**An Analysis of the Interactions between Programmer, Language, and Algorithm**

**Andrew Combs**

**Cody Jenkins**

**Department of Computer Engineering/Computer Science**

**University of Louisville**

**4-9-2018**

**An Analysis of the Interactions between Programmer, Language, and Algorithm**

Andrew Combs

Cody Jenkins

Department of Computer Engineering/Computer Science

University of Louisville

4-9-2018

**Executive summary**

The purpose of this experiment is to analyze the relationships between programmer, programming language, and algorithm. It was executed by having the insertion sort, selection sort, and bubble sort algorithms written in C#, Java, and Python by Cody and Andrew. The running time of each algorithm to sort a given list of floating-point numbers is the response.

**Table of Contents**

Executive Summary i

1. Introduction 1

2. Experiment Design 1

3. Experiment Results 2

4. Analysis 2

5. Results and Conclusion 5

Appendix A: Complete Experiment Results X

Appendix B: Complete Analysis Results X

A.1 ANOVA

A.2 Residuals

A.3 Tukey

References

**1. Introduction**

Sorting algorithms are a well-studied field, due to their common use in everyday applications. Because the runtime of an application is important to users, efficient algorithms are highly desired. In addition, the programming language an application is written in can also affect how quickly it runs. Compiled languages are generally faster than interpreted languages, due to their ability to modify the program before it is run to help enable it to run faster. Finally, it is worthwhile to explore the interaction between programmer and programming language. Of course, if a particular programmer has years of experience with a particular language, he or she will be well-versed in the various nuances of that language. Thus, it is the goal of this experiment to analyze the relationships and effects of programming language, algorithm, and programmer on the execution time of a given problem.

**2. Experiment Design**

For this experiment, the students decided to analyze the effects of different programming languages, different algorithms, and different programmers on program runtime. To evaluate this experiment, they decided to have two programmers implement three different sorting algorithms in three different languages. The three algorithms implemented are:

* Insertion sort
* Selection sort
* Bubble sort

The three languages used are:

* C#
* Java
* Python

The two programmers are:

* Andrew Combs
* Cody Jenkins

In order to analyze the relationship between the different factors, the students employed a full factorial design. Programs implementing each of the algorithms in each of the languages were written by both programmers, and their execution time was measured. Each program read in a file of 10,000 floating-point numbers into a list every time it runs, and sorts the list using one of the sorting algorithms. To minimize the effect of the environment in which the programs are running, the students elected to run them all on the same machine. The time each program took to sort the given set of numbers, in milliseconds, is the response. A separate program was written that would call the other programs and track their execution times.

**3. Experiment Results**

To generate the final data, each of the 18 different program and algorithm combinations are called 3 times, in completely random order. The run times are collected and imported into Minitab. This results in a table containing 3 factors, 1 response, and 54 responses representing 3 complete replicates. (The design is balanced.)

One concern with the experiment was the potential for the operating system to interfere with the program execution times. For example, if the input file were cached for some operations and not others, those operations would perform more quickly. However, a preliminary analysis of the data does not show symptoms of this type of problem. (Said symptoms are very high run times for the first calls and low run times for subsequent calls, or a single call occasionally taking several times longer than normal)

**4. Analysis**

To begin the analysis, an ANOVA is run considering all interactions. The ANOVA table (see Figure 1) shows all factors and interactions as being completely significant, with all P-Values equal to zero (less than alpha of 0.05.)

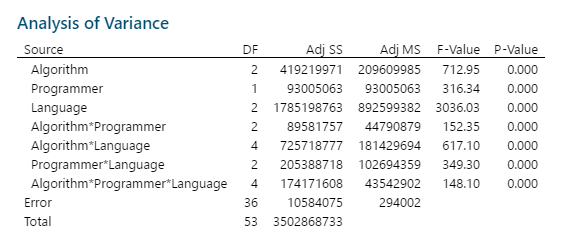


Figure 1: ANOVA Table

Consultation of the residual plots (see Figure 2) quickly reveals the reason for these unusually low P-Values. The residual plots cause us to conclude these data have almost no significant variation. That is to say, if the same program, having a constant programmer, algorithm, and language and run on the same machine with the same input, is run multiple times, the time required to run will be nearly constant. This is not particularly ground-breaking, as any domain expert would tell you to expect this result. It does, however, point to a fundamental flaw in this experiment’s design.



