# CSC 204 Lab 14: Analyzing Algorithms for Performance

## Goals

After doing this lab, you should be able to:

- time segments of code,
- graphically represent the performance of code segments,
- determine explicitly which sort algorithms give the best performance,
- compare the consistency of sort algorithms, and

As you work through this lab, look at each of the Java sort files to see how they work, the Demo files to see how to call the Sort algorithms, and the Timer files to see how to actually time the code. You should also review the `StopWatch.java` and `ArrayUtil.java` files to see how it works.

## Lab Preparation

Skim the material in Chapter 14 of your text. You should copy the Java files from Blackhawk's Lab14 folder into a new project in Eclipse named Lab14. Make a copy of this document and name it "YourName - Lab14". Use your copy of this document to answer the questions below.

## Materials Needed

Be sure you have the following on hand during the lab.

- Your copy of this sheet and your course notebook.
- The files for Lab 14

## The Stopwatch and ArrayUtil Classes

1. Look at the StopWatch.java file. List the four things that must be done with the StopWatch class to time a segment of code.

`start(), stop(), getElapsedTime(), reset()`

2. Look at the ArrayUtil.java file. Is it possible that the same number can be assigned to multiple array indices?

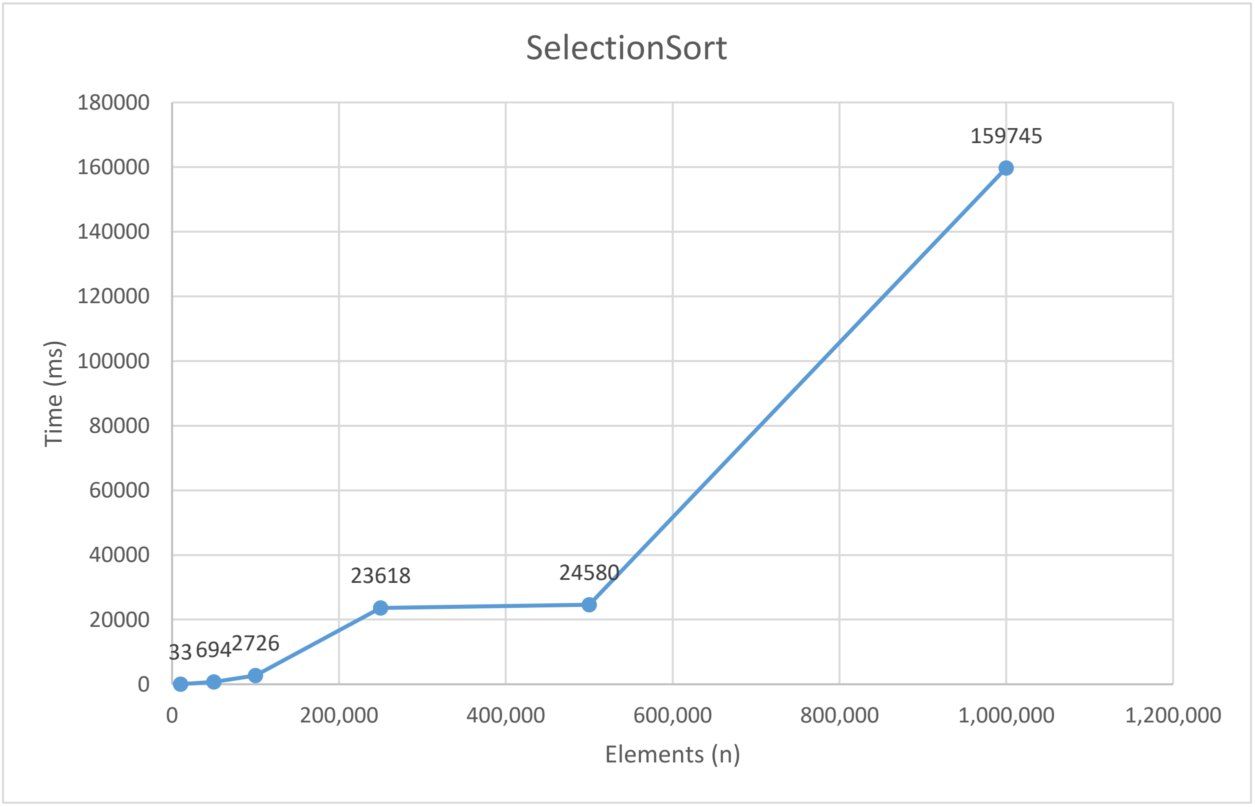
`Yes`

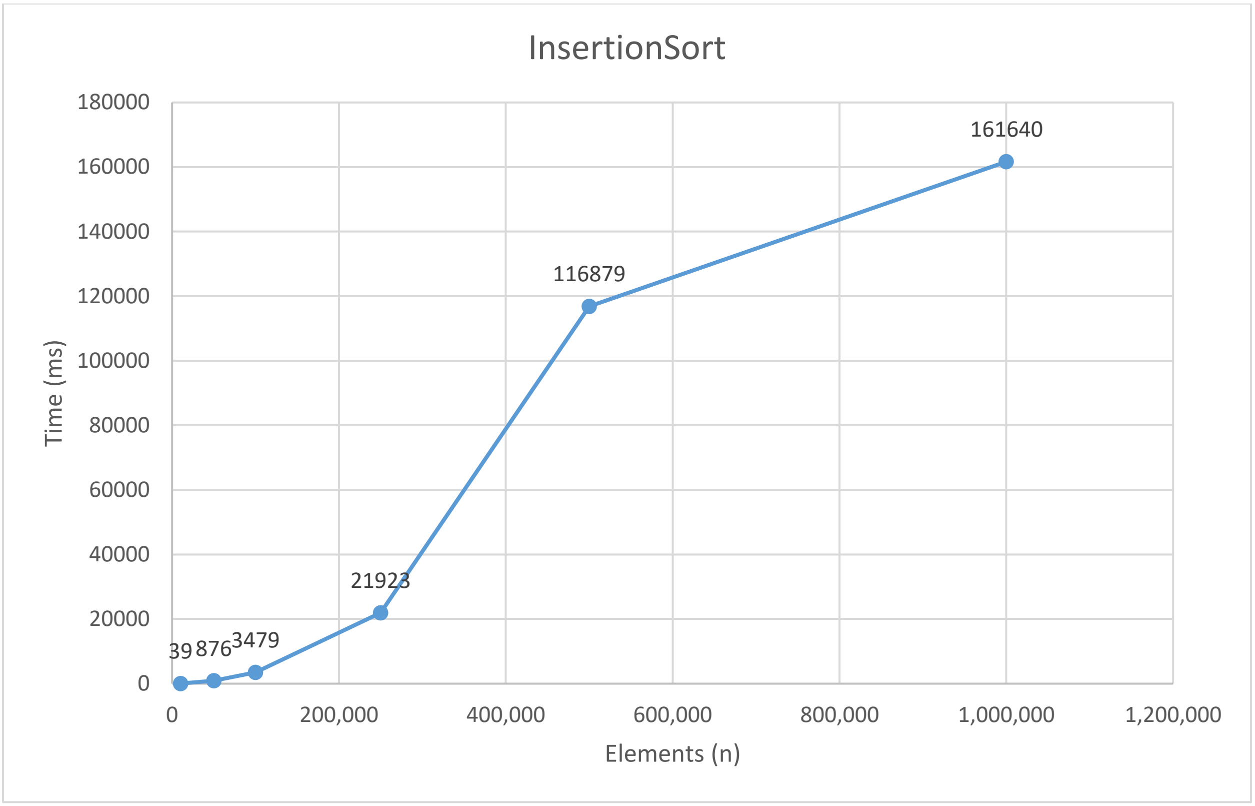
## Generating Performance Data

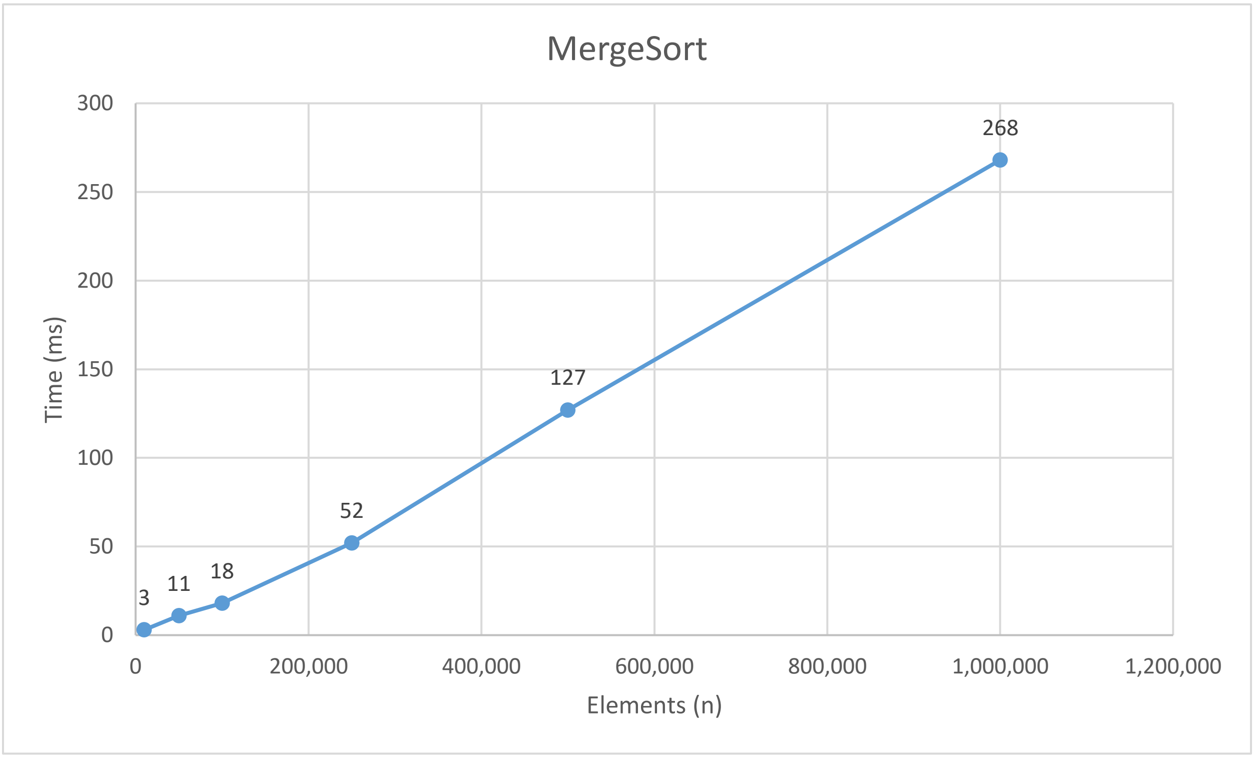
1. To generate performance data it is generally necessary to run a program numerous times using the same size and type of data set and record the runtime. The size of the data set is increased and the "experiment" is repeated. This process continues until one can clearly see how the algorithm performs.
In this lab we will be comparing the performance of the SelectionSort, InsertionSort, MergeSort, and QuickSort. To make things a little easier for you, the programs SelectionSortTimer, InsertionSortTimer, MergeSortTimer, and QuickSortTimer have already been written. You simply have to compile and run them.
Complete the following chart by running the indicated program with the specified number of array elements, and record the runtime in milliseconds. You may need to run the smaller array dimensions multiple times to get meaningful results. **You may work in small teams to fill in this chart as some of the runtimes are long.**

