CS 452 - Design and Analysis of Algorithms

Homework 3

John Unger

This lab adheres to the JMU honor code,
a copy of which can be accessed here:
http://www.jmu.edu/honor/code.html

Task 1

The programming language I have chosen to implement the sort algorithm is Java. In order, to measure CPU computation time, a regular Nano-second timer would measure the real-world time that had passed for the process to finish. However, in modern computers and systems there is more than one process running, and the CPU scheduler switches in between processes. Therefore, a ThreadMXBean object will be used to measure the thread's own CPU timer, giving a more precise time where in which only the thread has been active. A time stamp can be made using this:

```
ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();
long time = threadMXBean.getCurrentThreadCpuTime();
```

To find the time that was needed to sort the array, one simply needs to keep record of when it started sorting and when it finished, the difference will be the total amount of time needed to compute.

Task 2

The sort algorithm used to make this happen was Insertion Sort (a copy of its implementation has been included in the appendix of this document). The original insertion algorithm has been pulled from http://www.java2novice.com/java-interview-programs/insertion-sort/ and is edited per the needs that were presented for this assignment.

Task 3

The sorting algorithm (as described in Task 2), is basically going through the array and puts each successive element in whichever position that is relatively sorted thus far. I filled the array with random numbers (used modulo to keep the values distributed and in range), and sorted it, in which different sizes and values were used to test the algorithm. To be sure to have a more consistent time the process of testing multiple sizes, was done multiple times, as seen in the table.