Figure 2: Residual Plots

Experimental replicates should represent multiple implementations, rather than multiple executions. To collect more relevant data, each programmer should implement each algorithm in each language multiple times, and the run times of these multiple distinct programs should be considered. However, this leads to some implementation issues. If a programmer is asked to implement the same algorithm twice, the implementations are likely to be nearly identical. In order to obtain sufficiently different samples, large programs would be required, which would make the experiment difficult to implement. Perhaps a more effective approach would be to change the factor from Programmer to Programmer Type. In this way, multiple programmers would be grouped by a common, relevant attribute. While this wouldn’t allow comparative analysis of individual programmers, it could lead to interesting conclusions such as “Based on their most proficient language, what type of programmer is best at switching languages?”

Setting aside, for a moment, the many issues with this model, a Tukey analysis is performed. While the inability to strongly assume normality means the results of this analysis can’t be assumed to be representative of a larger population, the analysis may still be beneficial in exploring the current data set.

Looking first at single factor comparisons (see Figures 3,) it appears that selection sort is the most effective algorithm, Andrew is the most effective programmer, and Java is the fastest language. Adding two-way interactions (see Figure 4) seems to indicate that language is the most important factor.

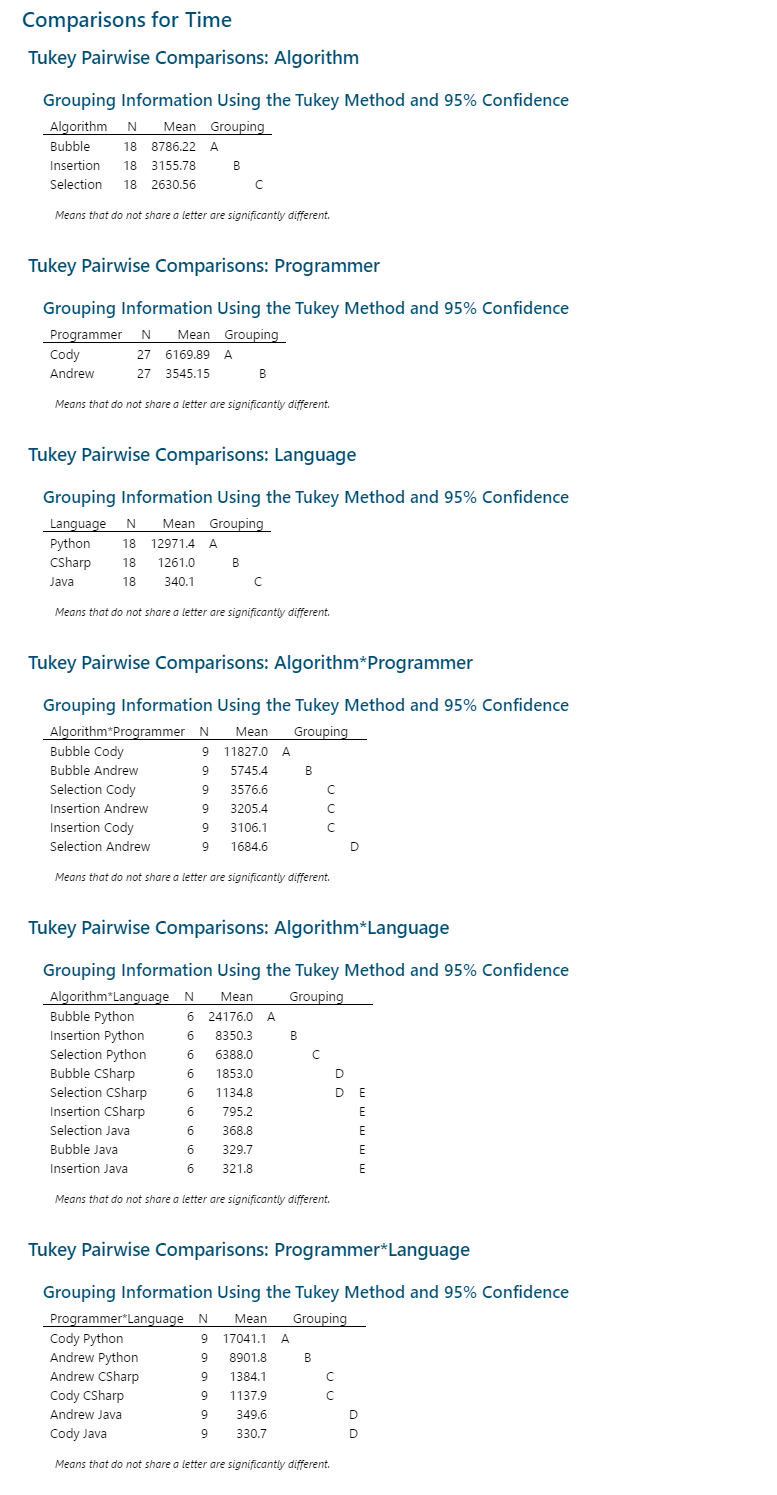


Figure 3: Single Factor Tukey

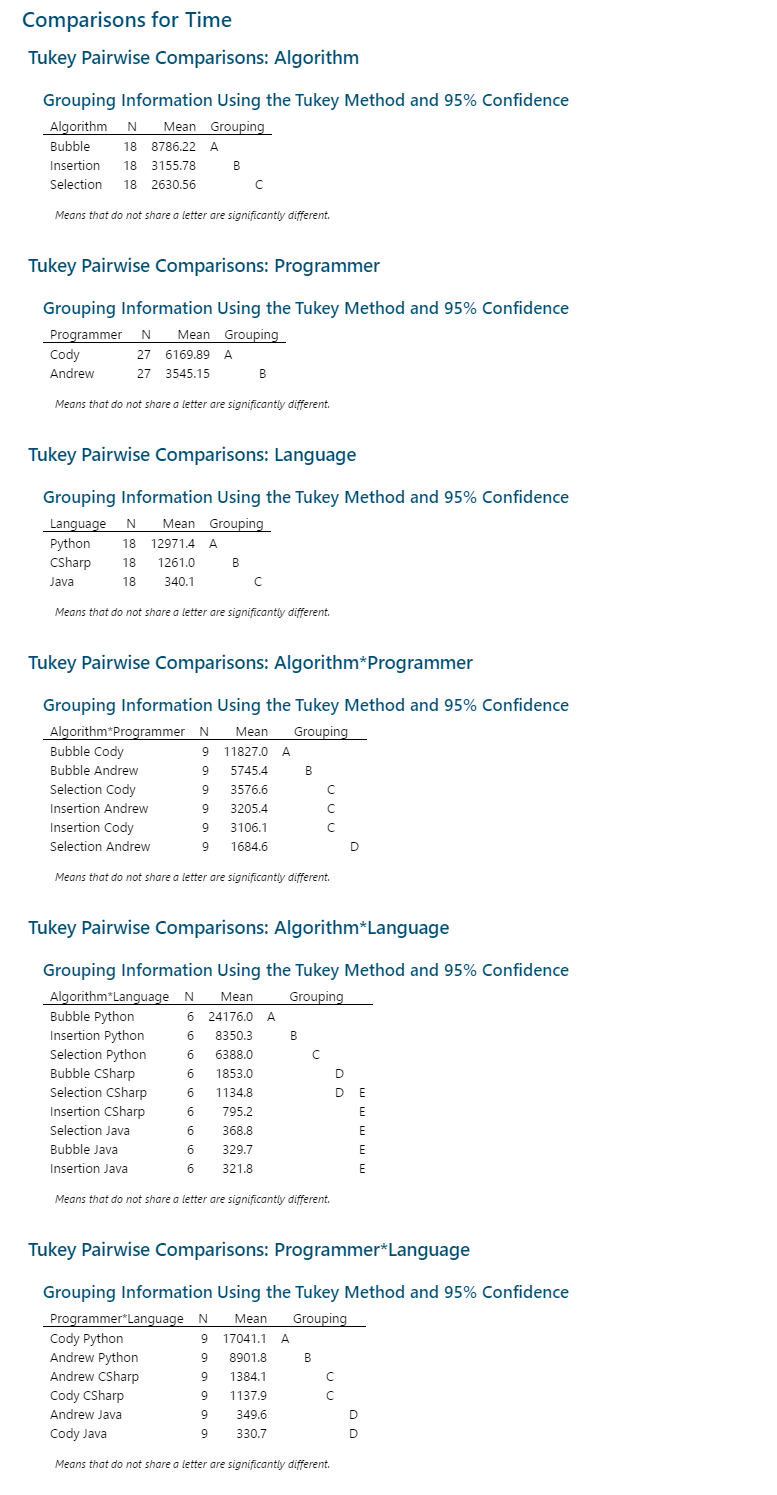


Figure 4: Two-way Interaction Tukey

**5. Results and Conclusion**

The runtime of various algorithms are well-defined and have been studied frequently. Divide-and-conquer search algorithms, which divide the input into smaller pieces and recombine them later, perform much better than comparison-based search algorithms. The three algorithms were chosen because they all had polynomial time complexity.