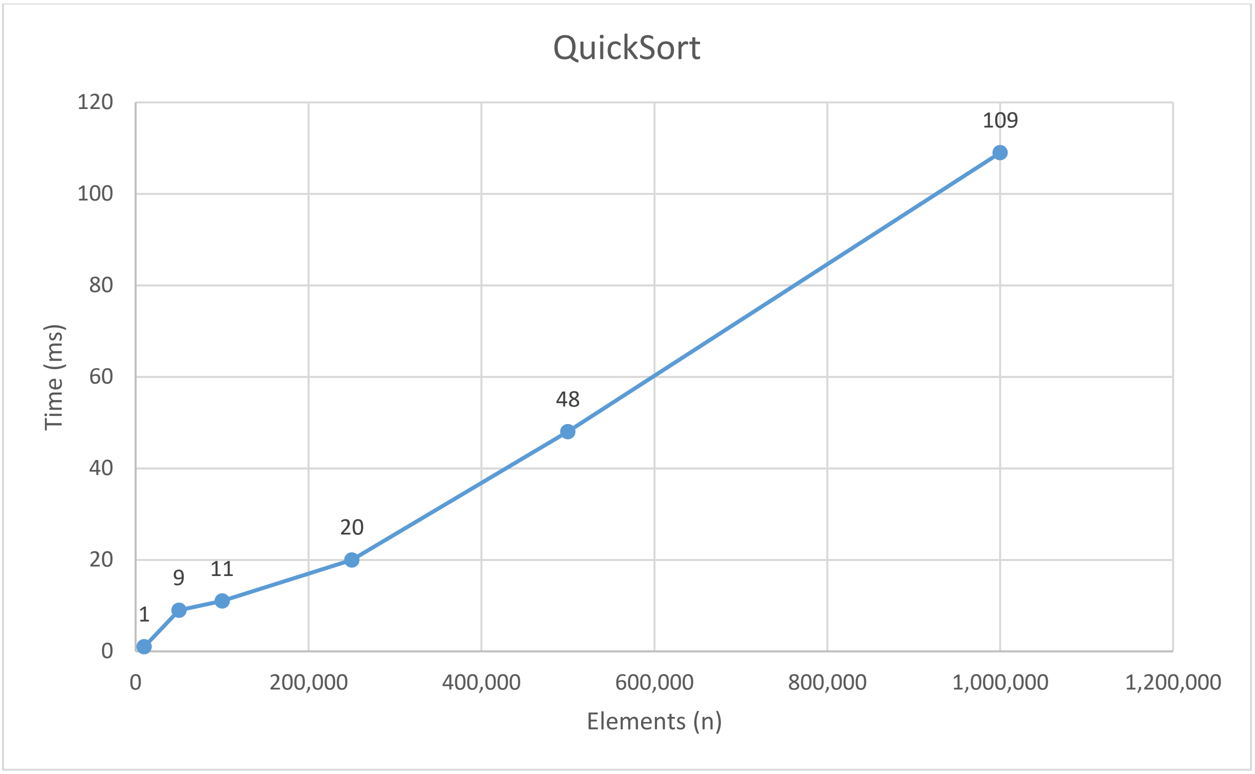
| | *10,000* | *50,000* | *100,000* | *250,000* | *500,000* | *1,000,000* |
|---|---|---|---|---|---|---|
| ***SelectionSort*** | 33 | 694 | 2,726 | 23,618 | 24,580 | 159,745 |
| ***InsertionSort*** | 39 | 874 | 3,479 | 21,923 | 116,876 | 161,640 |
| ***MergeSort*** | 3 | 11 | 18 | 52 | 127 | 268 |
| ***QuickSort*** | 1 | 9 | 11 | 20 | 48 | 109 |

2. With the data from the table, use *Excel or GoogleSheets* to create a plot of time (y-axis) vs. the number of elements, n (x-axis) for each of the methods above. In this view, which, if any, of the methods look like they increase linearly with the number of the elements.




InsertionSort
Time (ms)
Elements (n)
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
39
876
3479
21923
116879
161640


MergeSort
Time (ms)
Elements (n)
0
50
100
150
200
250
300
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
3
11
18
52
127
268



3. Now, plot the log of the time vs the log of the array size. On a log-log plot, the function $y = ax^b$ is transformed to `log(y) = log(a)+b * log(x)` which should result in a straight line with the slope equal to the “order” (b) of the function. Based on your plot, what can you say about the order of the various sorting methods as they relate to each other?

```
SelectionSort and InsertionSort seem to increase exponentially
while MergeSort and QuickSort increase linearly.
```

4. Based on your results, why it is necessary to test with large array sizes?

```
All of the times are fairly low at the start but as the array
increases, SelectionSort and InsertionSorts’ times get slower
and slower, thus proving that they are not good in sorting a
high number of elements.
```

## Problems with Consistency

1. While the QuickSort algorithm generally performs well, its weakness is that its performance is not always consistent. Using 10000 elements run the algorithm 10 times and see if you spot any inconsistencies in the runtimes. Record your shortest and longest runtime.

```
2, 3, 2, 3, 2, 3, 2, 4, 3, 2
Shortest: 2ms
 Longest: 4ms
```

2.  Increase the number of elements and repeat the experiment.  Do any of the runtimes significantly differ from the average?

```
Elements: 1,000,000
69, 73, 79, 83, 78, 79, 83, 72, 80, 73
Shortest: 69ms
 Longest: 83ms
The average is 76.9ms and there is a 14ms difference between the
longest and shortest times.
```

## Turn In

Create a folder named “Lab14” in your shared Google folder.  Place your copy of this document into your Lab14 folder.  Also, upload your spreadsheet graph to your shared Lab14 folder.