2014

Comparative Study of C, Java, C# and Jython

Poonam Goyal

A COMPARATIVE STUDY OF C, JAVA, C# AND JYTHON

By

Poonam Goyal

A thesis submitted to the
School of Computing
In partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

December, 2014

The thesis "Comparative Study of C, Java, C# and Jython" submitted by Poonam Goyal, School of Computing student in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

Dr. Roger Eggen
Thesis Advisor and Committee Chairperson

Dr. Sanjay P. Ahuja

Dr. Ching-Hua Chuan

Accepted for the School of Computing:

Dr. Asai Asaithambi
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

Dr. Mark A. Tumeo
Dean of the College

Accepted for the University:

Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENTS

I wish to thank my parents, Shakun Goyal and Raj kumar Goyal, for their unwavering support during the many hours I dedicated to achieving this milestone in my life and career. I highly appreciate the understanding and patience of my family for achieving one of the greatest moments of my lifetime.

I thank my thesis director, Dr. Roger Eggen, for his expert advice, feedback and suggestions. I also thank Dr. Sanjay Ahuja and Dr. Ching-Hua Chuan, who agreed to be on my thesis committee and provided great feedback in this research process. Working with these three distinguished UNF faculty members has been an honor and a privilege.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Comparing programming languages is a common topic among programmers and software developers. With the recent changes in programming standards and continual upgrades in hardware design, many new programming languages are being developed, while existing ones are currently going through several enhancements in terms of design and implementation. In this research, we present a comparative study of four programming languages, C, Java, C#, and Jython, with respect to the following criteria: memory consumption, CPU utilization, and execution time. Each test was performed in a distributed system using TCP sockets with 1, 2, 4 and 8 clients, and on a symmetric multiprocessing system.

Chapter 1

INTRODUCTION

Many programming languages exist today, making it difficult for a programmer or developer to decide which language will work best for his or her specific project. Most of the time, the decision is based on the programmer's advanced knowledge of a particular language, which may not always be the best for their specific problem. Other times, the decision depends merely on the popularity of a particular language.

Several arguments have been made about how various algorithms perform with respect to speed, complexity, and efficiency while solving identical problems on identical hardware. It is equally important to learn how algorithms perform when written in different programming languages; by identifying the differences, a programmer can choose a programming language based on its strengths.

In this study, programming languages C, C#, Java and Jython are compared by implementing the following algorithms: Bubble Sort, Quick Sort, Linear Search, and Binary Search, in three categories on identical hardware and operating system: (1) CPU utilization; (2) memory usage; and (3) execution time for client server communication using TCP socket.

The results of this research show that a programming language does have an effect on performance and other properties of execution. If one programming language implementation executes efficiently in one category, it might not perform similarly in other categories. The results of this study make it possible to determine the most appropriate language to solve a particular problem. The appropriateness is determined by statistical data and measurement analysis.

Chapter 2

LITERATURE REVIEW

This chapter summarizes recent articles and other publications focusing on programming language comparison. The work of Lutz Prechelt [Prechelt 00] and P Sestoft [Sestoft 10], as discussed in sections 2.1 and 2.4, are chosen as primary references as they also attempt to solve the same problem of comparing different programming languages. This section also includes the review of a few other research papers, as the authors present well-structured analysis of the problem at hand.

2.1 An Empirical Comparison

The article by Lutz Prechelt, "An Empirical Comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a Search/String-Processing Program," [Prechelt 00] presented the programming language comparison as implemented by different programmers. The study compared various properties such as run-time, memory constraints, and reliability. The research described by Prechelt also considered language efficiency. Our study compares efficiency of four languages, while Prechelt's study focused on human factors including various programming styles. Also, due to the nature of the study, Prechelt did not use a Relative Complexity Metric, which is a representation of a program's metrics, to statistically compare the programming languages. There are additional variations resulting from different programming styles and different frameworks. Prechelt used a

Relative Complexity Metric, by implementing the algorithms for comparison on a common hardware configuration. [Prechelt 00]

2.2 A Comparison of C and Pascal

Authors Alan R. Feurer and Narain H. Gehani , in [Feuer & Gehani82] “A Comparison of the Programming Languages C and Pascal,” considered the language constructs and design patterns of C and Pascal. The authors believe that Pascal programs tend to be more reliable than C because of Pascal’s richer set of data types, strong typing, readability and portability. In contrast, the authors also believe that C is much more flexible, and can be used effectively in more applications than Pascal, since the programmer has more control. The authors list all of the strengths and weakness of each language in much the same way as our research project. All of the features and data types of each language were considered with an in- depth look at the language aspects of C and Pascal. After describing the languages in detail, the authors listed which applications should be implemented in which language [Feuer & Gehani82]. Feurer and Narain’s study did not provide measurement data, or statistical analysis to give valid insight into the comparison of these two languages. This article is more a collection of programmer opinions rather than statistical fact [Feuer & Gehani82].

2.3 Nonprocedural Computer Language and Programmer Productivity

Authors Harel & McLean in [Harel & McLean85] “The Effects of Using a Nonprocedural Computer Language on Programmer Productivity,” looked at the differences of two programming languages, Focus, a non-procedural language, and Cobol, a procedural language. Harel & McLean studied **a** comparison of programmer's productivity and execution time. Six "mid-sized" applications were developed by different programmers. Independent variables associated with this study are:

1. Hardware.
2. Programming mode.
3. Organizational characteristics of the program development.
4. Source languages.
5. Types of applications.
6. Programmer's expertise.

In addition, there are several dependent variables linked with these independent variables, the major one being time. Several run-time factors are also studied, such as total CPU time for compilation and execution, total number of source lines, and I/O operations. Each of these variables was measured and studied. Each measurement was statistically analyzed including simple averages and standard deviation. Once all of the data was processed, the authors gave their results, concluding that Cobol was faster and more efficient for the CPU, but that Focus was a more productive language from the perspective of the programmer. [Harel & McLean, 1985]

2.4 Numeric Performances in C, C# and Java

Author Peter Sestoft in [Sestoft10] "Numeric Performances in C, C# and Java**,**" compared the performance of C, C# and Java on 4 trivial cases: matrix multiplication, division of intensive loops, polynomial evaluation, and distribution function. The tests revealed that execution speeds vary significantly among these three languages. The C language performed exceptionally well followed by C#, while Java's performance was not satisfactory. Facts that qualify special consideration: [Sestoft10].

- "Considering Java's bulky array depiction and the lack of precarious code, it is noteworthy how upright the Sun Hotspot-server and virtual machine executes.
- Microsoft's C#/.NET runtime normally performs fine, but there is ample scope for enhancement in the safe code for matrix multiplication.
- The Mono C#/.NET runtime stayed consistent, and in Version 2.6 the overall performance is good."[Sestoft10].

2.5 A Performance Analysis of Java and C

Authors Ambika Pajjuri and Haseeb Ahmed in [Pajjuri00] "A Performance Analysis of Java and C," compared the Java and C, two admired programming languages. The authors presented a performance assessment of varied algorithms written in C and Java on Windows and UNIX operating system environments. The metrics used in the study were memory usage, speed of execution, overhead and additional special features that distinguish these two programming languages.

Both programming languages were inspected on how their design selections impact performance over semantics and programming paradigms. The algorithms considered were those frequently used in embedded systems, and the MD5 (Message-digest) cipher. Outcomes illustrated that, overall, C delivered superior runtime performance over Java across both Linux and Windows platforms. [Pajjuri00]

Chapter 3

RESEARCH METHODOLOGY

In order to measure the effectiveness of this comparison, the research is divided into several components. The first of several major components of this research are the programming languages themselves.

3.1 Why C, C#, Java and Jython

3.1.1 The C programming language

The C programming language is a robust language. As it combines the features of high-level languages with the capabilities of low-level languages, it is appropriate for writing business packages, as well as system software and applications. It has been chosen for this study as it is the foundational language of many other programming languages.

3.1.2 The C# programming language

The C# programming language is a multi-paradigm, object-oriented programming language which facilitates inheritance, abstraction, polymorphism, and encapsulation. The objective of the language is to enhance a programmer's productivity. It is growing in popularity due to its efficiency and ease of coding. The framework manages the execution of applications and Web services. In addition, it offers several other functionalities that include memory management and security enforcement. Like Java, C#

also has automatic garbage collection and has a similar syntax structure. In this study, we will examine Mono, which is an Open Source compiler for C#, which is used in order to provide consistency for the study. It has been chosen for this study for its syntactical similarity with Java and to explore how it compares to Java [Bates04].

3.1.3 The Java programming language

The Java programming language has been chosen for this study since it is dynamic, object oriented, distributed, portable, multithreaded, and strongly typed.

3.1.4 The Jython programming language

The Jython programming language has been chosen for this study as it is a combination of Java and Python programming languages, implemented to generate Java byte code. Jython runs on any JVM (Java Virtual Machine) and thus we wanted to determine if it executes as fast as the popular Java.

3.2 Algorithms Studied

Each programming language performs different algorithms such as Bubble Sort, Quick Sort, Linear Search, and Binary Search using integer, float and string data types. The algorithm and language implementation, using a variety of data types, will yield appropriate performance characteristics of the languages for meaningful comparison.

3.2.1 Sorting

Sorting is a technique for ordering a list of numbers in a particular sequence. In this research, we will be performing experiments on two types of sorting algorithms: Bubble Sort and Quick Sort.

3.2.1.1 Bubble Sort

Sorting activities for Bubble Sort:

1. Make multiple passes over the list. In every pass:
    a. Compare adjacent elements in the list.
    b. Exchange the elements if they are out of order.
    c. Each pass moves the largest (or smallest) elements to the end of the list
2. Repeating this process in several passes eventually sorts the array into ascending (or descending) order.

Bubble sort is only suitable to sort an array with small data size.

3.2.1.2 Quick Sort

Quick Sort works on the technique of dividing the list in two parts based on values higher or lower than a randomly chosen pivot element, and then recursively quick sorts each of the sub lists.

1. Choose an element randomly from the list that will work as the pivot element.
2. Arrange the list in such a way that all the elements with values higher than the pivot come after the pivot, and similarly, all the elements having lower values than the

pivot come before the pivot, and elements equal to the pivot can be placed on either side of the pivot. This brings the pivot to its final sorted position. This process partitions the list into two parts where all elements less than the pivot are in one part, and all elements greater than the pivot are in the other part.

3. Repeat Step 1 and 2 to each of the partitions.

3.2.2 Searching

Searching is a process of finding a particular item from a collection of items. Typically, the result of search is either true or false, which indicates whether the searched item was found. If the item was not found, a result of false is returned; if the item was found, its location is reported. There are two basic approaches: Linear search and Binary search.

3.2.2.1 Linear Search

The linear search algorithm looks down a list, one item at a time.

3.2.2.2 Binary Search

The binary search algorithm compares the desired item with the item in the middle of a sorted list. If the middle item is larger than the desired item, the first half is examined; otherwise the second half of the list is examined. The process is repeated on the half to be examined.

Each algorithm was written in all the four languages (C, Java, C# and Jython) using appropriate features of the given language.

3.3 Metrics and Statistical Analysis

From this study, it can be determined which programming language implementation was better than the rest for each of the algorithms when different data types are used. The results of the analysis provide programmers with useful information that helps them choose the best programming language for the implementation of the four different algorithms used in this study [Squared05].

Big O notation is the accepted measure for categorizing the performance of an algorithm. It depicts the performance of an algorithm when the amount of data increases. Along with CPU Utilization, Big O notation can also be used to determine the memory consumption of a particular algorithm. It also provides information about the rate of change of processing time of an algorithm as the amount of data increases [Bell09].

3.3.1 CPU Utilization

CPU usage refers to the amount of work done by the processor. Observing CPU utilization reveals the workload of a given physical processor for real machines. CPU utilization is a fundamental performance measure, used to determine a computer's efficiency. Computers do not accomplish tasks when the CPU is idle.

3.3.2 Memory Usage

Memory usage specifies the amount of main memory used or referenced by a program while it is running. This includes all types of active memory regions, such as the code segment, that includes program instructions, initialized as well as uninitialized data segments, call stack, and heap memory. It also includes the memory needed to pull any added data structures like symbol tables, open files, shared libraries mapped to a current process, and debugging data structures that a program requires while executing.

Larger programs use more memory. The programs themselves usually do not devote the largest portions to their own memory usage; rather, the structure introduced by the runtime environment consumes most of the memory.

3.3.3 Runtime in Distributed System using TCP Sockets

A distributed system consists of multiple independent computers connected by a network that is equipped with distributed system software. This software allows computers to coordinate their activities and share their resources of system software, hardware, and data.

In distributed programming, a problem is divided into many sub problems or tasks which are distributed to different machines. Although the machines run independently, they still have to interact with each other for the inputs and the results. This is contradictory to the scenario in which the end-user assumes that there are different computers whose locations and functionality is not transparent. Advantages of distributed systems include enhanced reliability, availability and performance, local self-sufficiency, more economy, and many others. In this study we will calculate the runtime in a Distributed System using TCP Sockets [Berlin03].

3.4 The Host Environment

The programs for studying CPU utilization and memory usage were run on Atlas, whereas client-server programs were run on Uranus. The specifications for the two systems are described in the following subsections

3.4.1 Uranus

Uranus is a thirteen-node Beowulf cluster with Gigabit Ethernet network. All nodes are made up of 2.83GHz Intel Xeon processor.