As was expected, python took much longer to run than both C# and Java. This is due to the fact that Python is interpreted, whereas both C# and Java are compiled. Compiled languages benefit from utilizing compiler optimizations, which help make the code run faster. However, what was not expected was that Java was faster than C#. This could be due to the fact that C# relies on the common-language runtime and just-in-time compiling to increase its flexibility as part of the .Net framework.

The next step in pursuing this subject is to design a second experiment involving many more programmers, such that variation will come from differences in implementation rather than differences in program execution. This should provide higher quality, more normalized data from which better conclusions can be derived.

**Appendix A: Complete Experiment Results**

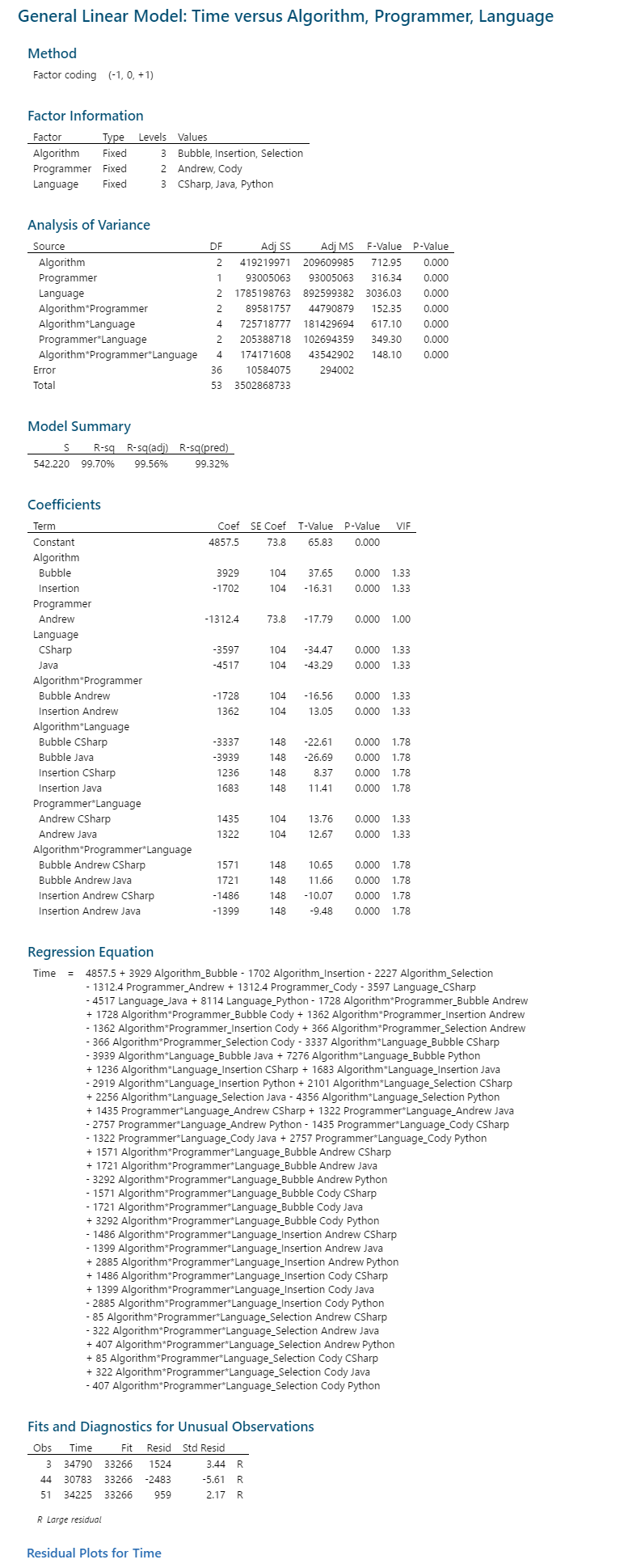
In this section, the raw data collected in the experiment are presented.