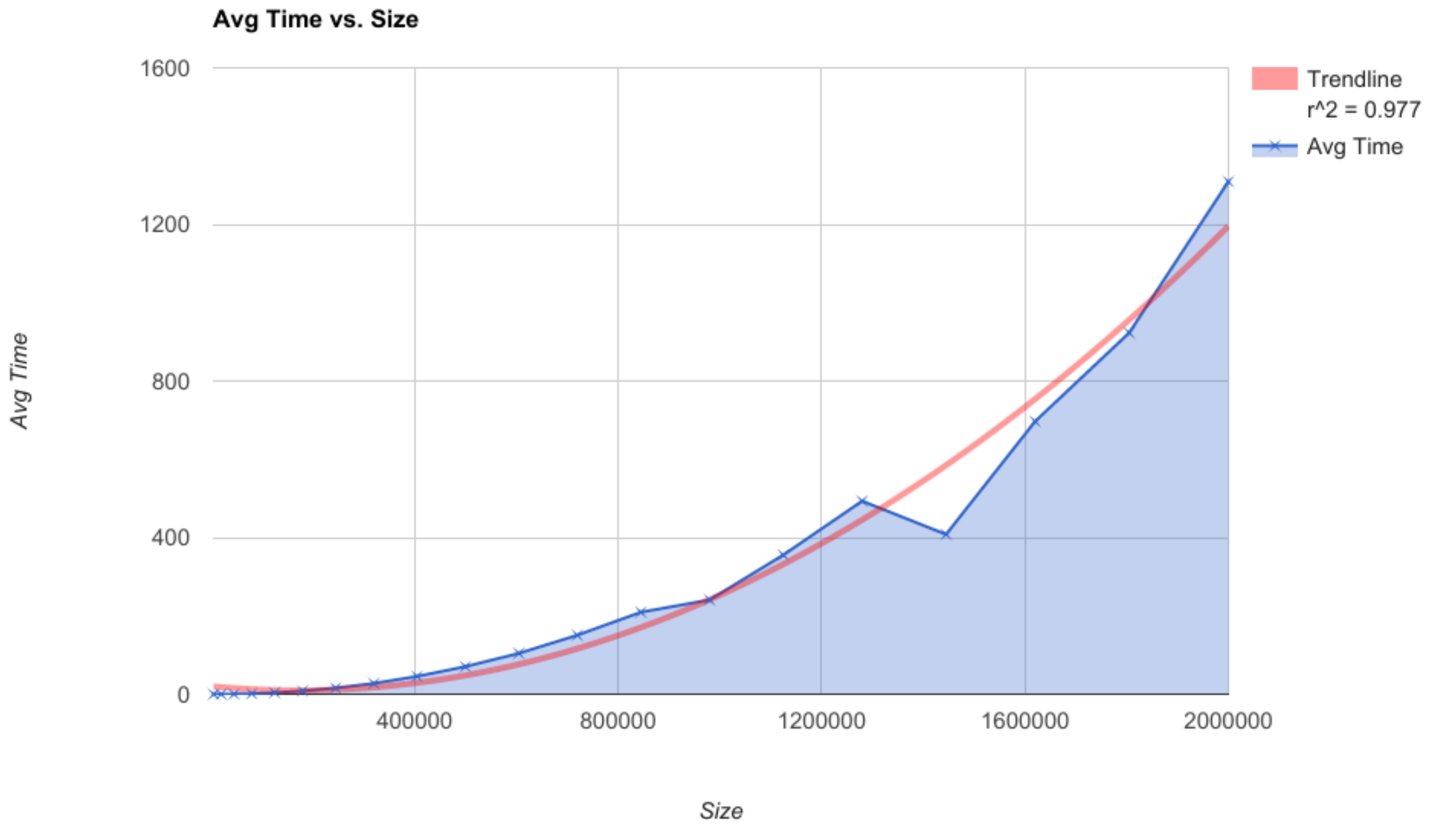
Instance	**5000 (size)**	**20000**	**45000**	**80000**	**125000**	**180000**	**245000**	**320000**
1	0.01472	0.084193	0.380809	1.524866	3.889905	8.351781	15.794015	27.669268
2	0.030751	0.085928	0.387093	1.5221	3.869203	8.349863	15.848817	27.581617
3	0.034102	0.086045	0.37819	1.523356	3.890117	8.342456	15.83253	27.622285
4	0.041713	0.087069	0.383507	1.527765	3.876895	8.363688	15.787309	27.629251
5	0.03404	0.084959	0.454735	1.532698	3.889057	8.368896	15.841139	27.689176
6	0.03996	0.082716	0.457082	1.533402	3.886221	8.305845	15.840046	27.69265
7	0.026151	0.084241	0.450036	1.53245	3.888039	8.340484	15.833557	27.665364
8	0.040555	0.083583	0.452631	1.524659	3.905434	8.365778	15.81111	27.597543
9	0.028036	0.082184	0.4497	1.527235	3.88082	8.340064	15.855658	27.679146
10	0.035409	0.084089	0.453611	1.5282	3.884488	8.328665	15.828791	27.625198
11	0.031951	0.083688	0.455168	1.528422	3.893768	8.34776	15.849594	27.649164
12	0.040109	0.084102	0.450146	1.535674	3.899643	8.317984	15.840473	27.658583
Avg	0.03312475	0.08439975	0.4293923333	1.52840225	3.887799167	8.343605333	15.83025325	27.64660375

Instance	**405000 (size)**	**500000**	**605000**	**720000**	**845000**
1	45.488014	70.473321	104.269719	150.597594	209.315346
2	45.490746	70.531358	104.2237	150.742122	209.511605
3	45.513529	70.45612	104.373716	150.578994	209.504775
4	45.397277	70.399463	104.09854	150.698956	209.206329
5	45.470806	70.373462	104.309464	150.663185	209.609808
6	45.478619	70.501453	104.364546	150.67057	209.216278
7	45.54006	70.406284	104.294528	150.58429	209.462535
8	45.510294	70.456065	104.339475	150.78532	209.348747
9	45.382813	70.28566	104.273304	150.767509	209.37667
10	45.458131	70.437868	104.426486	150.780529	209.459994
11	45.502447	70.356132	104.277441	150.548126	209.527239
12	45.439696	70.353485	104.274299	150.655867	209.355221
Avg	45.47270267	70.41922258	104.2937682	150.6727552	209.4078789