3.4.2 Atlas

Atlas is a shared memory multiprocessor machine. It has a Quad Quad-Core Intel Xeon Processor with 64 threads running at 2.00 GHz along with 128 GB RAM.

Chapter 4

RESULTS AND DISCUSSION: CPU UTILIZATION

This chapter presents the results of our comparison study of the programming languages C, Java, C# and Jython using the metrics of CPU utilization on the Bubble Sort, Quick Sort, Linear Search and Binary Search algorithms. We used six different data sizes 10000, 20000, 40000, 60000, 80000 and 100000.

CPU usage refers to the amount of work done by the computer's processor. Let us say process X starts at time a, and finishes at time b. The total time that Process X takes from start to finish is b-a. Of this time, let us say p is the time spent on Process X (between a and b), and q is the idle time (between a and b). Then $p + q = b - a$. Using these quantities, we may define CPU utilization as follows:

CPU Utilization for Process X = (Time spent on Process X) / (Time spent on Process X + Idle Time) = $p / (p + q)$, or $p / (b - a)$, expressed as a percentage.

Note: CPU Utilization with the C programming language was almost negligible when compared to the other programming languages studied.

Statistical Significance

In order to test the statistical significance of the results obtained from the algorithms, a paired t test was performed. The significance level chosen was 0.05. The null hypothesis states that the CPU Utilization obtained from the algorithm using one programming language is not significantly smaller than the CPU Utilization obtained from the other programming language. The alternative hypothesis is that the CPU Utilization obtained from the algorithm using one programming language is significantly smaller than the other programming language.

Table 1 below shows the results of the T-test. The p-value is less than 0.05 for C & Java, C# & Jython, Jython & C and Java & Jython but not for Java & C#. Therefore, we cannot reject the null hypothesis for Java & C#. As can be seen, the p-value is less than 0.05 in all tests except Java and C# demonstrating that the differences in CPU Utilization are statistically significant. Figures 1 to 12 depicts the CPU Usage of C, Java, C# and Jython.

Statistical Significance Test		
Comparison	**p-value**	**Statistical Significance**
C & Java	0.000003	Yes
Java & C#	0.499597	No
C# & Jython	0.000084	Yes
Jython & C	0.000001	Yes
Java & Jython	0.000002	Yes
C& C#	0.000001	Yes

Table 1: Statistical Significance of CPU Utilization

4.1 Bubble Sort CPU Utilization for Integer and Float data

Figure 1 shows the CPU Utilization of bubble sort on integer data, and under implementation of C, Java, C# and Jython. C proved to be the leader and Java came in second. This may be because C performs better than Java in low-level numeric computation.

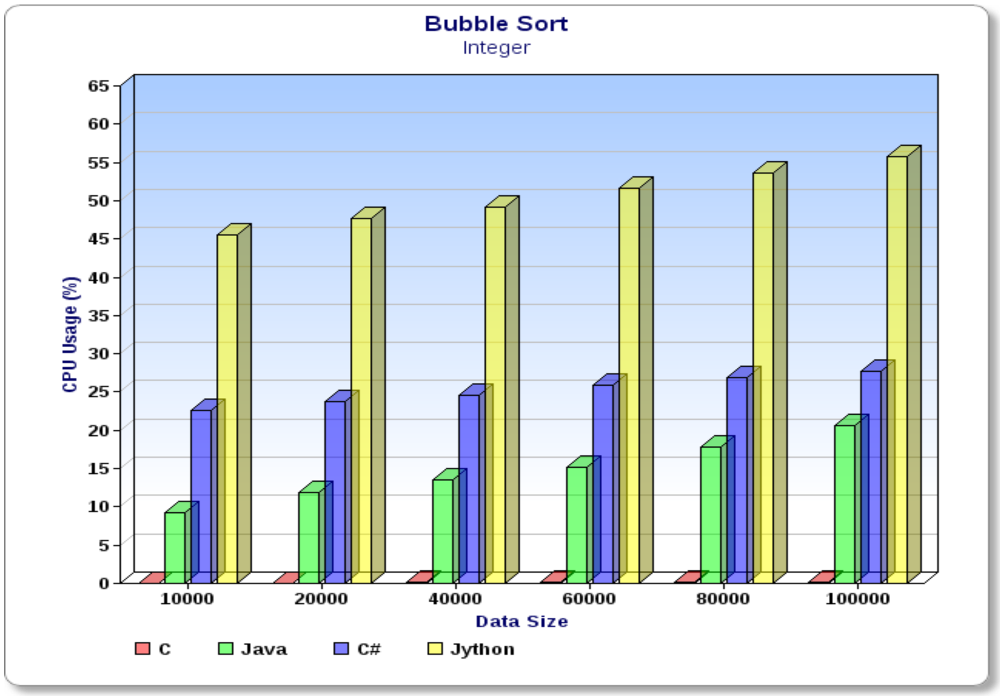

Figure 1: CPU Utilization of Bubble Sort (Integer)

Figure 2 shows that when float data is used. C still proves to be the leader in CPU utilization and Java comes in second.

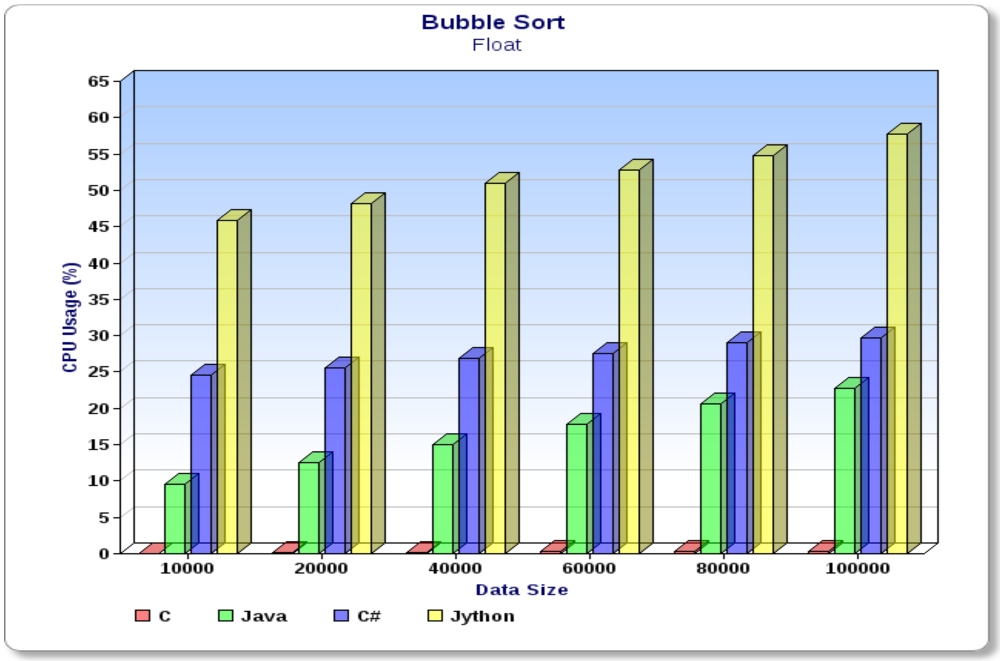

Figure 2: CPU Utilization of Bubble Sort (Float)

In C, program statements are compiled into a lower number of machine instructions. On the other hand, Java program statements, when interpreted by a Java virtual machine, are compiled to a larger volume of byte code that involves more machine instructions, as compared to the statically-compiled programming language, C.

C# runs great on Linux, and Mono as a platform has advanced quite nicely. C# employs less CPU as compared to Jython, and more CPU than C and Java. CPU usage by Jython was the highest, among the other three programming languages. However, the algorithm is completely CPU bound; therefore, Bubble Sort directly measures the performance of the Jython byte code interpreter. Therefore it is slower than executing the equivalent native code. Whenever a long running CPU bound loop is written in Jython, a considerable performance loss is expected.

4.2 Bubble Sort CPU Utilization for String data

Figure 3 shows the CPU Utilization of bubble sort on String data sorting with C, Java, C# and Jython. C was the leader once again with the lowest CPU usage, and Jython was second.

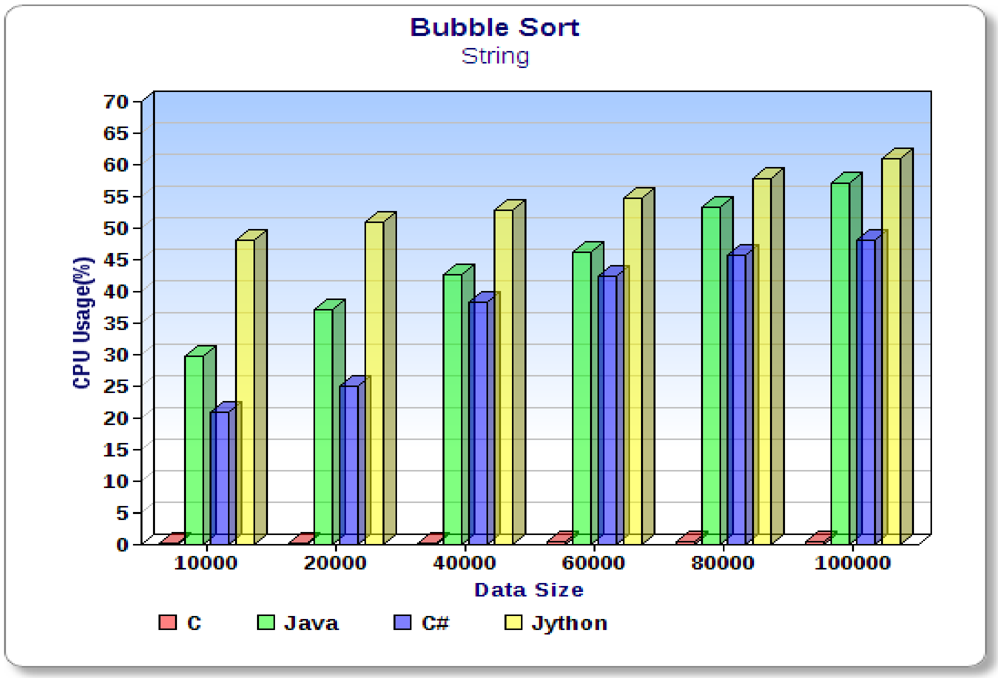

Figure 3: CPU Utilization of Bubble Sort (String)

In the case of the Jython programming language, processor usage is less when compared to developer time. C# and Java were third and fourth respectively. C# runs well on Linux, and Mono as a platform has advanced quite nicely. It employs less CPU as compared to Java. C performs much better than Java in low-level numeric computations. In C, program statements are compiled into a smaller number of machine instructions. Whereas in Java, program statements, when interpreted by a Java virtual machine, are compiled to a larger number of byte codes that involves a larger volume of machine instructions, as compared to the statically compiled programming language C. Just like C, Java's byte codes are also compiled into machine instructions at runtime. String operations are slow, as Java uses immutable, UTF-16 encoded string objects, which requires significant memory, operations, and CPU resources.

4.2.1 Quick Sort CPU Utilization for Integer and Float data

Figures 4 depict that the programming language that required the greatest CPU usage was C#, which makes it the most complex solution for this algorithm. C was once again first with the least CPU usage value.

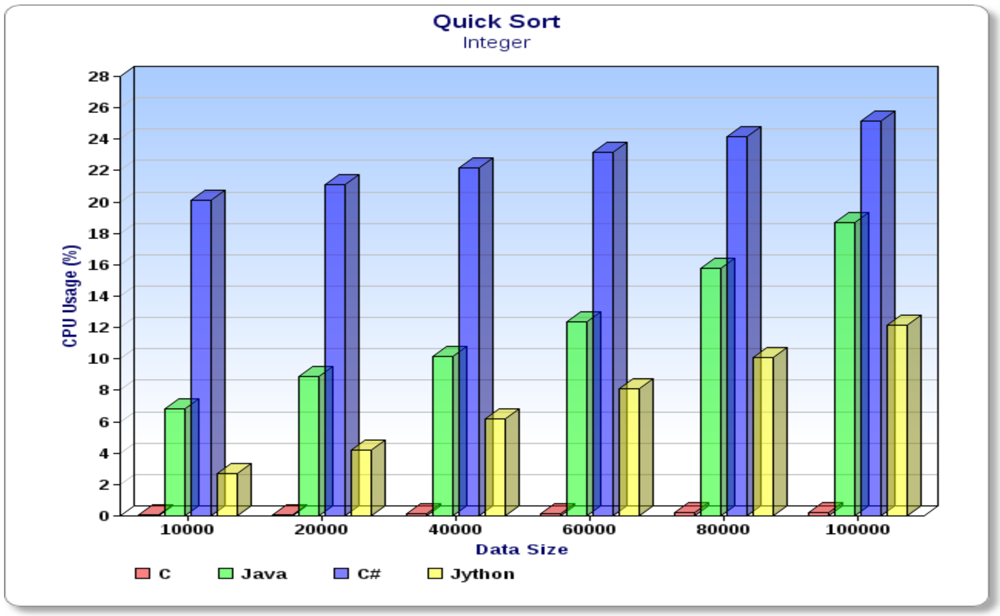

Figure 4: CPU Utilization of Quick Sort (Integer)

Figure 5 shows that C performs much better than Java in low-level numeric computations. In C, program statements are compiled into a smaller number of machine instructions, whereas in Java, program statements, when interpreted by a Java virtual machine, are compiled to a larger number of byte codes that involve more machine instructions.

