Ram Gopal Varma Alluri
002196886

INFO 6205 Program Structures and Algorithms
Assignment 2

Task 1:
Implementing three methods from class timer. Below is the source code .

```
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");
   pause();
   for(int i=0;i<n;i++) {
      if(preFunction!=null)  preFunction.apply(supplier.get());
      resume();
      U b = function.apply(supplier.get());
      pauseAndLap();
      if(postFunction!=null) postFunction.accept(b);
   }

   return meanLapTime();
}
private static long getClock() {

   return System.nanoTime();
}
private static double toMillisecs(long ticks) {

   return TimeUnit.NANOSECONDS.toMillis(ticks);
}
```

Task 2:
Implementing insertion sort algorithm using two different methods, the first method using helper class i.e., *instrument = true*. The second method using traditional insertion sort algorithm where *instrument != true* .Below is the source code

```
public void sort(X[] xs, int from, int to) {
   final Helper<X> helper = getHelper();
   if(helper.instrumented())
   {
      for(int i=from+1;i<to;i++)
      {
         for(int j=i;j>0 && helper.swapStableConditional(xs,j);j--);

      }
   }
   else
   {

      for (int i = from; i < to; i++) {
         int j=i;
         while ( j > 0   && (xs[j-1].compareTo(xs[j])) > 0 ) {
```

```
            X x = xs[j-1];
            xs[j-1] = xs[j];
            xs[j] = x;
            j--;
        }
    }
  }

}
```

Task 3:
Implementing a program 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. Below is the source code (Insertion Sort Class)

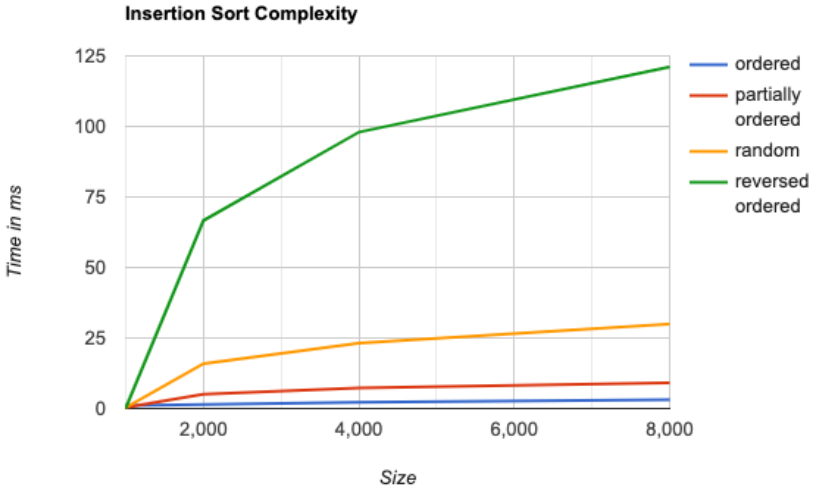
```
public static void main(String[] args) {

  StringBuilder output=new StringBuilder();
  output.append("SL.NO,")
        .append("Array Length(n),")
        .append("Ordered Array,")
        .append("Reverse Ordered Array,")
        .append("Partially Ordered Array,")
        .append("Random Array,").append("Mean Time")
        .append("\n");

  for (int t = 1; t < 5 ; t++) {

    final int n= (int) (Math.pow(2,t)*500);


    final Supplier<Integer[]> intsSupplierOrdered = () -> {
      Integer [] result = (Integer[]) Array.newInstance(Integer.class, n);
      for (int i = 0; i < n; i++) result[i] = i*100;
      return result;
    };

    final Supplier<Integer[]> intsSupplierReversed = () -> {
      Integer [] result2 = (Integer[]) Array.newInstance(Integer.class, n);
      for (int i = 0; i < n; i++) result2[i] = (n-i)*100;
      return result2;
    };

    final Supplier<Integer[]> intsSupplierPartial = () -> {
      Integer [] result3 = (Integer[]) Array.newInstance(Integer.class, n);
      for (int i = 0; i < n/2; i++) result3[i]=i*100 ;
      for(int i=(n/2);i<n;i++) result3[i]=n-i*100 ;
      return result3;
    };

    final Supplier<Integer[]> intsSupplierRandom = () -> {
      Integer [] result4 = (Integer[]) Array.newInstance(Integer.class, n);
      for (int i = 0; i < n; i++) result4[i]=(int)(Math.random()*100);
      return result4;
    };
```

```
        final double timeForOrdered = new Benchmark_Timer<Integer[]>(
                "intArraysorter",
               null,
                x->new InsertionSort<Integer>().sort(x,0,x.length-1),
                null
        ).runFromSupplier(intsSupplierOrdered, 30);

        final double timeForReversed = new Benchmark_Timer<Integer[]>(
                "intArraysorter",
                null,
                x->new InsertionSort<Integer>().sort(x,0,x.length-1),
                null
        ).runFromSupplier(intsSupplierReversed, 30);

        final double timeForPartial = new Benchmark_Timer<Integer[]>(
                "intArraysorter",
                null,
                x->new InsertionSort<Integer>().sort(x,0,x.length-1),
                null
        ).runFromSupplier(intsSupplierPartial, 30);

        final double timeForRandom = new Benchmark_Timer<Integer[]>(
                "intArraysorter",
                null,
                x->new InsertionSort<Integer>().sort(x,0,x.length-1),
                null
        ).runFromSupplier(intsSupplierRandom, 30);

        double meanTime=(timeForOrdered+timeForReversed+timeForPartial+timeForRandom)/4;

        output.append(t).append(",")
                .append(n).append(",").
                append(timeForOrdered).append(",")
                .append(timeForReversed).append(",")
                .append(timeForPartial).append(",")
                .append(timeForRandom).append(",")
                .append(meanTime).append(",")
                .append("\n");
        System.out.println("ordered "+timeForOrdered);
        System.out.println("reversed "+timeForReversed);
        System.out.println("partial "+timeForPartial);
        System.out.println("random "+timeForRandom);
        System.out.println("mean time "+ meanTime);


    }

    try {
        PrintWriter writer = new PrintWriter("./src/main/java/edu/neu/coe/info6205/sort/elementary/insertion-sort-
benchmark.csv");
        writer.write(output.toString());
        writer.close();
    }catch (FileNotFoundException e){
        e.printStackTrace();
    }

}
```

