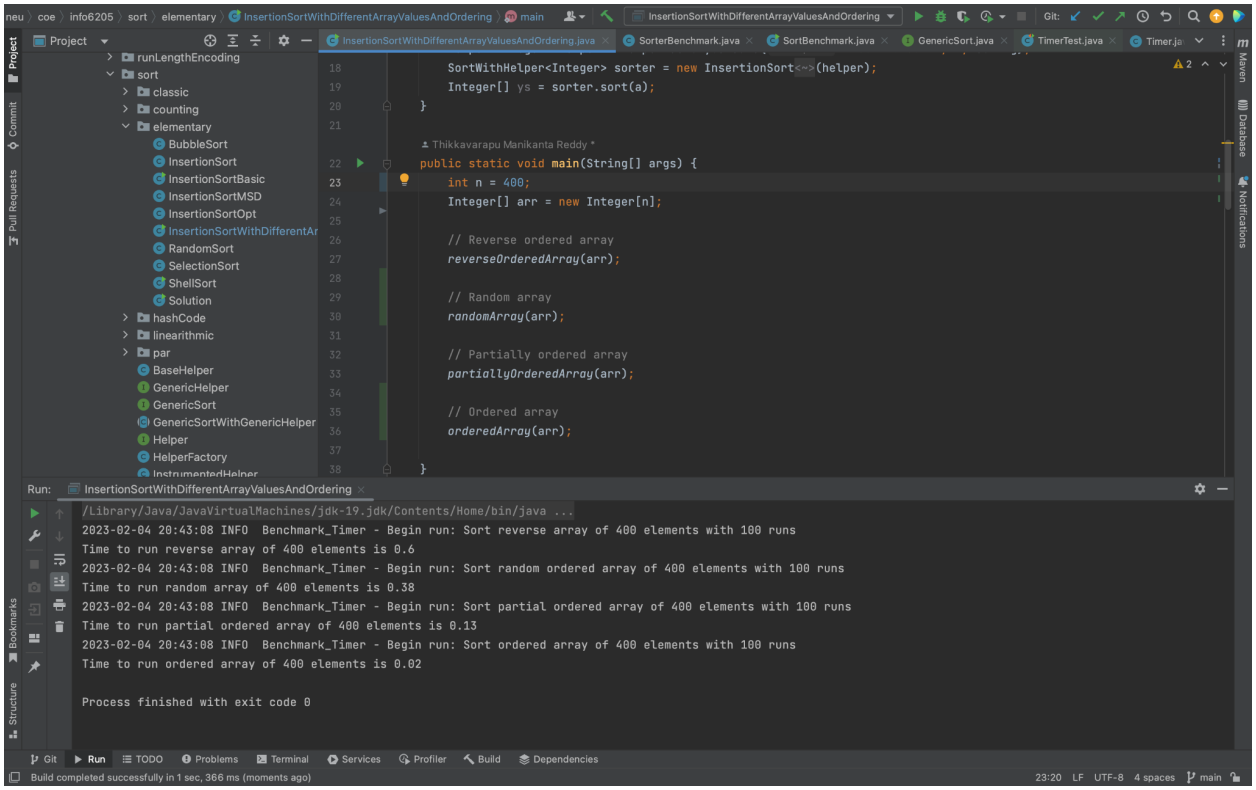
```
   public static void partiallyOrderedArray(Integer[] arr) {
       Consumer<Integer[]> partialOrderedFunc = partialOrderedArr ->
arraySort(arr.length, partialOrderedArr);
       Benchmark_Timer<Integer[]> partialOrderedTimer = new
Benchmark_Timer<>("Sort partial ordered array of " + arr.length + " 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);
   }
}
```