Instance	**980,000 (size)**	**1,125,000**	**1,280,000**	**1,445,000**
1	109.586517	374.597594	493.592465	275.949223
2	283.771492	374.336363	493.116337	275.428897
3	283.723459	374.425993	493.416815	275.642246
4	112.101999	374.385328	493.314432	297.711022
5	113.632321	374.335139	493.447093	528.829796
6	283.831877	374.572315	493.504957	297.775348
7	283.772061	374.86793	493.094022	297.584309
8	283.868203	374.654406	493.299233	529.148011
9	283.678939	146.964479	493.226353	530.004269
10	283.712346	374.653822	493.63295	529.452317
11	283.663205	374.613862	493.152996	530.310207
12	283.885762	374.445491	492.97431	529.288198
Avg	240.7690151	355.5710602	493.3143303	408.0936536

Instance	**1,620,000 (size)**	**1,805,000**	**2,000,000**
1	379.16978	922.841337	1308.783165
2	379.63539	923.493107	1309.584447
3	379.70364	923.358657	1309.71377
4	802.558399	922.909581	1309.172039
5	802.685682	923.387859	1309.350237
6	802.528304	923.460683	1309.624584
7	802.179789	922.549914	1309.51932
8	801.995942	922.670037	1309.341465
9	802.554277	922.183683	1308.43094
10	802.251417	923.377128	1309.474821
11	802.731532	923.508074	1309.120721
12	802.249381	922.253209	1308.775142
Avg	696.6869611	922.9994391	1309.240888