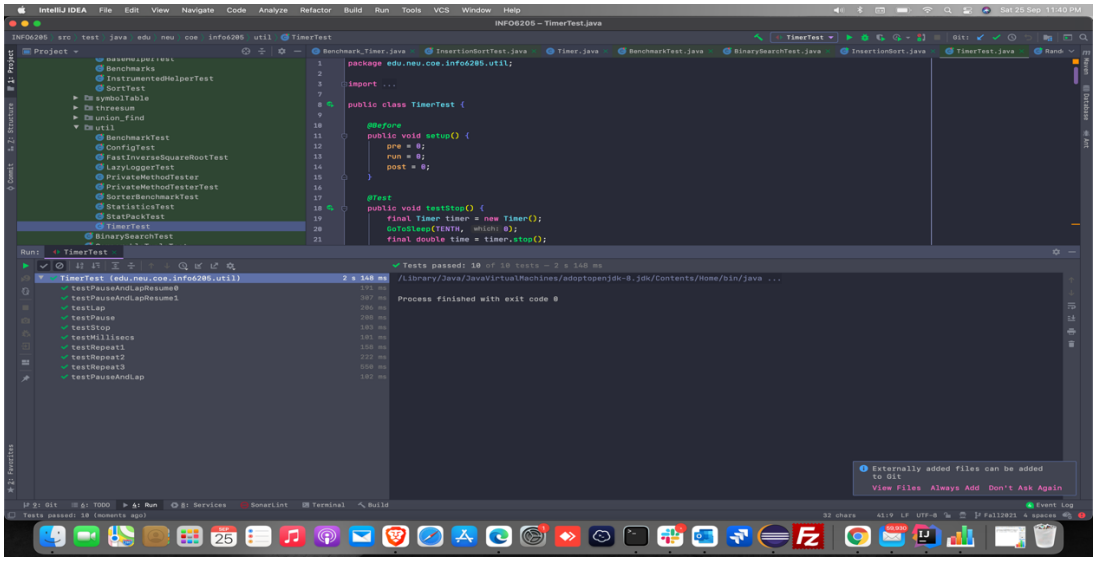
Conclusion:

I conclude that when arrays with different sizes and orders are input into the algorithm, the time cost will have a linear growth as array size increases. Sorting the ordered array takes the least time when compared to other arrays