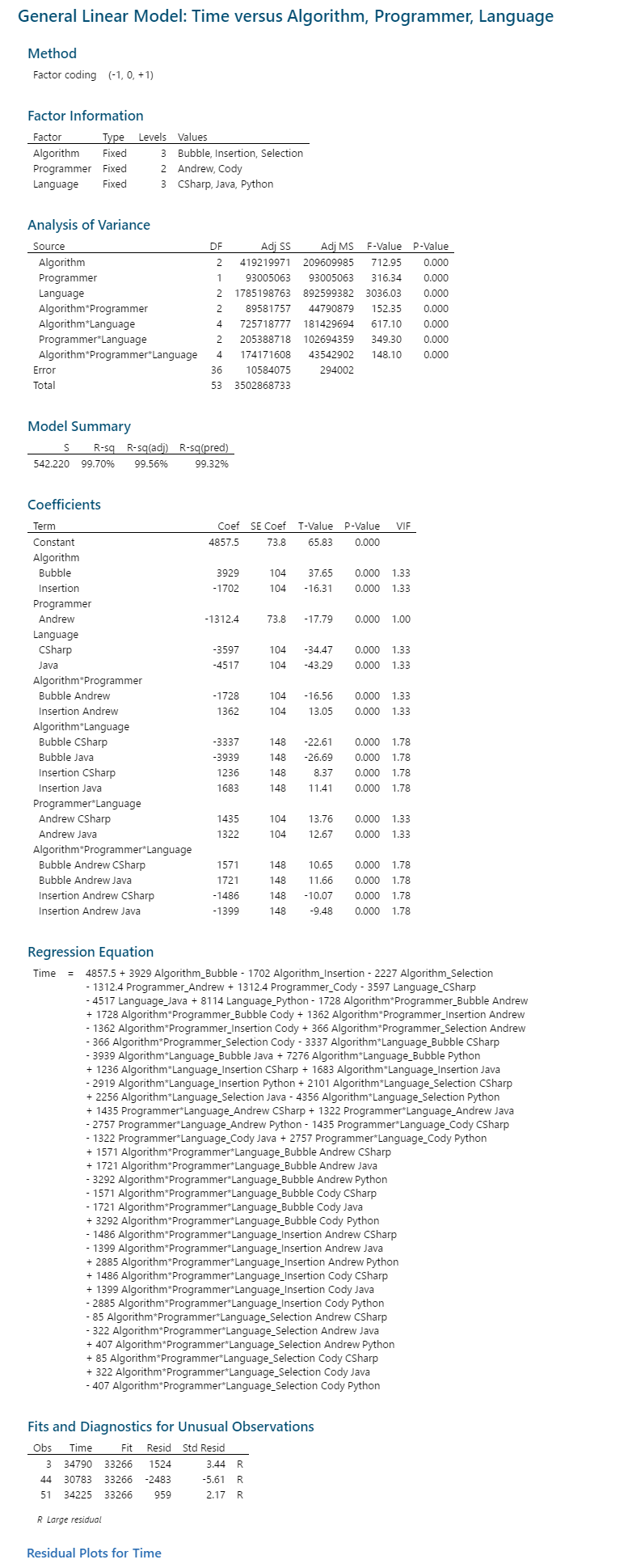
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Programmer | Language | Time |  | Algorithm | Programmer | Language | Time |
| Selection | Andrew | Java | 566 |  | Insertion | Cody | Python | 7978 |
| Selection | Cody | Python | 9795 |  | Bubble | Andrew | Java | 334 |
| Bubble | Cody | Python | 34790 |  | Selection | Andrew | CSharp | 1523 |
| Bubble | Cody | CSharp | 1997 |  | Selection | Cody | CSharp | 708 |
| Selection | Cody | Python | 9501 |  | Selection | Andrew | CSharp | 1496 |
| Bubble | Andrew | CSharp | 1792 |  | Bubble | Cody | CSharp | 1817 |
| Insertion | Andrew | Java | 265 |  | Insertion | Cody | Java | 324 |
| Bubble | Andrew | Python | 15698 |  | Selection | Andrew | Java | 319 |
| Selection | Andrew | CSharp | 1598 |  | Bubble | Andrew | CSharp | 1914 |
| Selection | Cody | Python | 9756 |  | Selection | Andrew | Java | 383 |
| Selection | Cody | Java | 319 |  | Selection | Cody | CSharp | 765 |
| Insertion | Andrew | Python | 8989 |  | Insertion | Andrew | CSharp | 816 |