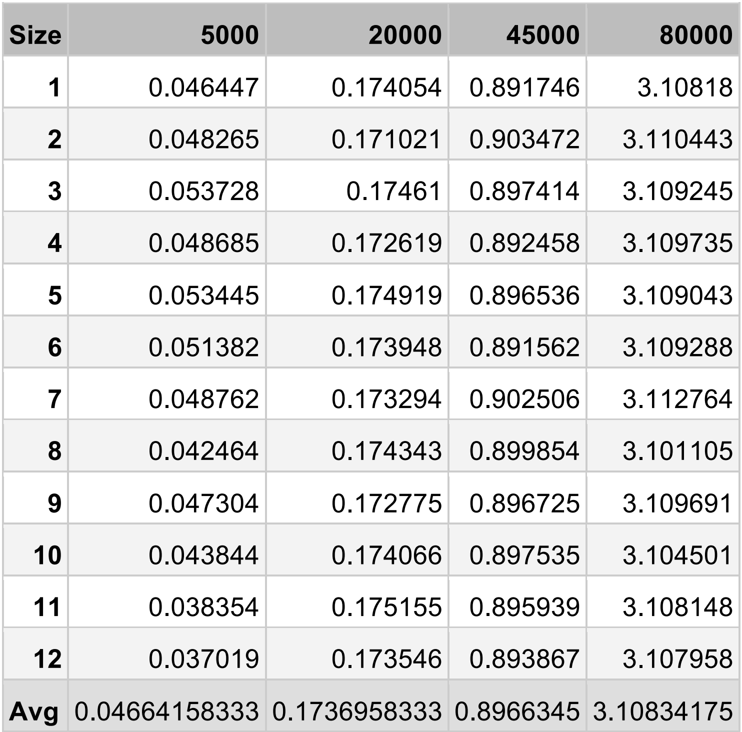
Task 4



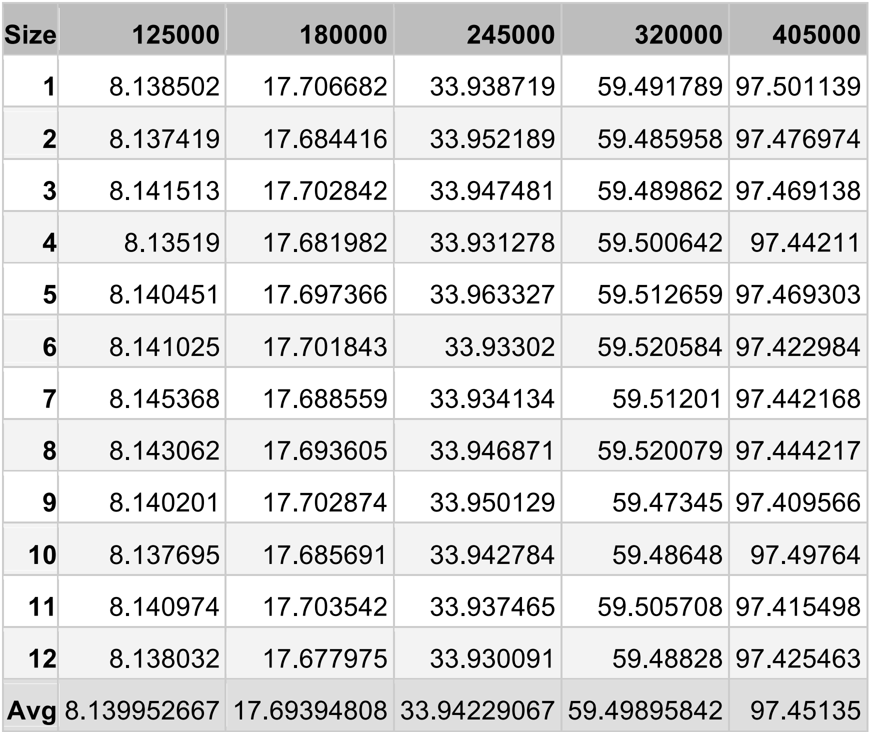
When looking at the trend-line, also known as the spline of the graph, a polynomial like line is being formed. After doing some research as well, it can be concluded that the insertion sort method follows a general O(n^2) trend.

EC Task 3

Here are the charts for the worst-case scenario:



Size	5000	20000	45000	80000
1	0.046447	0.174054	0.891746	3.10818
2	0.048265	0.171021	0.903472	3.110443
3	0.053728	0.17461	0.897414	3.109245
4	0.048685	0.172619	0.892458	3.109735
5	0.053445	0.174919	0.896536	3.109043
6	0.051382	0.173948	0.891562	3.109288
7	0.048762	0.173294	0.902506	3.112764
8	0.042464	0.174343	0.899854	3.101105
9	0.047304	0.172775	0.896725	3.109691
10	0.043844	0.174066	0.897535	3.104501
11	0.038354	0.175155	0.895939	3.108148
12	0.037019	0.173546	0.893867	3.107958
Avg	0.04664158333	0.1736958333	0.8966345	3.10834175