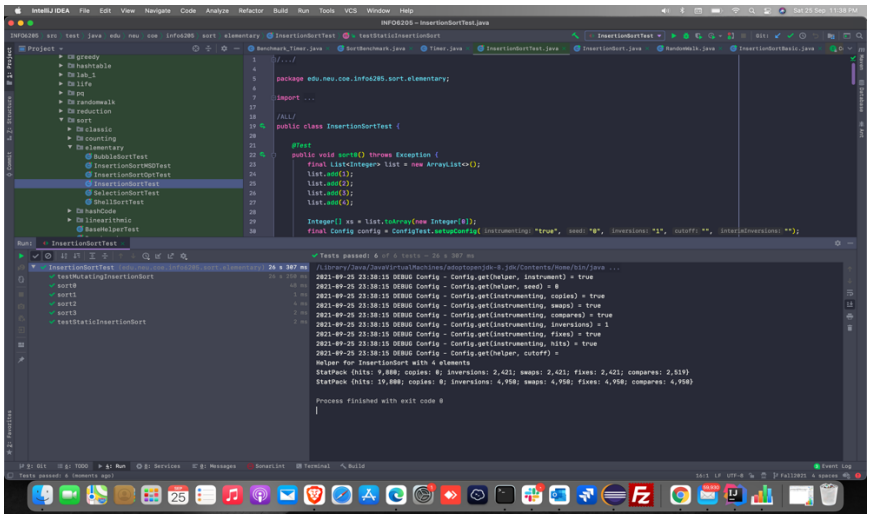
SL.NO	Array Length(n)	Ordered Array (ms)	Reverse Ordered Array (ms)	Partially Ordered Array (ms)	Random Array (ms)	Mean Time (ms)	
1	1000	0.7333333333333330	2.566666666666670	2.2333333333333300	1.3	1.7083333333333300	
2	2000	0.3333333333333330	9.833333333333330	8.333333333333330	4.066666666666670	5.641666666666670	
3	4000	0.3333333333333330	33.53333333333330	27.033333333333300	17.3	19.55	
4	8000	0.23333333333333300	131.36666666666700	92.26666666666670	57.733333333333300	70.4	



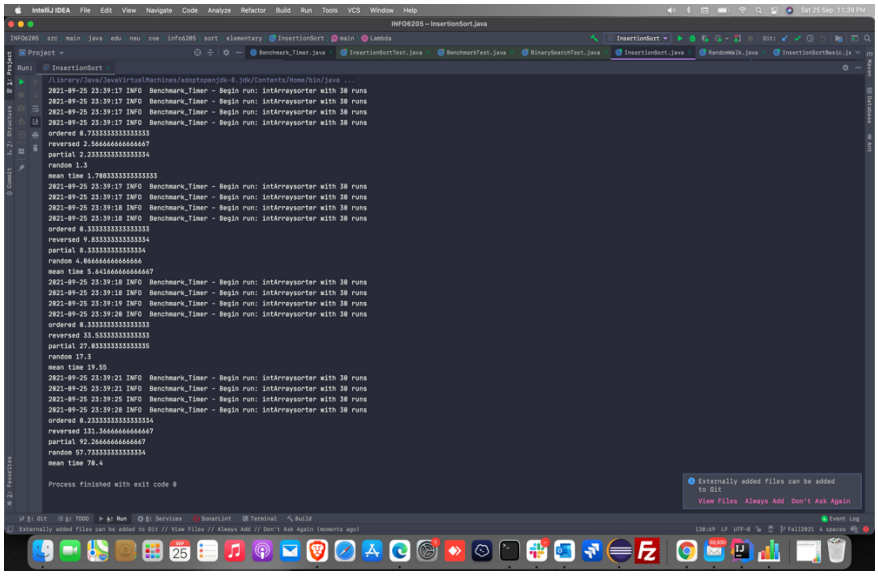
Test cases:
For task 1



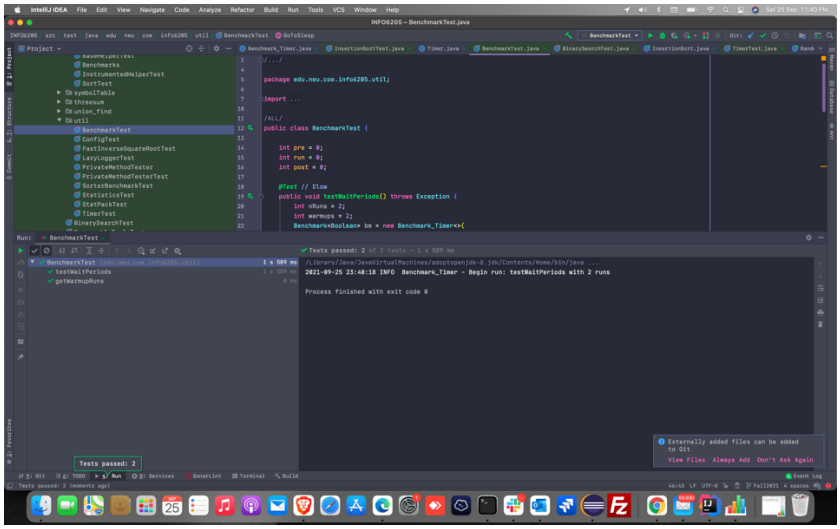
For task 2



For task 3



Benchmark test



Source Code (Link)