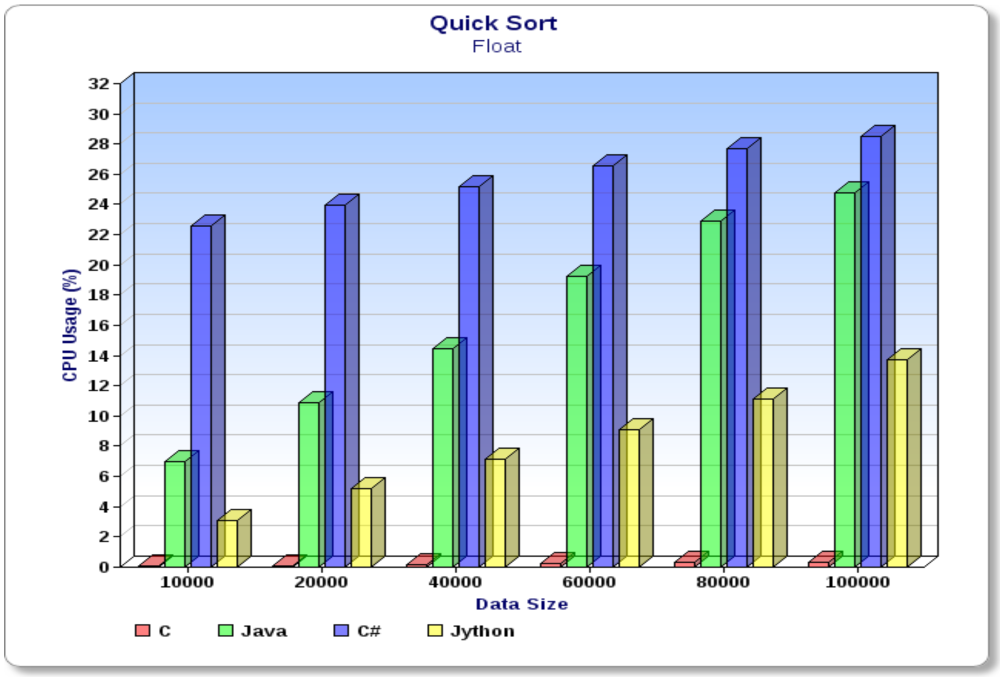

Figure 5: CPU Utilization of Quick Sort (Float)

Java's byte codes are compiled into machine instructions only at runtime. Jython CPU usage was less than Java and C#. Jython is essentially Python written in Java. Java libraries can be used while still coding in Python. Though Jython suffers from some performance penalties too, it is usually of negligible consequence for a project. The advantage of Jython is that CPU usage is economical as compared to developer time.

4.2.2 Quick Sort CPU Utilization for String data.

Figure 6 shows that for Quick Sort using string data, C was the leader once again with the lowest CPU usage, and Jython was second. In the case of Jython, processor usage is less when compared to developer time. C# and Java were third and fourth respectively. C# runs well on Linux, and Mono as a platform is mature. It employs less CPU as compared to Java. C performs much better than Java in low-level numeric computations. In C, program statements are compiled into a smaller number of machine instructions, whereas in Java, program statements, when interpreted by a Java virtual machine, are compiled to a larger number of byte codes that involve many more machine instructions as compared to the statically-compiled programming language C. Similar to C, Java's byte codes are also compiled into machine instructions at runtime. String operations are, to a small degree, slow as Java uses immutable, UTF-16 encoded string objects. This suggests it needs a lot of memory, operation, and CPU. A few operations are a lot more complicated than with ASCII (C).

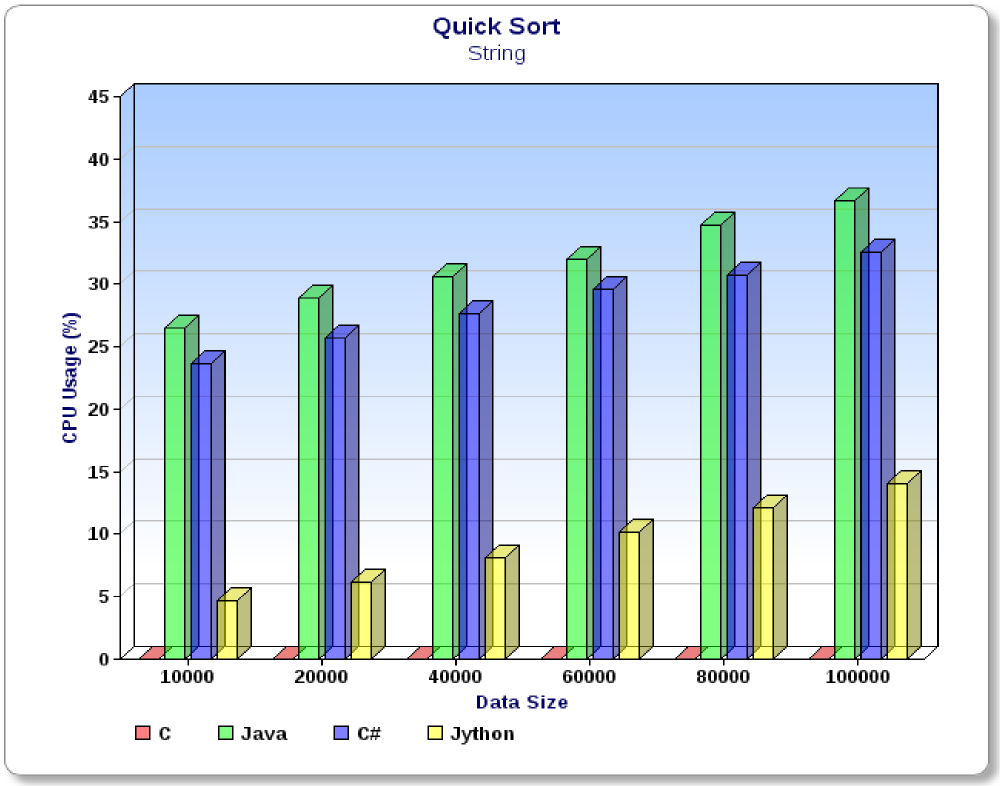

Figure 6: CPU Utilization of Quick Sort (String)

4.3 Linear Search

4.3.1 Linear Search CPU Utilization for Integer, Float and String data

As shown in figures 7, 8, and 9 for linear search with integer, float and string, C was once again first in terms of lowest CPU usage. Jython was second, owing to the fact that it is a combination of Java and Python. Java libraries can be used while still coding in Python. Though Jython suffers from some performance penalties, it is usually of negligible consequence for a project. The benefit is that CPU usage is economical as compared to developer time. C# and Java were third and fourth respectively. However, the CPU usage of Java is higher than a statically-compiled programming language like C and similar to

Just-in-Time compiled languages like C#, because of its similar syntax structure. But Mono as a platform works quite well for C# in terms of CPU usage.

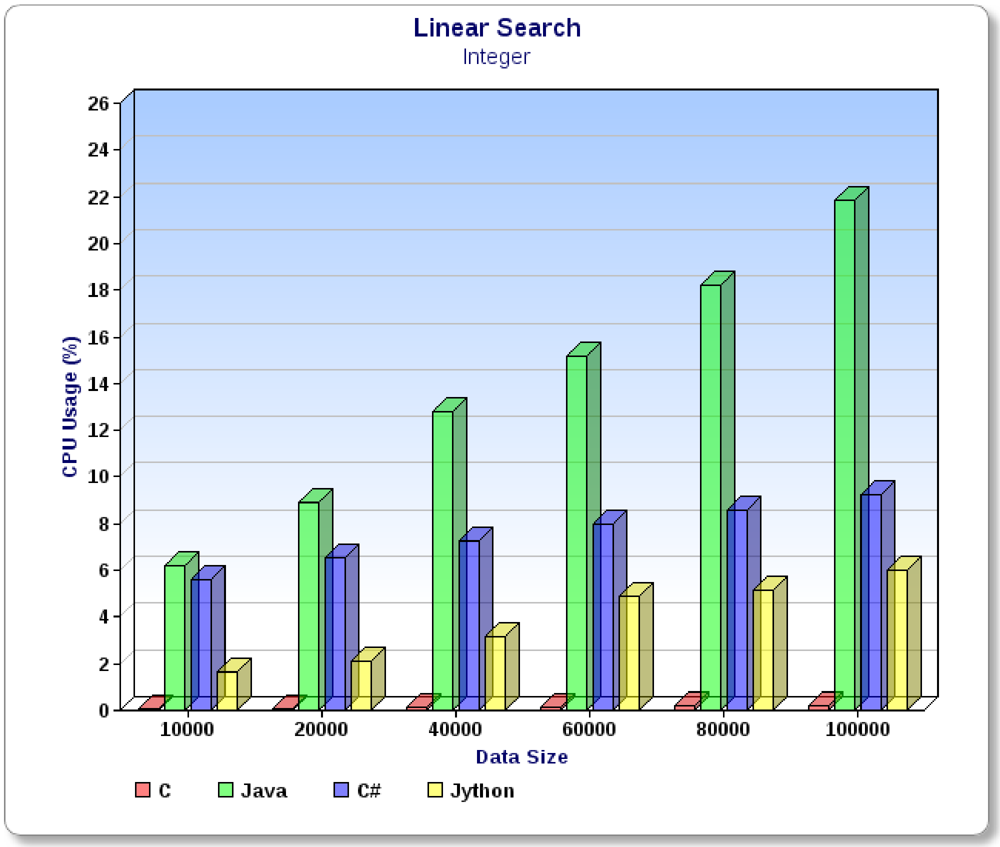

Figure 7: CPU Utilization of Linear Search (Integer)

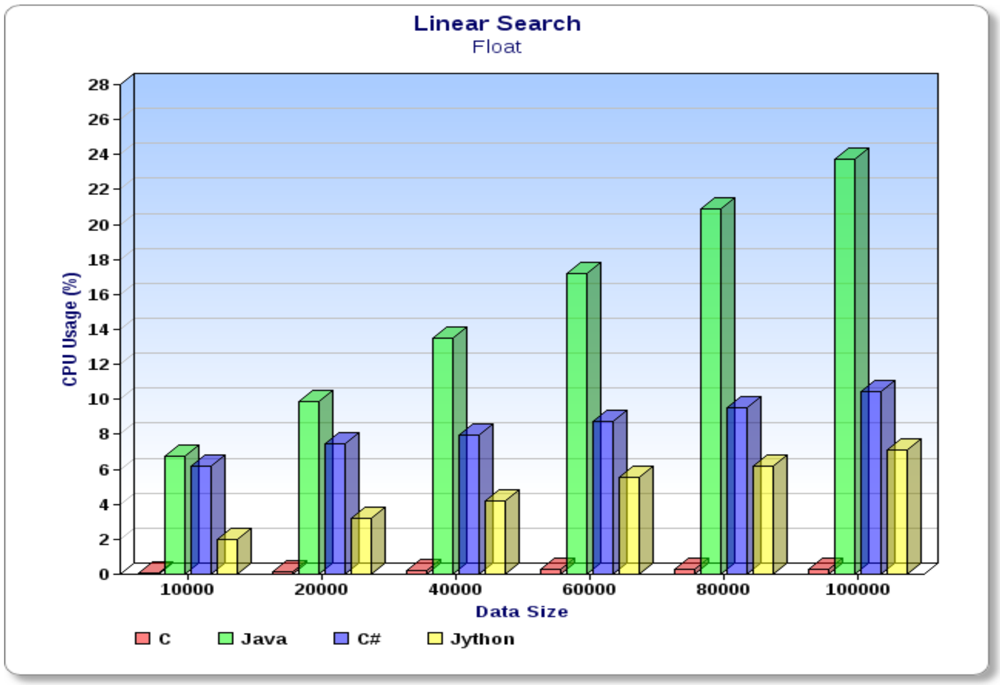

Figure 8: CPU Utilization of Linear Search (Float)

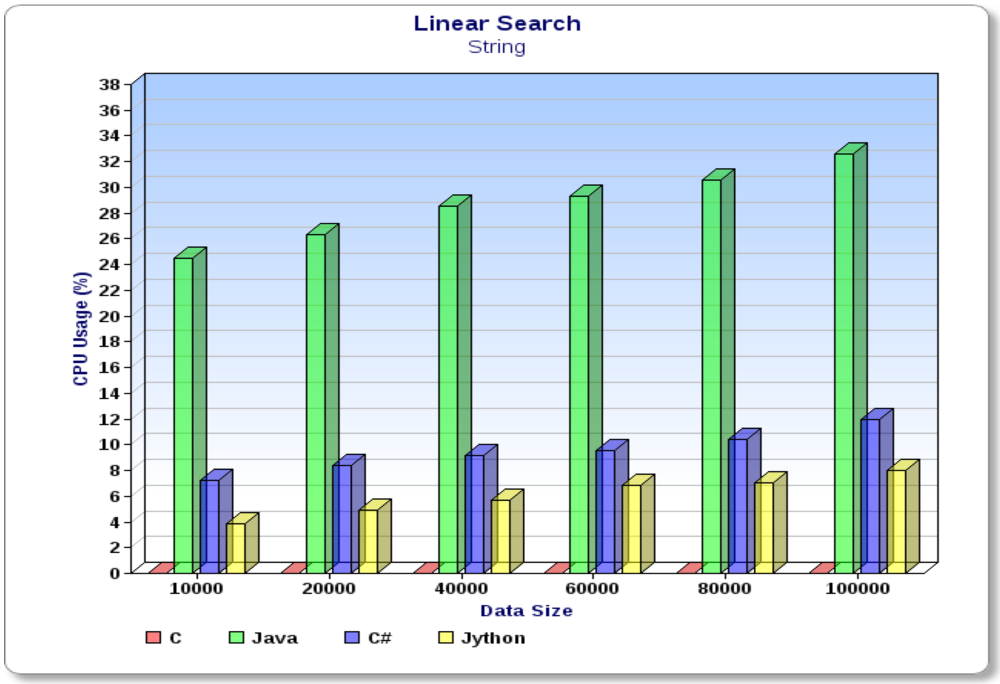

Figure 9: CPU Utilization of Linear Search (String)

4.4 Binary Search

4.4.1 Binary Search CPU Utilization for Integer, Float and String data

As evident from figures 10, 11, and 12 for CPU utilization for binary search, C was the leader with almost negligible CPU usage as compared to the other three programming languages (C#, Java, and Jython). In second place was C#, which runs nicely on Mono. C# utilized less CPU as compared to Java, and used more CPU than C and Jython. It appears that, for this case, Mono was not mature enough to contend against a seasoned veteran programing language like Java. The CPU usage of Jython was less than Java and C#, since Jython is Python written in Java. Java libraries can be used, but still are coded in Python. The performance penalties associated with Jython are of typically very little to no consequence for a project.