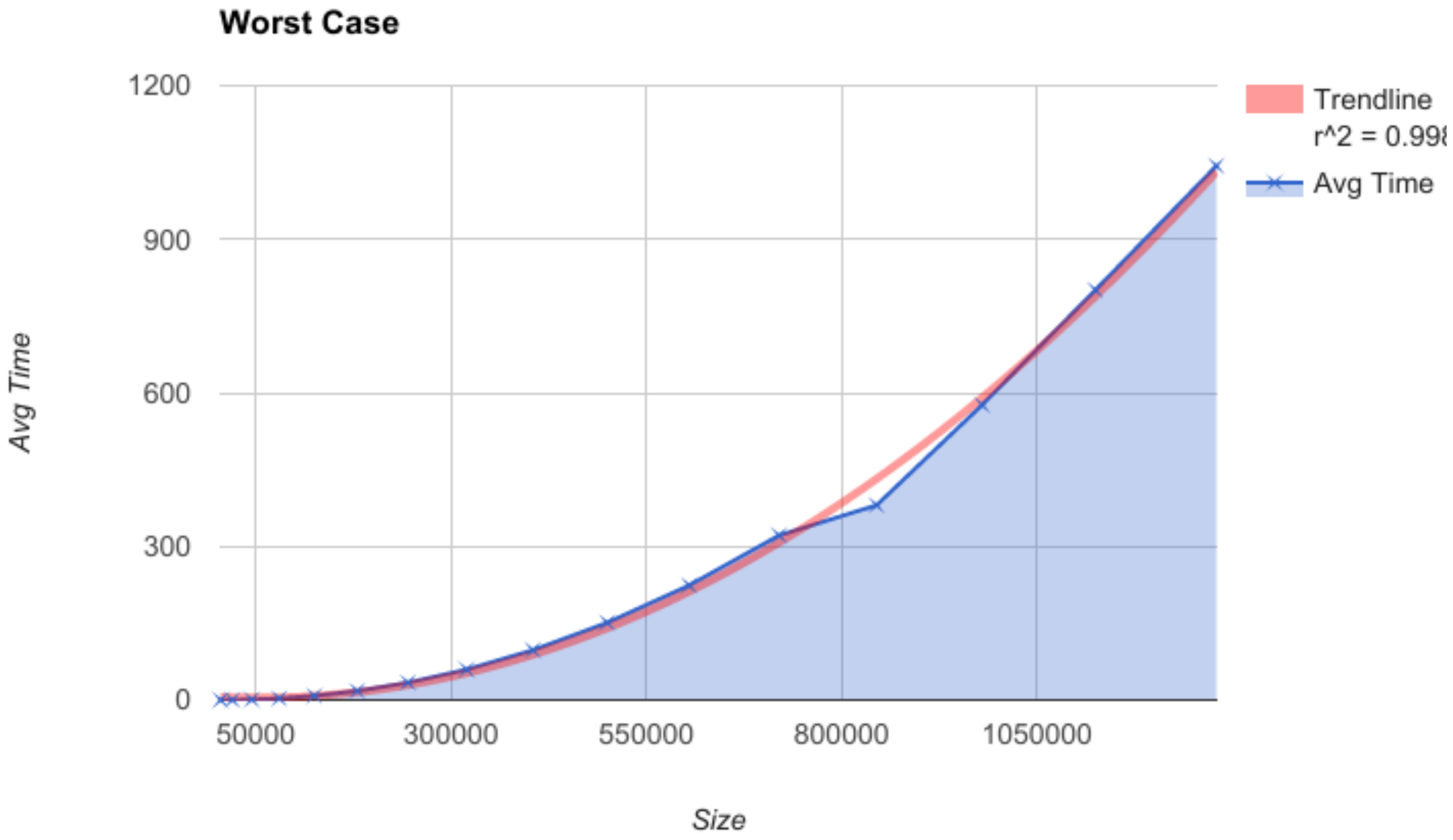


Size	125000	180000	245000	320000	405000
1	8.138502	17.706682	33.938719	59.491789	97.501139
2	8.137419	17.684416	33.952189	59.485958	97.476974
3	8.141513	17.702842	33.947481	59.489862	97.469138
4	8.13519	17.681982	33.931278	59.500642	97.44211
5	8.140451	17.697366	33.963327	59.512659	97.469303
6	8.141025	17.701843	33.93302	59.520584	97.422984
7	8.145368	17.688559	33.934134	59.51201	97.442168
8	8.143062	17.693605	33.946871	59.520079	97.444217
9	8.140201	17.702874	33.950129	59.47345	97.409566
10	8.137695	17.685691	33.942784	59.48648	97.49764
11	8.140974	17.703542	33.937465	59.505708	97.415498
12	8.138032	17.677975	33.930091	59.48828	97.425463
Avg	8.139952667	17.69394808	33.94229067	59.49895842	97.45135