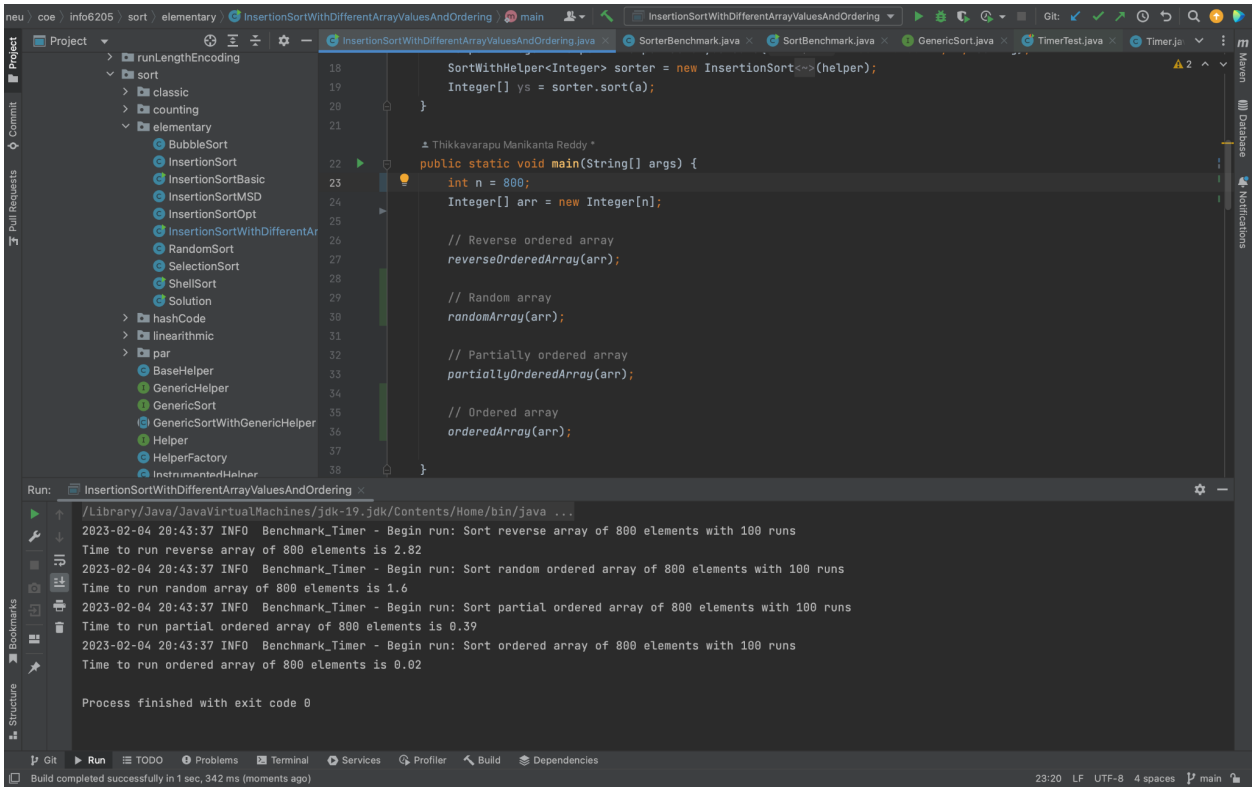
Unit test cases

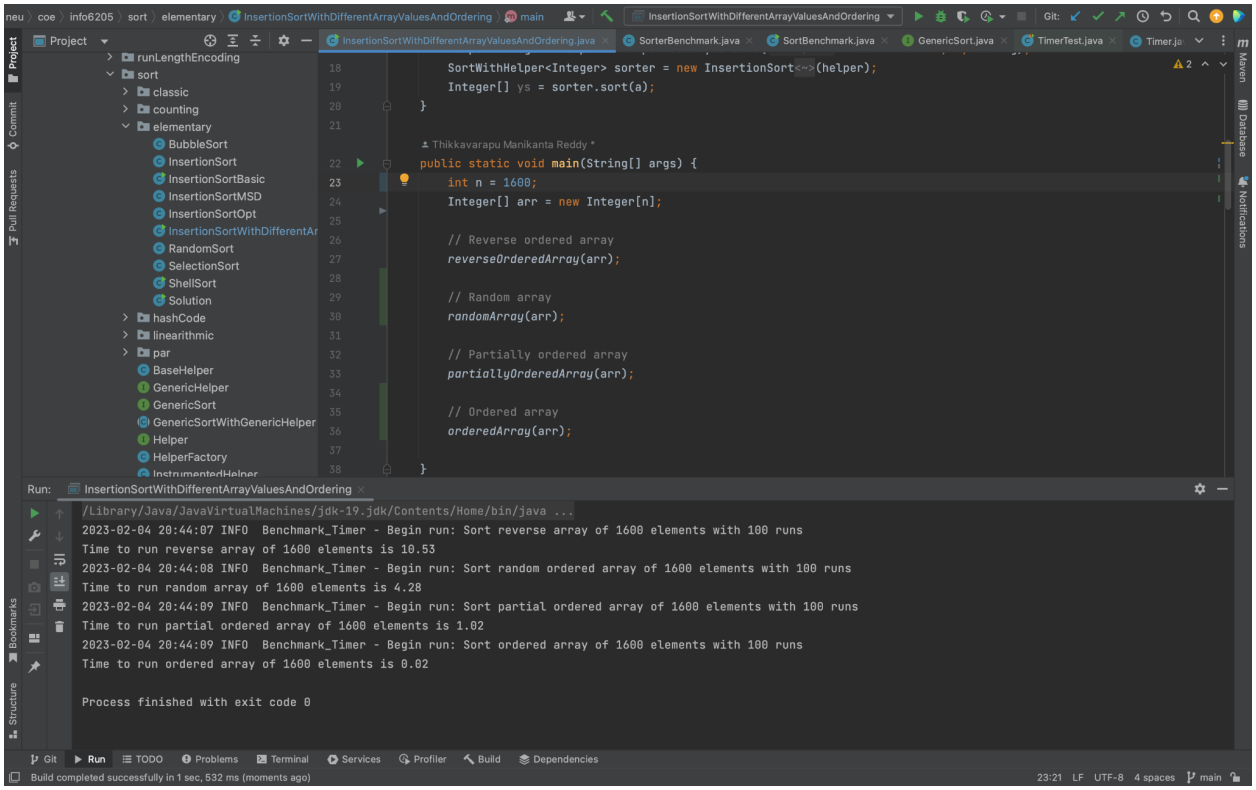


Output Screenshots

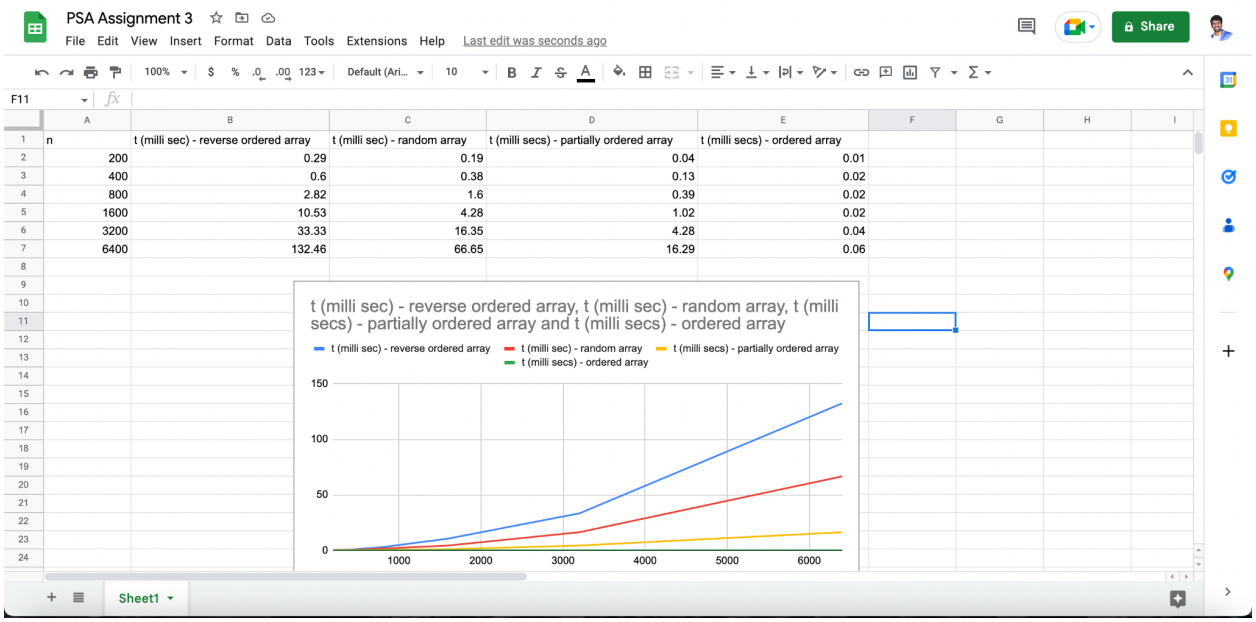


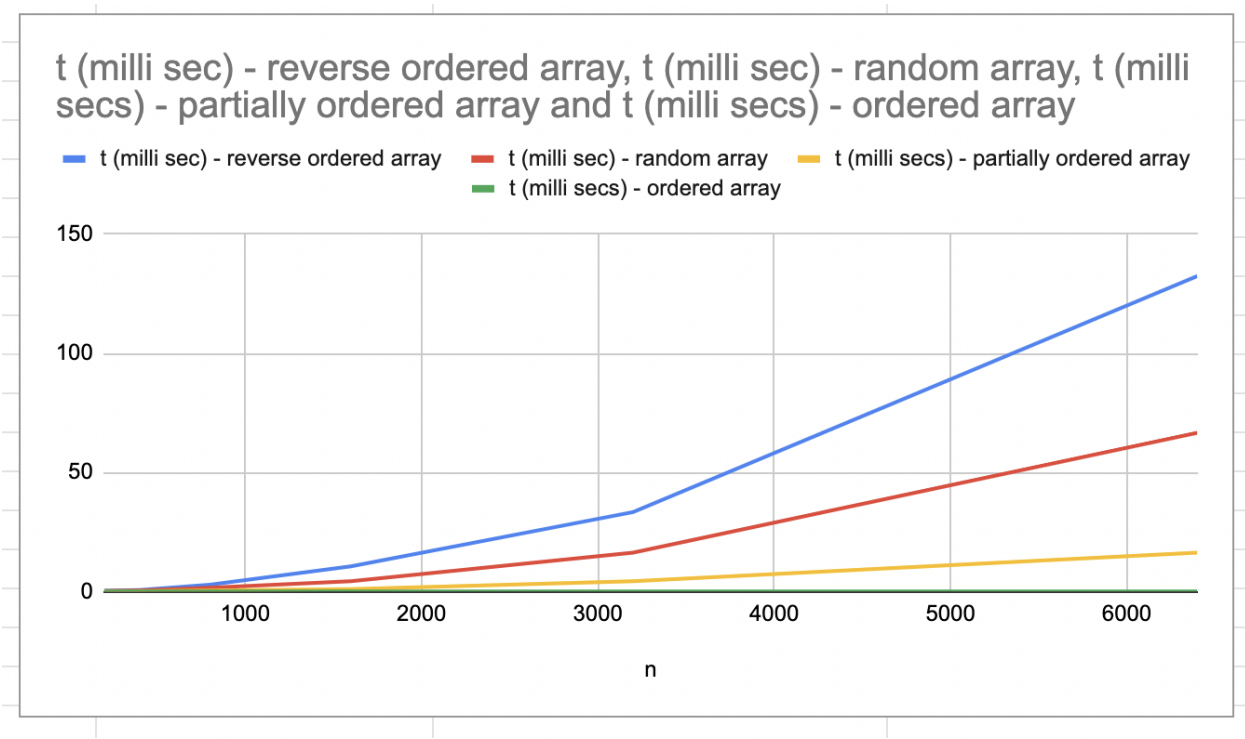

neu 〉 coe 〉 info6205 〉 sort 〉 elementary 〉 InsertionSortWithDifferentArrayValuesAndOrdering 〉 main
InsertionSortWithDifferentArrayValuesAndOrdering
Git:
Project
InsertionSortWithDifferentArrayValuesAndOrdering.java
SorterBenchmark.java
SortBenchmark.java
GenericSort.java
TimerTest.java
runLengthEncoding
sort
classic
counting
elementary
BubbleSort
InsertionSort
InsertionSortBasic
InsertionSortMSD
InsertionSortOpt
InsertionSortWithDifferentAr
RandomSort
SelectionSort
ShellSort
Solution
hashCode
linearithmic
par
BaseHelper
GenericHelper
GenericSort
GenericSortWithGenericHelper
Helper
HelperFactory
SortWithHelper<Integer> sorter = new InsertionSort<~>(helper);
Integer[] ys = sorter.sort(a);
}
Thikkavarapu Manikanta Reddy *
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);