| Insertion | Cody | Java | 400 |  | Selection | Cody | CSharp | 719 |
| Bubble | Cody | CSharp | 1847 |  | Bubble | Andrew | Python | 14601 |
| Bubble | Andrew | CSharp | 1751 |  | Bubble | Cody | Java | 367 |
| Bubble | Andrew | Java | 369 |  | Selection | Cody | Java | 321 |
| Insertion | Cody | CSharp | 781 |  | Bubble | Cody | Python | 30783 |
| Insertion | Andrew | CSharp | 758 |  | Selection | Cody | Java | 305 |
| Insertion | Andrew | Python | 8406 |  | Insertion | Andrew | Java | 289 |
| Insertion | Cody | CSharp | 842 |  | Insertion | Andrew | Java | 330 |
| Insertion | Andrew | Python | 8187 |  | Bubble | Andrew | Python | 14959 |
| Insertion | Cody | Python | 8254 |  | Bubble | Cody | Java | 326 |
| Insertion | Cody | CSharp | 765 |  | Selection | Andrew | Python | 3038 |
| Insertion | Andrew | CSharp | 809 |  | Bubble | Cody | Python | 34225 |
| Insertion | Cody | Java | 323 |  | Selection | Andrew | Python | 3158 |
| Bubble | Andrew | Java | 291 |  | Selection | Andrew | Python | 3080 |
| Insertion | Cody | Python | 8288 |  | Bubble | Cody | Java | 291 |