Size	500000	605000	720000	845000
1	150.969018	223.904004	321.514709	369.133942
2	150.928149	223.898194	321.504658	369.729217
3	150.940293	223.966641	321.534813	369.315215
4	150.995186	223.941454	321.465426	370.004402
5	150.953327	223.867002	321.564379	369.751431
6	150.99338	223.801615	321.578381	370.038396
7	150.936691	223.847473	321.516101	391.275428
8	150.938622	223.959254	321.487382	391.736112
9	150.975191	223.803883	321.493127	391.8182
10	150.975796	223.827599	321.387759	391.015094
11	150.946147	223.908988	321.333895	391.631028
12	150.950829	223.94556	321.4028	391.221384
Avg	150.9585524	223.8893056	321.4819525	380.5558208

Size	980000	1125000	1280000
1	217.227178	800.729175	1043.355616
2	608.930074	800.744603	1043.261355
3	608.924359	800.848445	1043.424198
4	608.840267	800.779318	1043.273993
5	609.056273	800.766111	1043.249956
6	608.879819	800.685225	1043.211216
7	609.040831	800.714025	1043.087954
8	608.928233	800.565079	1043.207447
9	608.764078	800.726426	1043.30485
10	609.092696	800.708009	1043.088868
11	608.880749	800.74507	1043.363518
12	608.785749	800.475679	1043.238764
Avg	576.2791922	800.7072638	1043.255645

EC Task 4



Worst Case
1200
900
600
300
0
Avg Time
50000
300000
550000
800000
1050000
Size
Trendline
r^2 = 0.998
Avg Time

Appendix (Code)

```
/**
 * Created by John on 2/14/17.
 */

import java.lang.management.*;
import java.util.Random;
import java.util.Scanner;

public class homework3 implements Runnable{
    public long time;
    public int size = 1;

    public homework3() {
        time = 0;
    }


    private void printNumbers(int[] input) {

        for (int i = 0; i < input.length; i++) {
            System.out.print(input[i] + ", ");
        }
        System.out.println("\n");
    }

    public void insertionSort(int array[]) {
        int n = array.length;
        for (int j = 1; j < n; j++) {
            int key = array[j];
            int i = j - 1;
            while ((i > -1) && (array[i] > key)) {
                array[i + 1] = array[i];
                i--;
            }
            array[i + 1] = key;
//            printNumbers(array);
        }
    }

    private void randomFill(int[] ary) {
        if (size > 0) {

            Random rand = new Random();
            for (int i = 0; i < ary.length; i++) {
                // To keep values in a reasonable range
                ary[i] = (rand.nextInt() % (ary.length * 2));
            }
        }
        else{
            // THIS IS WORST CASE (sorted array that is reversed)
            for (int i = 0; i < ary.length; i++) {
                ary[i] = ary.length - i;
            }
        }
    }

    @Override
    public void run() {
        // Fill array
        int[] ary;
        if ( size > 0)
            ary = new int[size];
        else
            ary = new int[(-1 * size)];

        randomFill(ary);
        //test.printNumbers(ary);

        // Start timer and sorting
        ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();
        time = threadMXBean.getCurrentThreadCpuTime();

        insertionSort(ary);
        time = threadMXBean.getCurrentThreadCpuTime() - time;
    }
}

/**
 * Created by John on 2/16/17.
 */
public class main {
    public static int THREADS = 12;

    public static void main(String [] args)
    {
        for (int j = 1; j < 25; j++)
        {
            System.out.println("\nWith a size of " + 5000 * j * j + " in the
array:\n");

            homework3[] longRun = new homework3[THREADS];
            Thread[] t =new Thread[THREADS];

            // A size of 2000000 resulted in a time of 643 seconds

            for (int i = 0; i < THREADS; i++) {
                longRun[i] = new homework3();
                longRun[i].size = 5000 * j * j * (-1);
//                longRun[i].size = 2000000;
                t[i] =new Thread(longRun[i]);

                t[i].start();
            }
            int index = THREADS - 1;

            while (index >= 0) {
                try{
                    t[index].join();
                    System.out.println((longRun[index].time / 1000000000.0) + " ");
                    index--;
                }catch(Exception e){}
            }
        }

    }


}
```