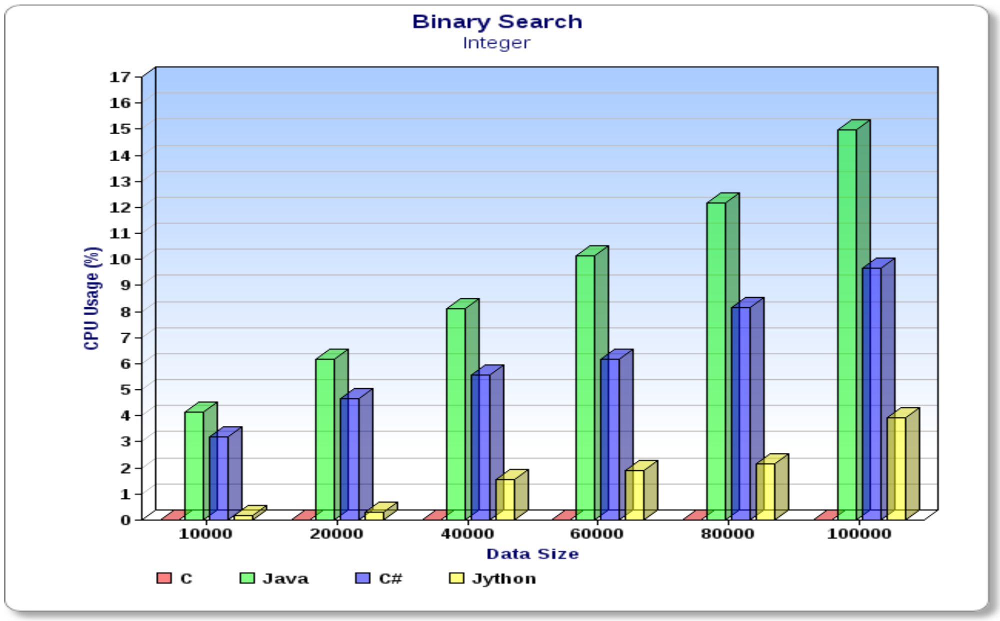

Figure 10: CPU Utilization of Binary Search (Integer)

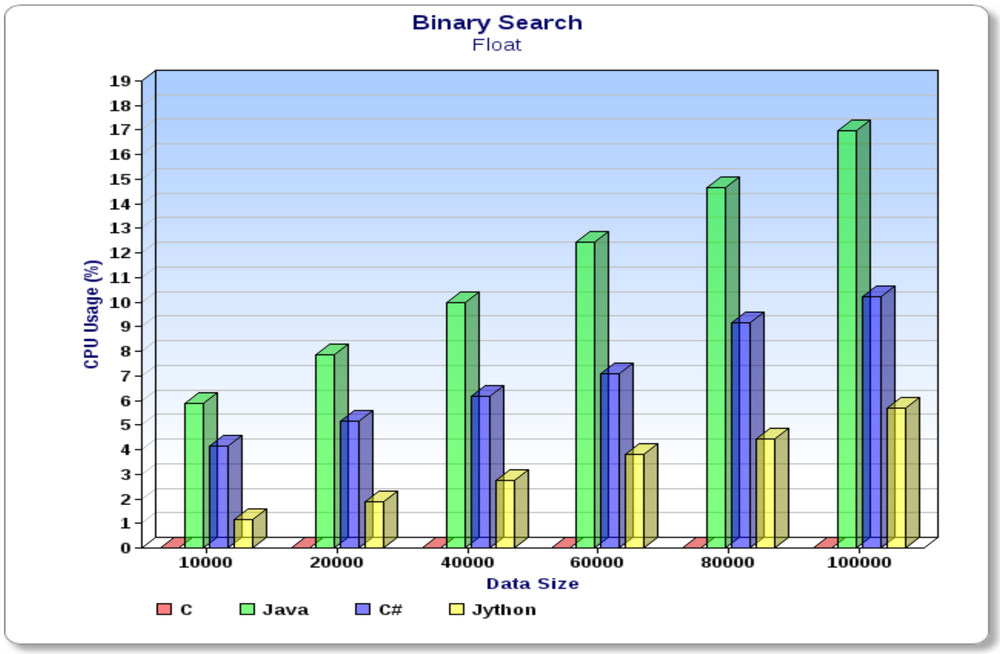

Figure 11: CPU Utilization of Binary Search (Float)

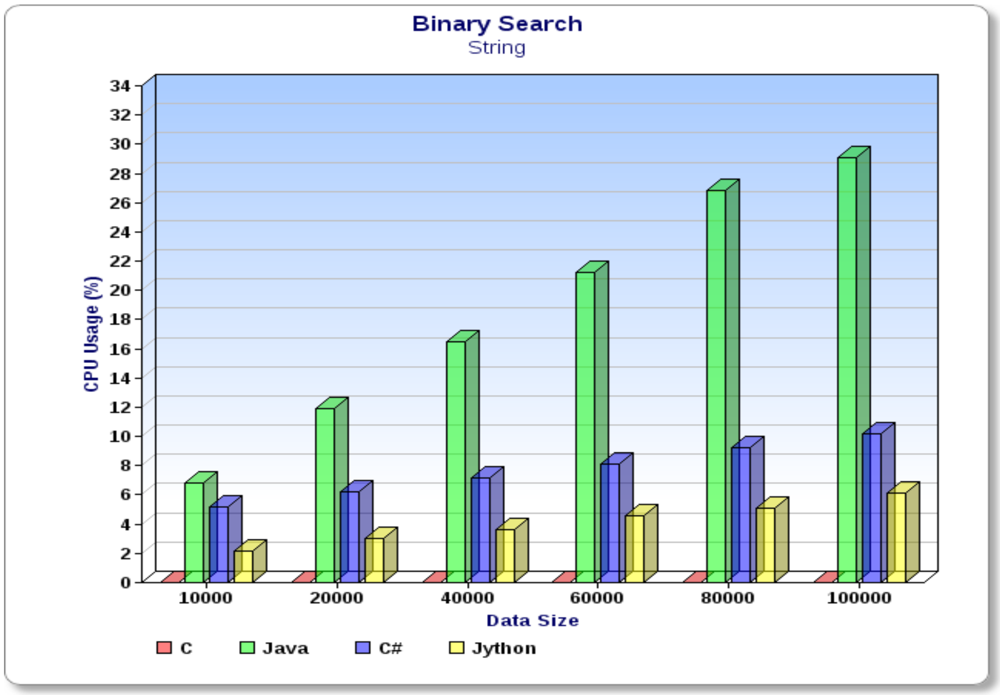

Figure 12: CPU Utilization of Binary Search (String)

Chapter 5

RESULTS AND DISUSSION: MEMORY USAGE

In this chapter, we will compare the memory usage for Bubble Sort, Quick Sort, Linear Search and Binary Search when implemented using C, Java, C# and Jython.

Memory Usage refers to the amount of memory used by a process. Let us say process X uses q bytes of memory. The total available memory is r using these quantities; we may define Memory Usage as follows:

$$\text{Memory Usage for process X} = \frac{\text{Memory used by Process X}}{\text{Total memory available}} = \frac{q}{r}$$

Memory Usage for process X = (Memory used by Process X) / (Total memory available) = q / r, expressed as a percentage. Total memory available = 128 GB = 1.374e+11 bytes

5.1 Bubble Sort

Statistical Significance

In order to test the statistical significance of the results obtained from the algorithms, a paired t test was performed. The significance level chosen was 0.05. The null hypothesis states that the Memory Usage obtained from the algorithm using one programming language is not significantly smaller than the Memory usage obtained from the other programming langue. The alternative hypothesis is that the Memory usage obtained from the algorithm using one programming language is significantly smaller than the other programming langue.

Table 2 below shows the results of the T-test. The p-value is less than 0.05 for C & Java, C# & Jython, Jython & C, Java & C#., Java & Jython and Java & C#. Therefore, we reject the null hypothesis in all cases. As can be seen, the p-value is less than 0.05 in all tests demonstrating that the differences in Memory usage are statistically significant. Figure 13 to 24 depicts the Memory Usage of C, Java, C# and Jython.

Statistical Significance Test		
Comparison	**p-value**	**Statistical Significance**
C & Java	0.000009	Yes
Java & C#	0.004026	Yes
C# & Jython	0.000001	Yes
Jython & C	0.000001	Yes
Java & Jython	0.000636	Yes
C& C#	0.002268	Yes

Table 2: Statistical Significance of Memory Usage

5.1.1 Evaluation of Bubble Sort

Figures 13 through 15 show the memory usage of Bubble Sort (integer, float, and string) and tables 3, 4, and 5 depict the data collected from the algorithms. C was first, with low memory consumption, and Jython was second. There was a slight relative increase in

Jython's memory usage with increasing data size. Because it is constructing byte code, it is spending significant resources in translation rather than in execution. Third was C#, again with a value close to Java, since their processing styles are similar. Java was the programming language that used the greatest amount of memory, because Java programs use automatic garbage collection, and objects are larger in Java since all objects are allocated on the heap, each having a virtual table including support for synchronization primitives.

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00043	0.00360	0.00085	0.00185
20000	0.00046	0.00630	0.00320	0.00185
40000	0.00053	0.01250	0.00450	0.00185
60000	0.00059	0.01680	0.00660	0.00185
80000	0.00064	0.02550	0.00930	0.00185
100000	0.00070	0.03100	0.01130	0.00185

Table 3: Result for Bubble Sort (Integer)

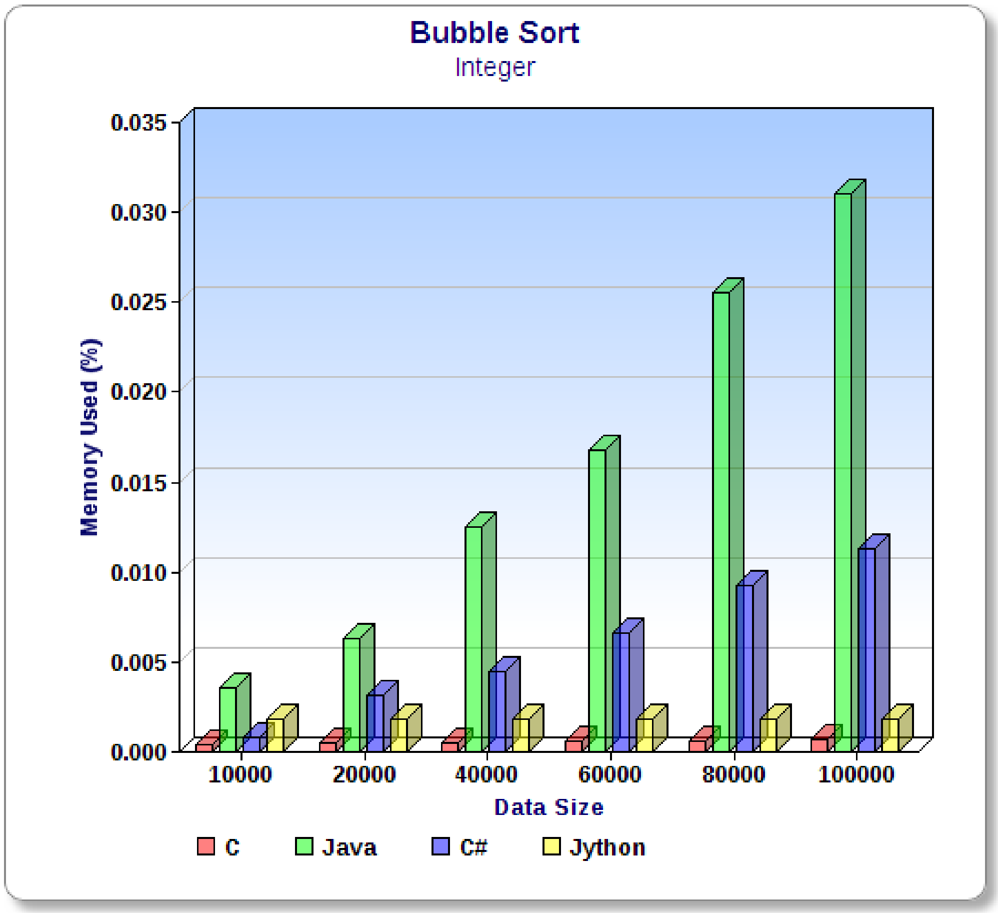

Figure 13: Memory Usage of Bubble Sort (Integer)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00047	0.00440	0.00155	0.00187
20000	0.00050	0.00800	0.00261	0.00187
40000	0.00057	0.01580	0.00279	0.00188
60000	0.00062	0.02040	0.00296	0.00188
80000	0.00068	0.02970	0.00325	0.00188
100000	0.00074	0.03540	0.00349	0.00188