**Appendix B: Complete Analysis Results**

In this section, all direct output from Minitab is presented in it’s entirety.

**B.1 ANOVA**

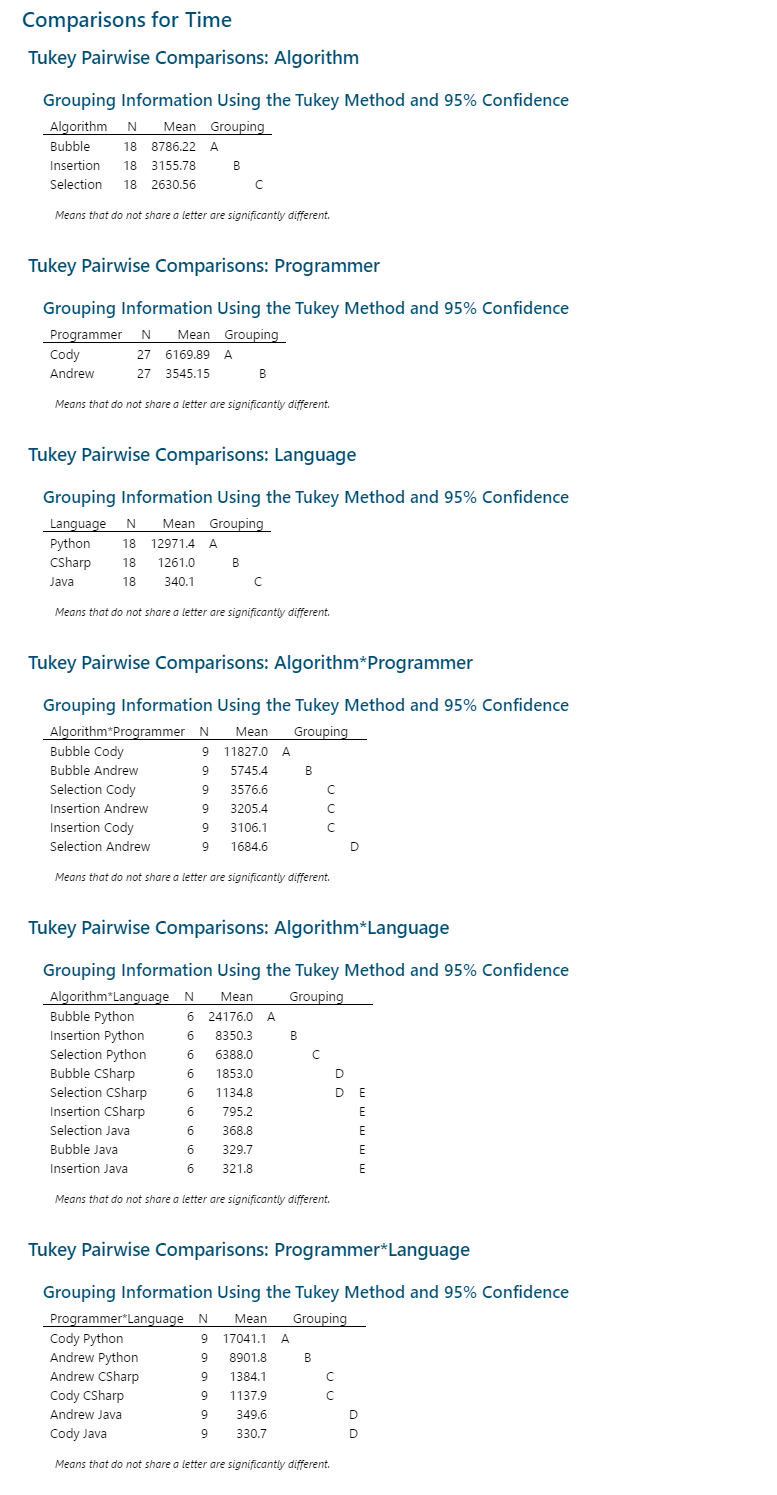




**B.2 Residuals**



**B.3 Tukey Comparisons**



**References**

Algorithm definitions were obtained from:

Selection sort. (2018, April 09). Retrieved April 9, 2018, from <https://en.wikipedia.org/wiki/Selection_sort>

Insertion sort. (2018, April 09). Retrieved April 9, 2018, from <https://en.wikipedia.org/wiki/Insertion_sort>

Bubble sort. (2018, April 09). Retrieved April 9, 2018, from <https://en.wikipedia.org/wiki/Bubble_sort>