}
Run: InsertionSortWithDifferentArrayValuesAndOrdering
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
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
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies
Build completed successfully in 1 sec, 366 ms (moments ago)
23:20 LF UTF-8 4 spaces main


neu 〉 coe 〉 info6205 〉 sort 〉 elementary 〉 InsertionSortWithDifferentArrayValuesAndOrdering 〉 main
InsertionSortWithDifferentArrayValuesAndOrdering
Git:
Project
InsertionSortWithDifferentArrayValuesAndOrdering.java
SorterBenchmark.java
SortBenchmark.java
GenericSort.java
TimerTest.java
runLengthEncoding
sort
classic
counting
elementary
BubbleSort
InsertionSort
InsertionSortBasic
InsertionSortMSD
InsertionSortOpt
InsertionSortWithDifferentAr
RandomSort
SelectionSort
ShellSort
Solution
hashCode
linearithmic
par
BaseHelper
GenericHelper
GenericSort
GenericSortWithGenericHelper
Helper
HelperFactory
SortWithHelper<Integer> sorter = new InsertionSort<~>(helper);
Integer[] ys = sorter.sort(a);
}
Thikkavarapu Manikanta Reddy *
public static void main(String[] args) {
int n = 800;
Integer[] arr = new Integer[n];
// Reverse ordered array
reverseOrderedArray(arr);
// Random array
randomArray(arr);
// Partially ordered array
partiallyOrderedArray(arr);
// Ordered array
orderedArray(arr);
}
Run: InsertionSortWithDifferentArrayValuesAndOrdering
/Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java ...
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
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies
Build completed successfully in 1 sec, 342 ms (moments ago)
23:20 LF UTF-8 4 spaces main



Excel sheet screenshots





Observations:

a) 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.
b) 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.
c) Since half of the array would already be in the sorted position, partially sorted order requires less time than the random array.
d) 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.