Table 4: Result for Bubble Sort (Float)

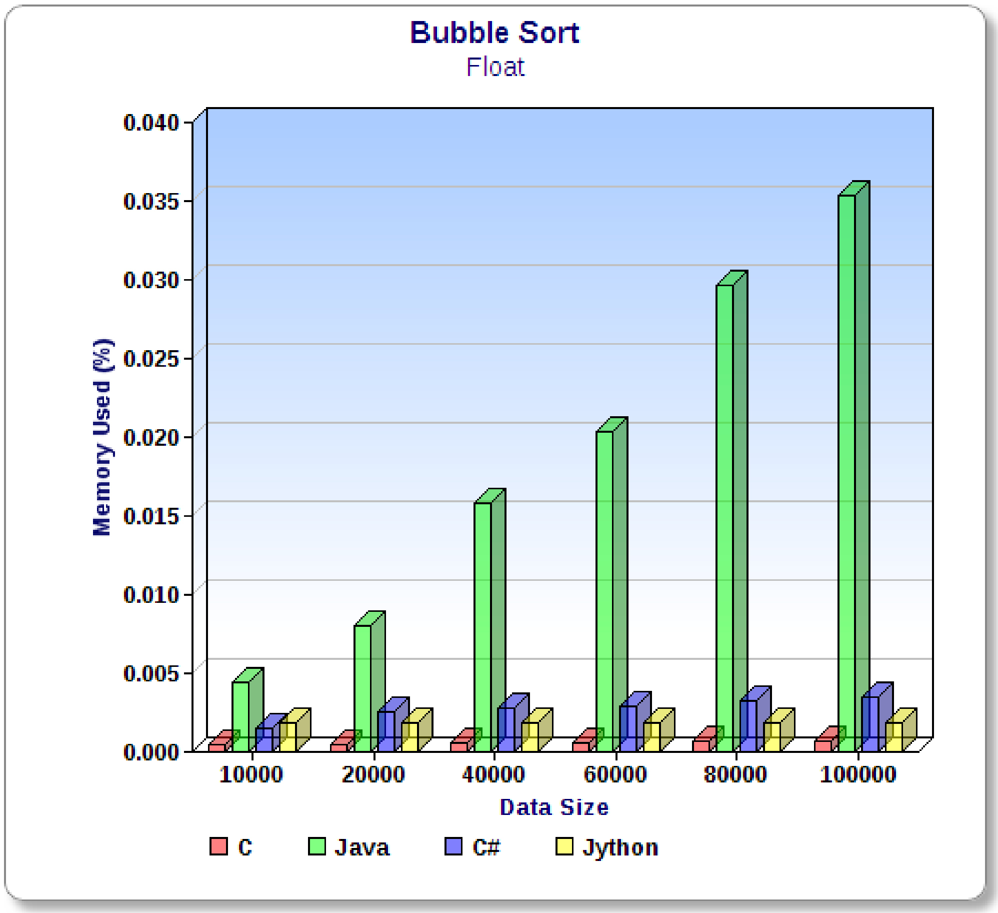

Figure 14: Memory Usage of Bubble Sort (Float)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00073	0.00319	0.00109	0.00188
20000	0.00103	0.00594	0.00370	0.00188
40000	0.00161	0.01090	0.00600	0.00188
60000	0.00219	0.01660	0.00939	0.00188
80000	0.00277	0.02129	0.01134	0.00188
100000	0.00335	0.02690	0.01311	0.00189

Table 5: Result for Bubble Sort (String)

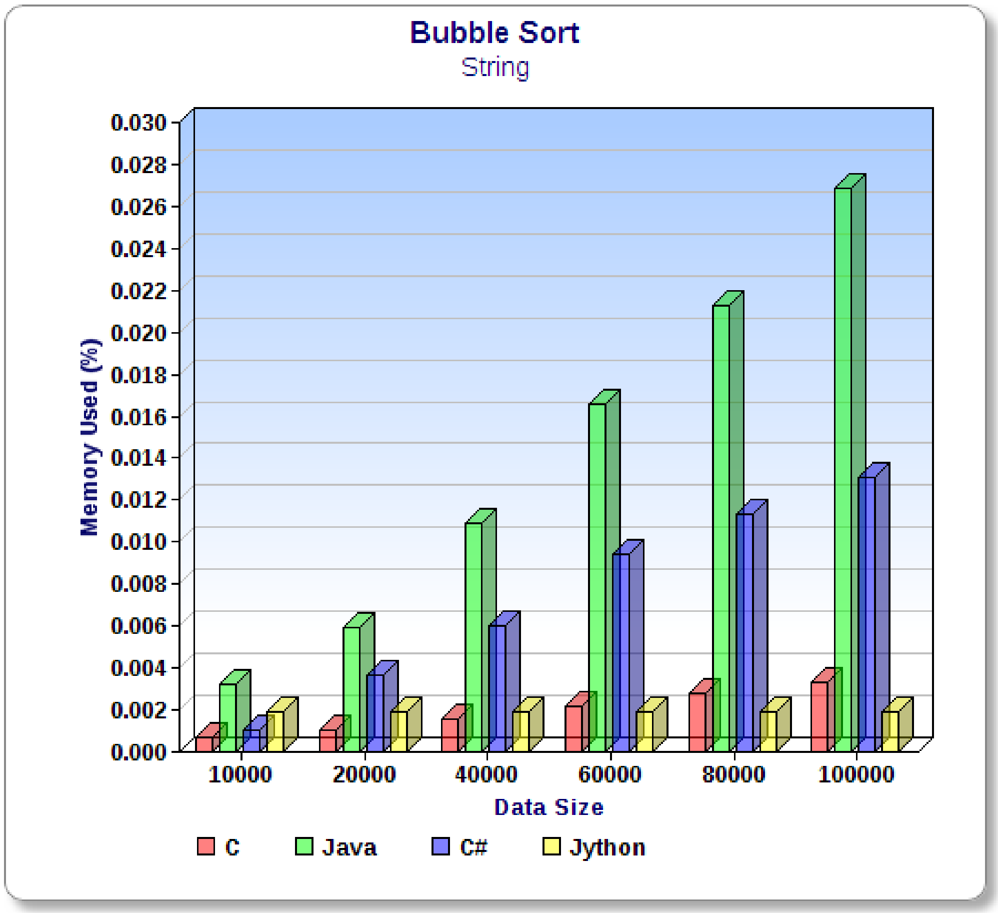

Figure 15: Memory Usage of Bubble Sort (String)

5.2 Quick Sort

5.2.1 Evaluation of Quick Sort

Tables 6, 7, and 8 depict the data collected from the algorithm and Figures 16 through 18 depict that the memory usage of C is less, as compared to the other programming languages. Java has the highest memory usage. Whereas Jython was again second, there was a slight increase in Jython's memory usage with increasing data size, because of the construction of byte code. It is spending all the resources in translation rather than execution. Third was C#, although Java and C# were actually quite similar. But C#/Mono won on memory usage over Java. Mono also has automatic garbage collection functionality like Java, it removes unused objects from the heap. This increases programmer productivity; however, if thousands of objects are created, additional work of the garbage collector will be required, resulting in more memory consumption, slowing the application. Java's implementation was the last.

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00043	0.00360	0.00055	0.00185
20000	0.00047	0.00630	0.00261	0.00185
40000	0.00053	0.01250	0.00270	0.00185
60000	0.00058	0.01680	0.00279	0.00185
80000	0.00064	0.02450	0.00297	0.00185
100000	0.00070	0.03060	0.00308	0.00185

Table 6: Result for Quick Sort (Integer)

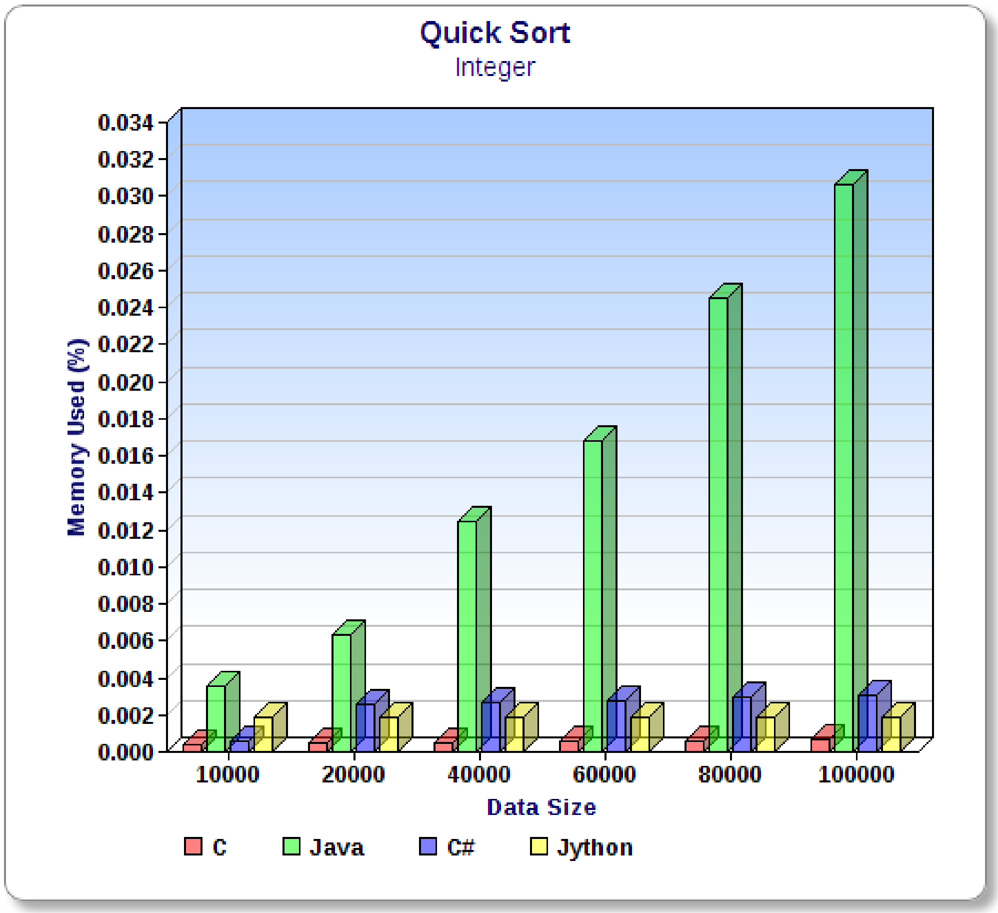

Figure 16: Memory Usage of Quick Sort (Integer)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00047	0.00490	0.00085	0.00186
20000	0.00050	0.00900	0.00421	0.00186
40000	0.00057	0.01770	0.00451	0.00186
60000	0.00062	0.02550	0.00650	0.00186
80000	0.00068	0.03500	0.00850	0.00187
100000	0.00074	0.04400	0.00980	0.00187

Table 7: Result for Quick Sort (Float)

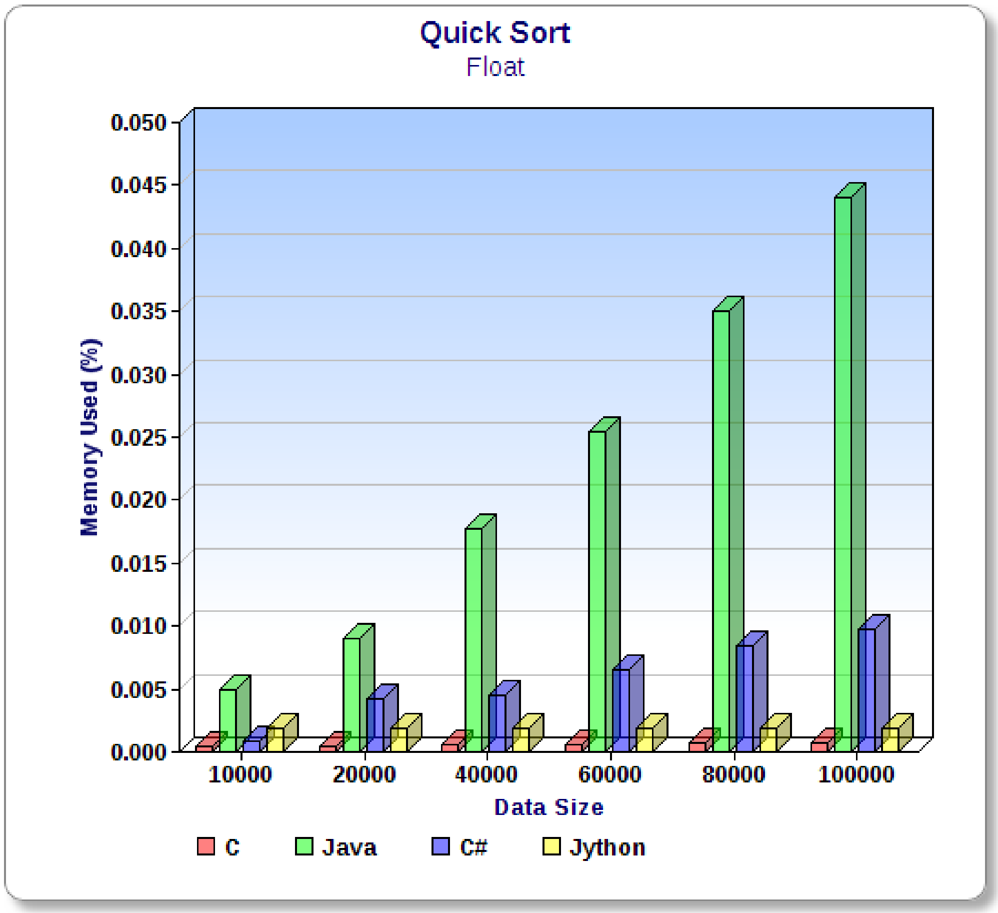

Figure 17: Memory Usage of Quick Sort (Float)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00073	0.00140	0.00099	0.00187
20000	0.00103	0.00660	0.00450	0.00187
40000	0.00160	0.00990	0.00680	0.00187
60000	0.00219	0.01700	0.00900	0.00188
80000	0.00278	0.02200	0.01000	0.00188
100000	0.00336	0.03100	0.01234	0.00188

Table 8: Result for Quick Sort (String)

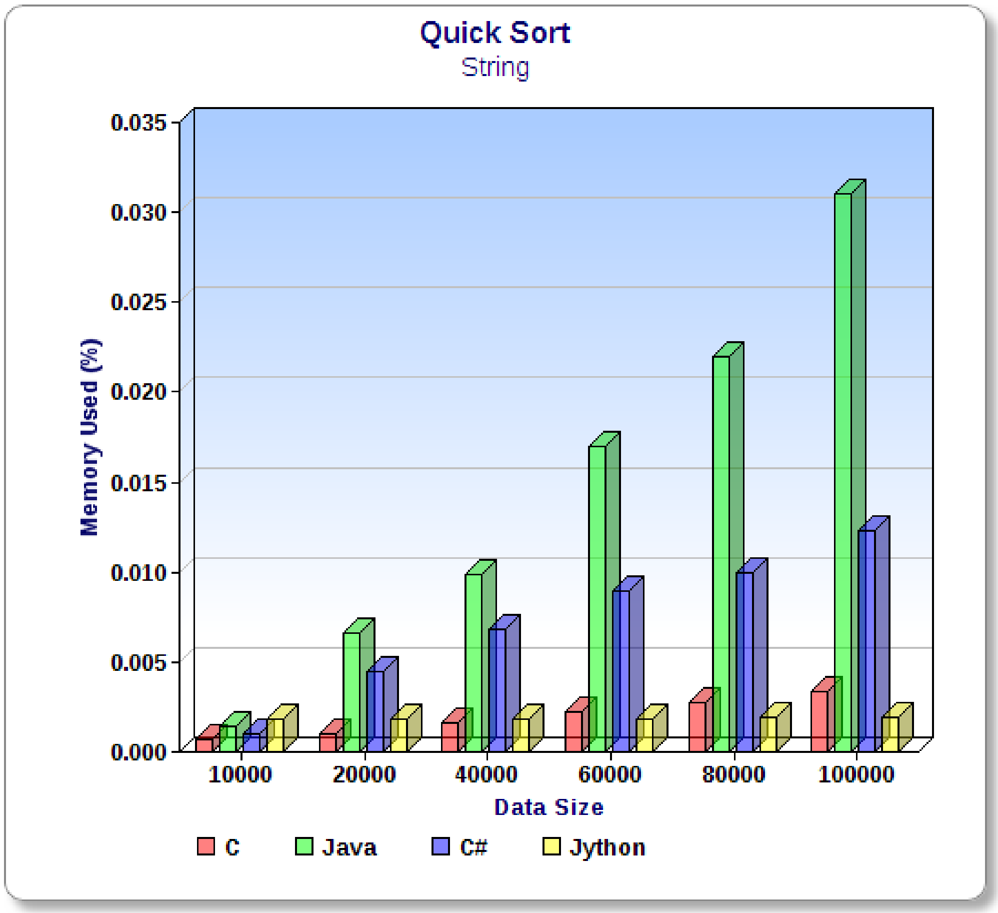

Figure 18: Memory Usage of Quick Sort (String)

5.3 Linear Search

5.3.1 Evaluation of Linear Search

Tables 9, 10, and 11 depict the data collected from the algorithms, and Figures 19 through 21 shows that the result of this algorithm was, as expected. C implementation of the algorithm was the leader with the lowest memory consumption. Jython, being stable at its second position for the same reason, showed a slight increase in memory usage with increasing data size, because of its constructing byte code, and because it is spending all its resources in translation rather than in execution. C#/Mono was third, since in Mono each object has an overhead of 8 bytes that comprises 4 bytes for the sync block and 4 bytes for the type. Every time an object is allocated, a garbage collector is finding space for the object. In case of unavailability of space, the garbage collector will look for and delete unreferenced objects. In the event there is an additional object allocation, then the garbage collector is required to release the objects. The memory consumption of Java is the highest, placing last of the four programming languages.

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00043	0.00223	0.00054	0.00181
20000	0.00046	0.00355	0.00183	0.00181
40000	0.00053	0.00591	0.00210	0.00182
60000	0.00057	0.00794	0.00228	0.00182
80000	0.00062	0.01030	0.00253	0.00182
100000	0.00069	0.01460	0.00283	0.00182

Table 9: Result for Linear Search (Integer)

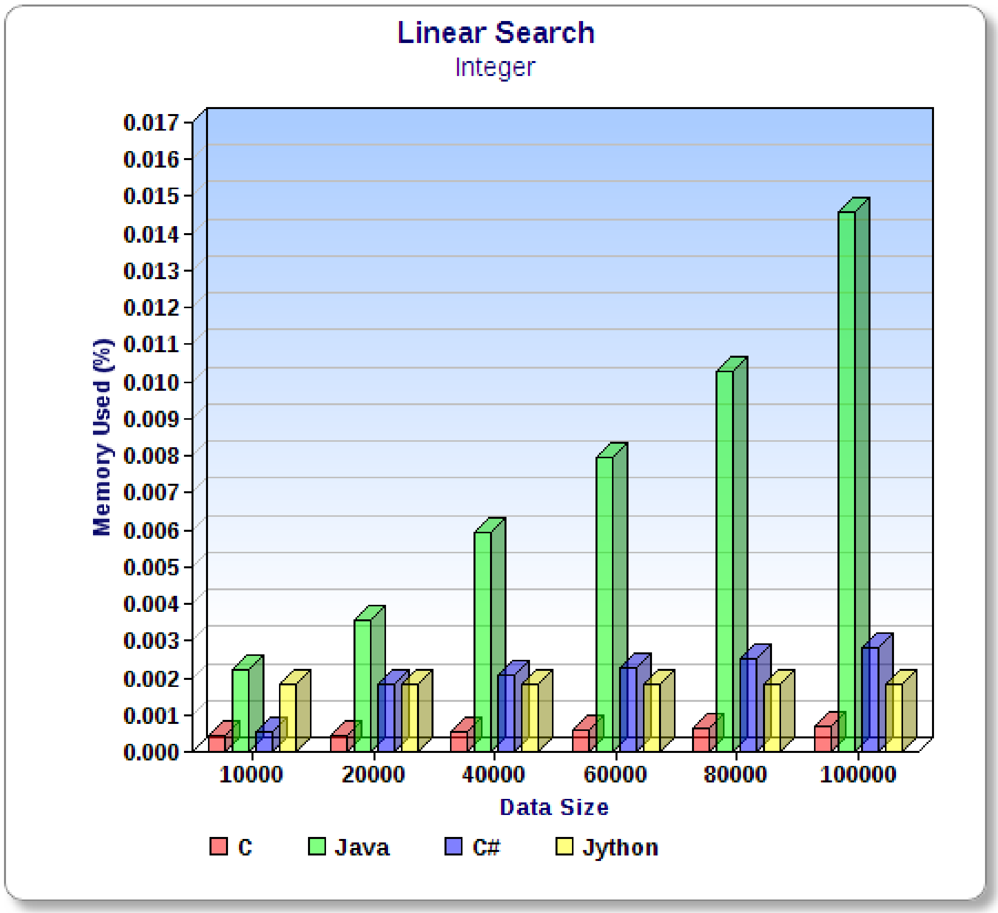

Figure 19: Memory Usage of Linear Search (Integer)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00047	0.00270	0.00080	0.00183
20000	0.00050	0.00640	0.00210	0.00183
40000	0.00056	0.01030	0.00350	0.00184
60000	0.00062	0.01368	0.00550	0.00184
80000	0.00068	0.01600	0.00650	0.00184
100000	0.00074	0.01848	0.00850	0.00184

Table 10: Result for Linear Search (Float)

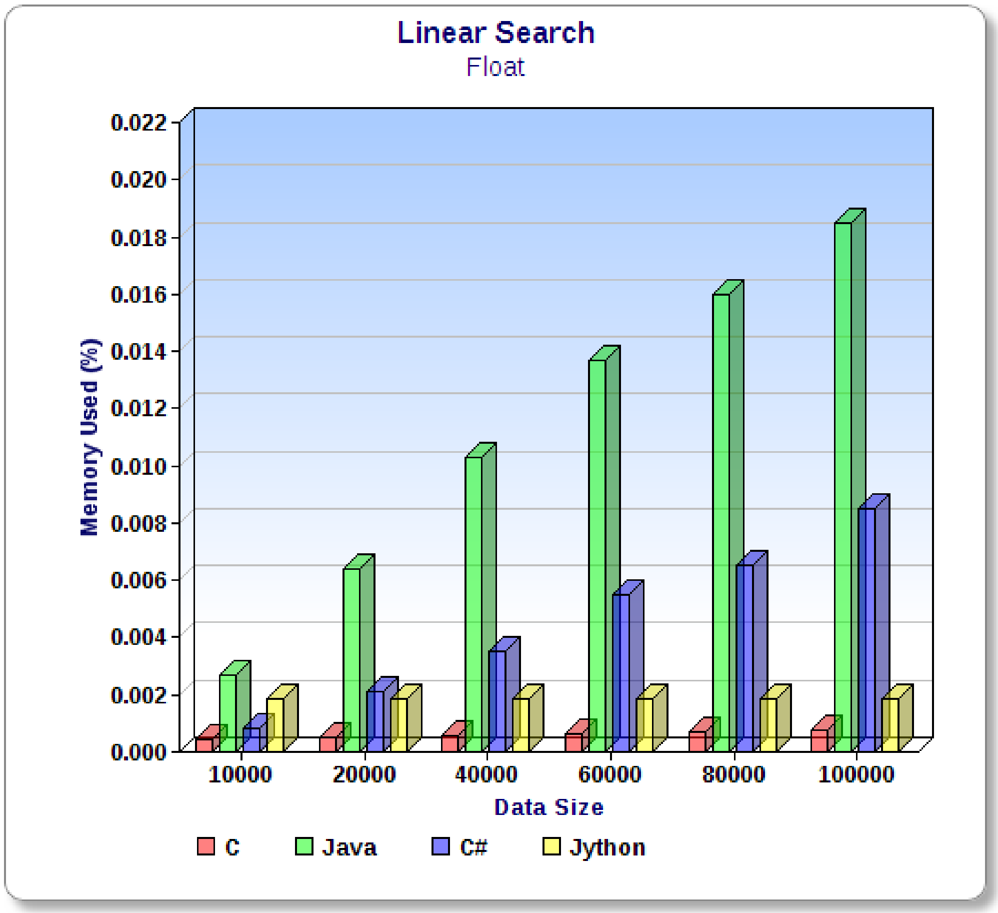

Figure 20: Memory Usage of Linear Search (Float)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00070	0.00177	0.00090	0.00184
20000	0.00100	0.00363	0.00230	0.00184
40000	0.00150	0.00634	0.00470	0.00184
60000	0.00210	0.00872	0.00680	0.00184
80000	0.00270	0.01040	0.00840	0.00184
100000	0.00330	0.01410	0.01060	0.00184

Table 11: Result for Linear Search (String)

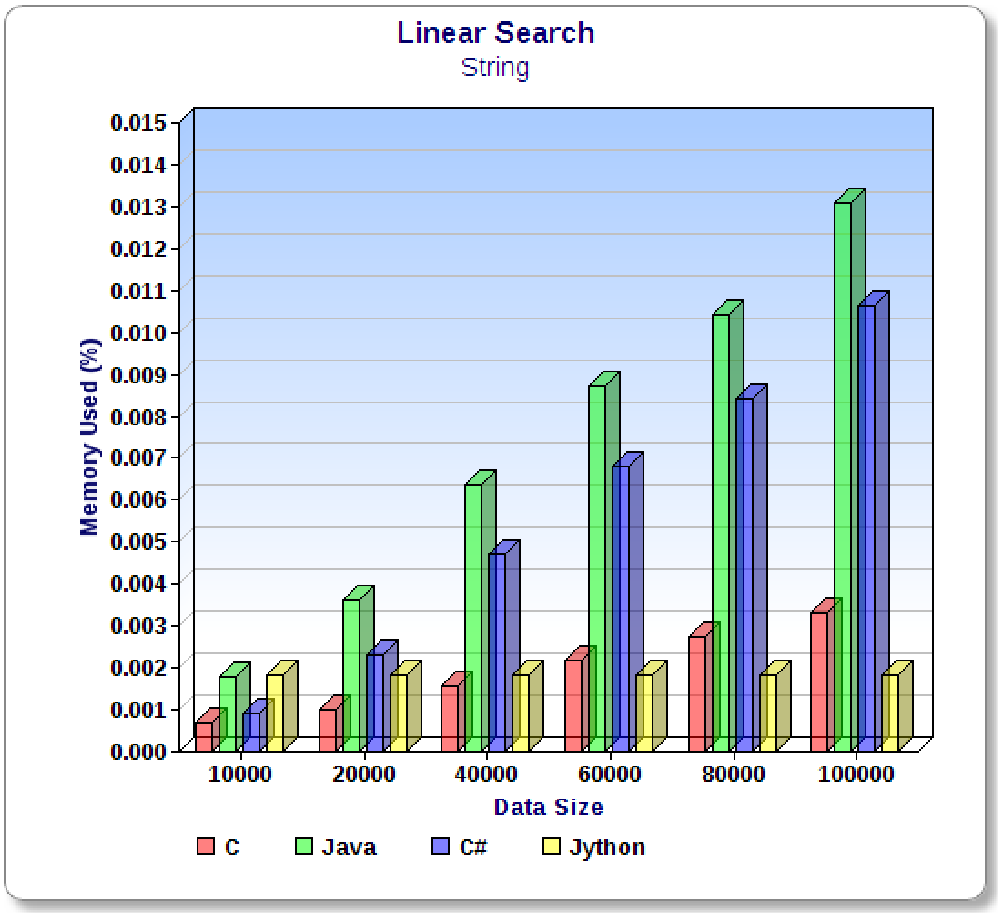

Figure 21: Memory Usage of Linear Search (String)

5.4 Binary Search

5.4.1 Evaluation of Binary Search

Tables 12, 13 and 14 depict the data collected from the algorithm. Figures 22 through 24 shows that the most efficient language for this algorithm was the compiled programming language C which had the lowest memory usage. In second place came Jython, whose memory was nearly consistent. Third and fourth were C# and Java respectively. C# runs nicely on the UNIX operating system, since Mono as a platform is mature. It utilizes less memory as compared to Java. In this case, Mono is not only mature enough to compete with the Java, in some cases, it can even supersede it. Java posted the highest memory usage because Java programs use automatic garbage collection, and objects are larger in java, as all objects in Java are allocated on a heap, and have a virtual table along with the support for synchronization primitives.

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00043	0.00144	0.00054	0.00174
20000	0.00043	0.00265	0.00127	0.00174
40000	0.00046	0.00434	0.00152	0.00176
60000	0.00053	0.00675	0.00170	0.00177
80000	0.00056	0.00918	0.00195	0.00178
100000	0.00061	0.01042	0.00224	0.00180

Table 12: Result for Binary Search (Integer)

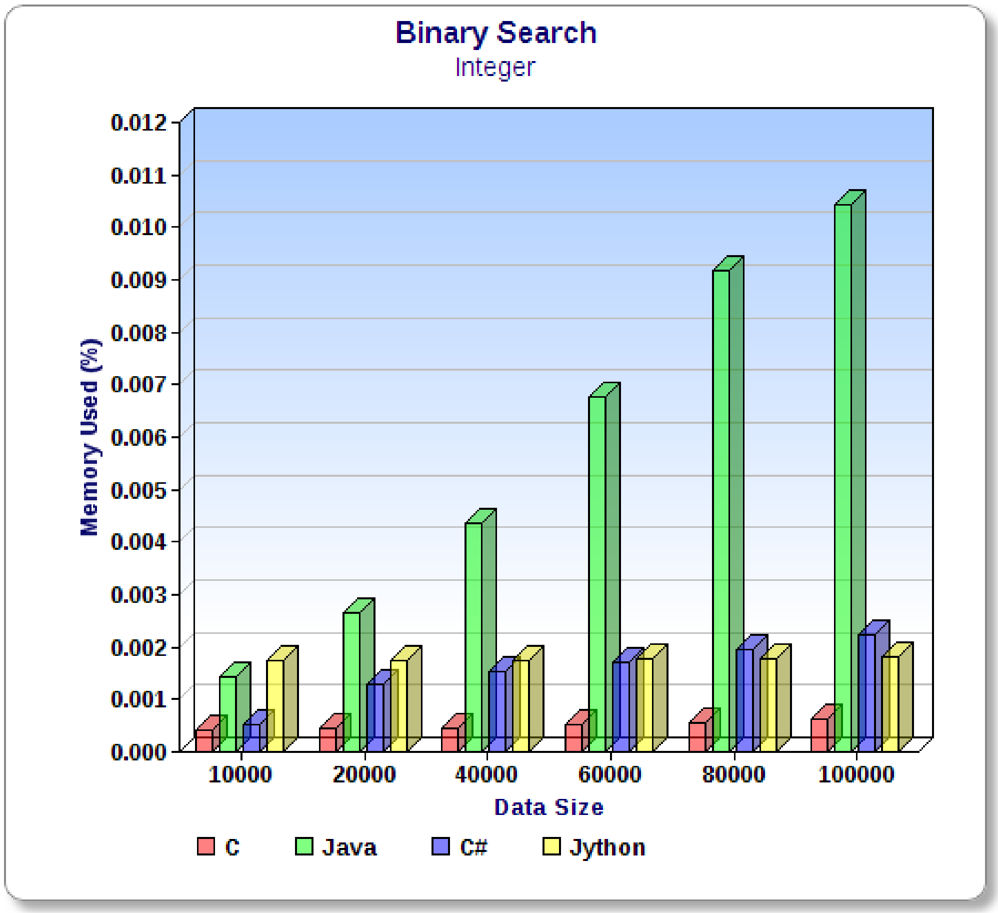

Figure 22: Memory Usage of Binary Search (Integer)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00044	0.00191	0.00054	0.00175
20000	0.00046	0.00328	0.00164	0.00175
40000	0.00053	0.00628	0.00292	0.00177
60000	0.00057	0.00913	0.00359	0.00178
80000	0.00062	0.01355	0.00432	0.00179
100000	0.00065	0.01594	0.00617	0.00181

Table 13: Result for Binary Search (Float)

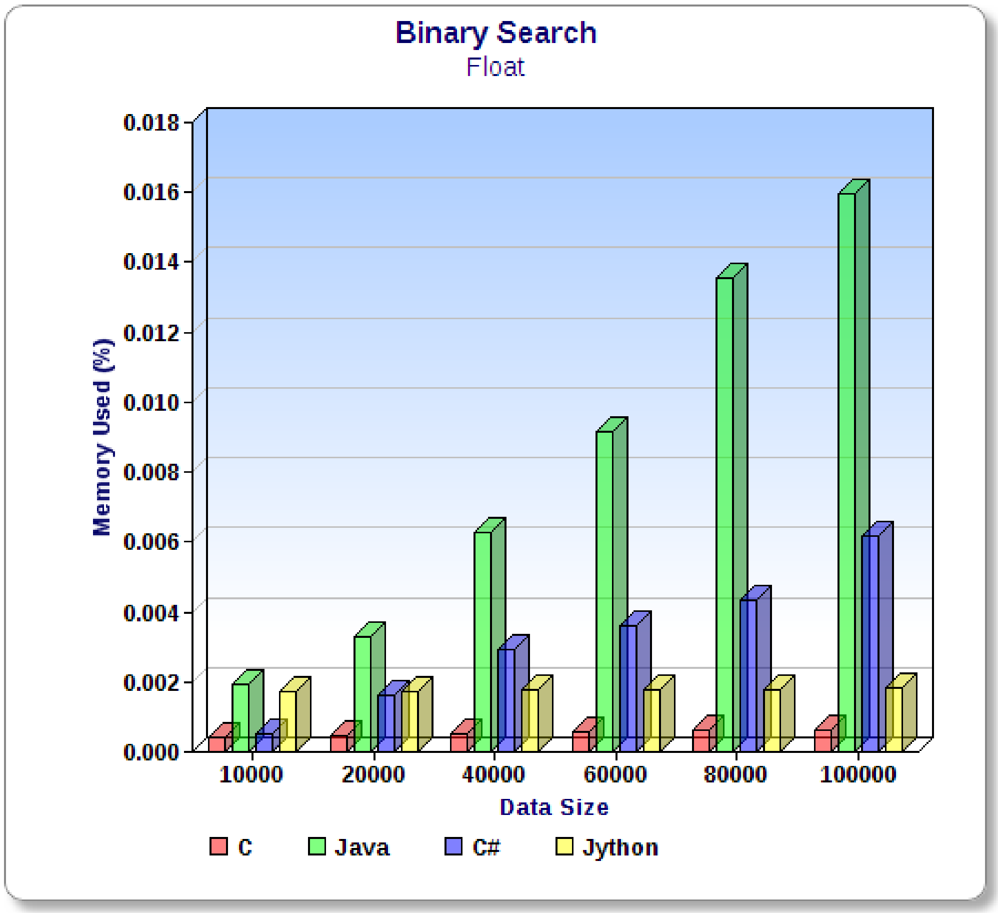

Figure 23: Memory Usage of Binary Search (Float)

Data Size	C(%)	Java(%)	C#(%)	Jython(%)
10000	0.00066	0.00155	0.00069	0.00176
20000	0.00085	0.00271	0.00182	0.00176
40000	0.00143	0.00540	0.00328	0.00178
60000	0.00194	0.00790	0.00556	0.00179
80000	0.00253	0.00920	0.00699	0.00180
100000	0.00299	0.01210	0.00844	0.00182

Table 14: Result for Binary Search (String)

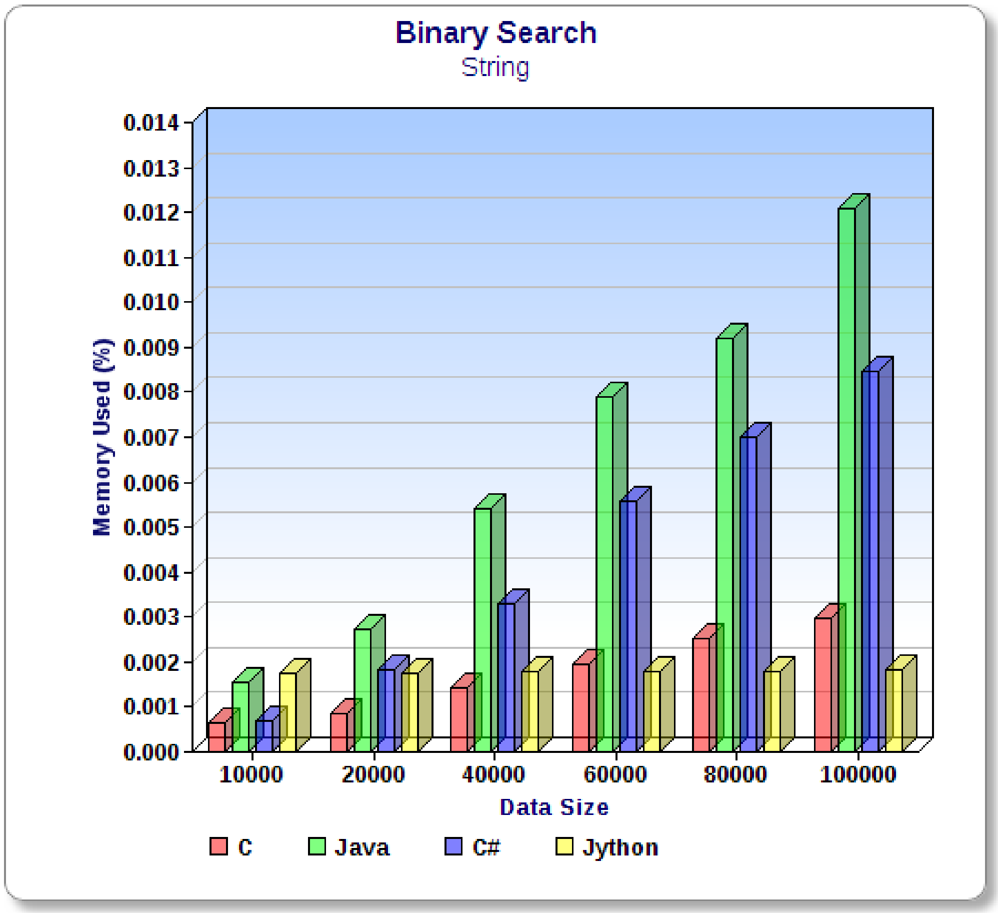

Figure 24: Memory Usage of Binary Search (String)

Chapter 6

RESULTS AND DISCUSSION: EXECUTION TIME IN A DISTRIBUTED SYSTEM USING TCP SOCKETS

First, the data to be sorted was generated on the server side by specifying the data size and number of clients from the command prompt, and then the server waited for the clients to connect. Once all the clients were connected, the server generated the data and sent data to the client. Once the data was sent to the clients, the timer started, and then stopped only after receiving the sorted data from all the clients. The server is the master and the clients are the workers in this usage.

Let us say the timer is started at X millisecond and is stopped at Y milliseconds. Using these quantities, we may define Run time as follows:

Difference (Z) = end time (Y) – start time(X)

6.1 Bubble Sort (Integer)

6.1.1 Evaluation of Bubble Sort (Integer)

Tables 15 through 18 depict the data obtained from the algorithm and also the result of ANOVA tests.

ANOVA provides a statistical test of whether or not the means of several groups are equal, and therefore generalizes the *t*-test to more than two groups.

Figures 25 through 28 show that Java emerged as the leader, and proved to be the least complex solution. Its elapsed time was the lowest with the increasing data size. Second was C#, but its values were close to Java since its execution structure is similar. The performances of C# and Java were very close, with Java having a slight edge. Java comprises built-in socket libraries, eliminating the requirement of third party socket operations. Third was the C implementation. Jython was the slowest since it is an interpreted programming language with slow execution speed. Jython's start-up time is slow, but the ease of programming makes Jython desirable. Since Jython is interpreted, it is not designed to be used for CPU bound applications.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	314	296	250	13782
20000	1331	884	960	55778
40000	5453	3259	3805	223309
60000	10371	7316	8570	501795
80000	22096	12991	15235	926146
100000	34689	20356	23815	137193
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.015				

Table 15: ANOVA results for Bubble Sort (Integer 1 client)

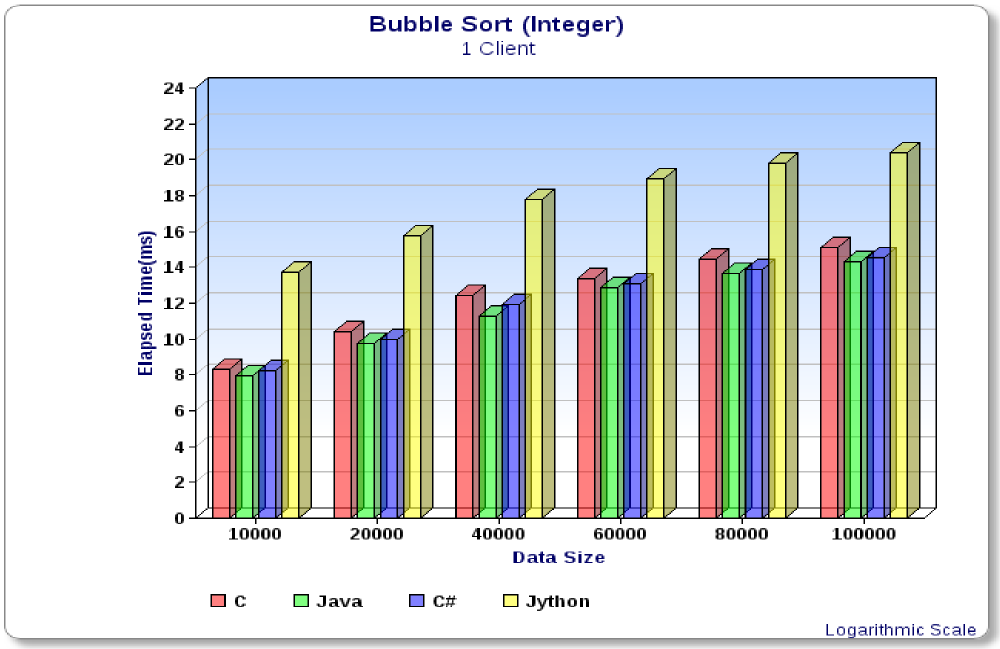

Figure 25: Bubble Sort (Integer) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	157	124	125	7063
20000	665	461	480	27897
40000	2726	1729	1902	111460
60000	5185	2862	4285	250299
80000	10934	6594	7617	447302
100000	17143	11078	11907	697992
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0081				

Table 16: ANOVA results for Bubble Sort (Integer 2 clients)

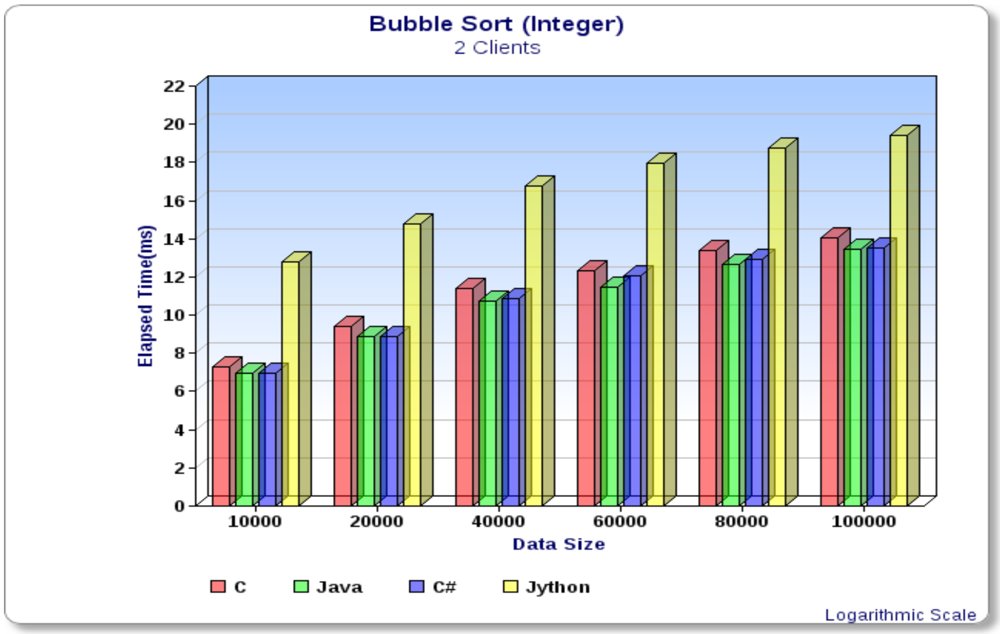

Figure 26: Bubble Sort (Integer) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	78	47	62	3567
20000	332	212	240	13798
40000	1363	784	951	55478
60000	2592	1792	2145	126329
80000	5295	3343	3808	224919
100000	6308	5286	5954	351484
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0081				

Table 17: ANOVA results for Bubble Sort (Integer 4 clients)

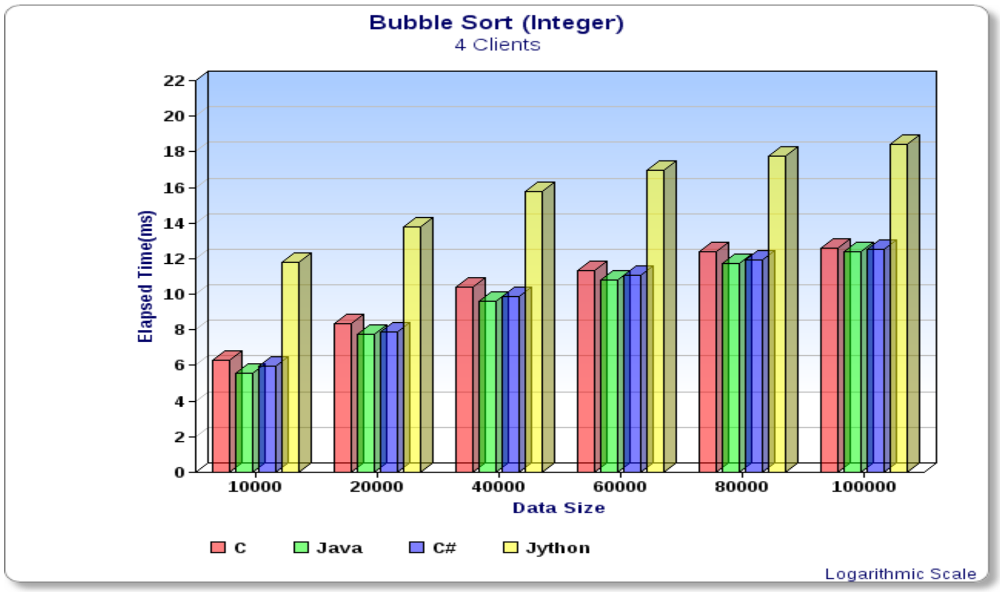

Figure 27: Bubble Sort (Integer) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	39	27	31	1738
20000	166	101	120	6927
40000	681	407	475	27917
60000	1296	941	1073	62331
80000	2221	1713	1904	11088
100000	3538	2638	2987	175994
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0078				

Table 18: ANOVA results for Bubble Sort (Integer 8 clients)

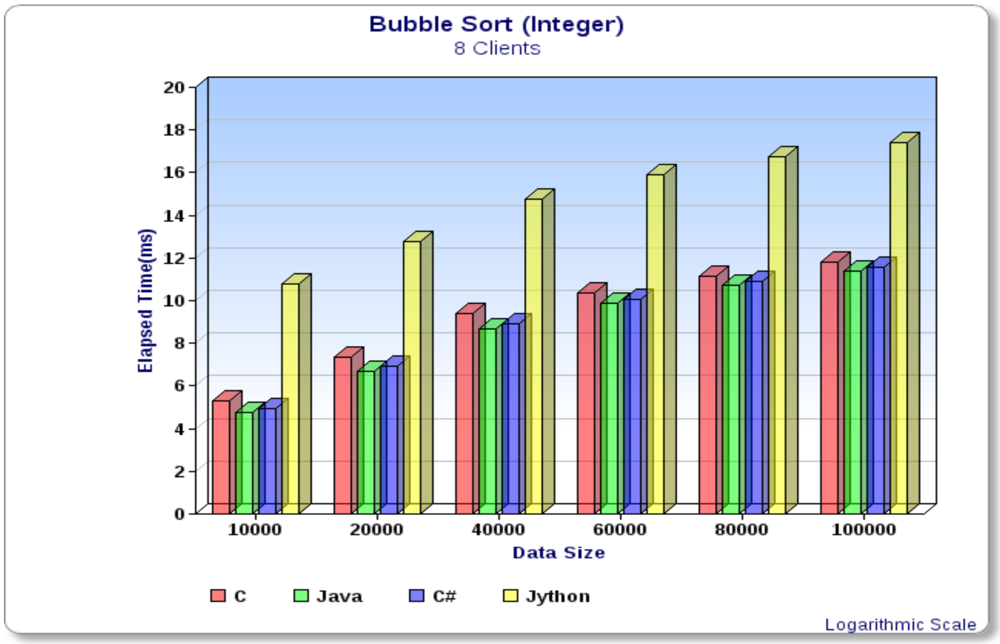

Figure 28: Bubble Sort (Integer) Using Socket Communication for 8 Clients

6.2 Bubble Sort (Float)

6.2.1 Evaluation of Bubble Sort (Float)

Table 19 through 22 depicts the data obtained from the algorithm and the result of ANOVA tests. Figures 29 through 32 shows the results for this algorithm were as expected. Java implementation was again the leader with the lowest elapsed time. Java and C# performed similarly for the reasons stated above. C implementation of the algorithm placed third. Jython had the expected result of being last, owing to its low execution speed and start up time.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	649	356	405	14903
20000	1455	1104	1235	58365
40000	5951	3750	4920	226813
60000	13557	8297	11045	504345
80000	24144	14670	19570	928930
100000	37821	22771	30550	1391903
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0078				

Table 19: ANOVA results for Bubble Sort (Float 1 client)

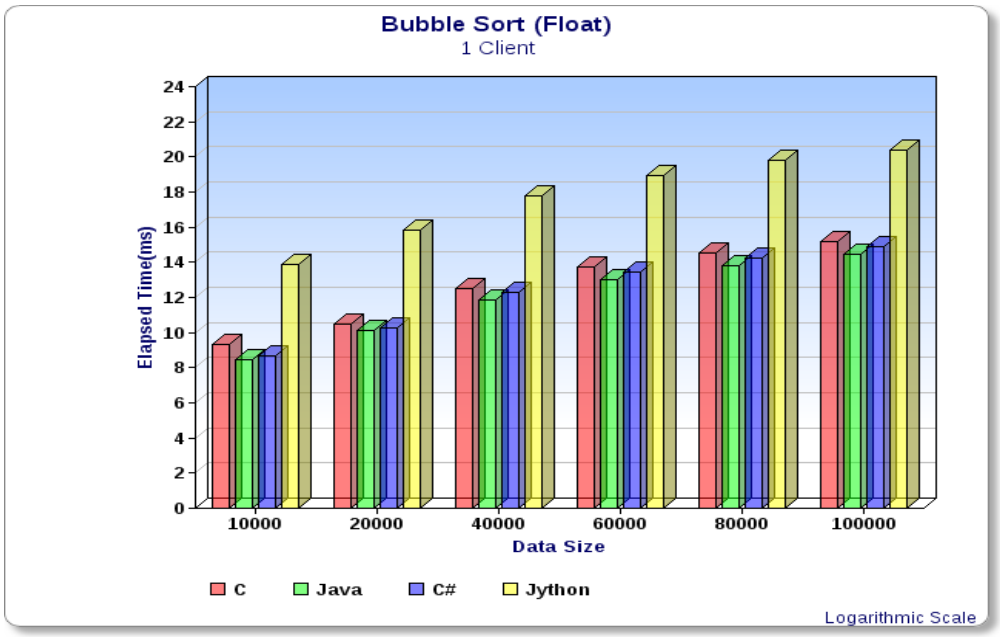

Figure 29: Bubble Sort (Float) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	171	178	202	8486
20000	717	552	617	29470
40000	2911	1875	2460	115502
60000	6626	4148	5523	264337
80000	11907	7335	9785	448589
100000	18742	11385	15275	700861
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0073				

Table 20: ANOVA results for Bubble Sort (Float 2 clients)

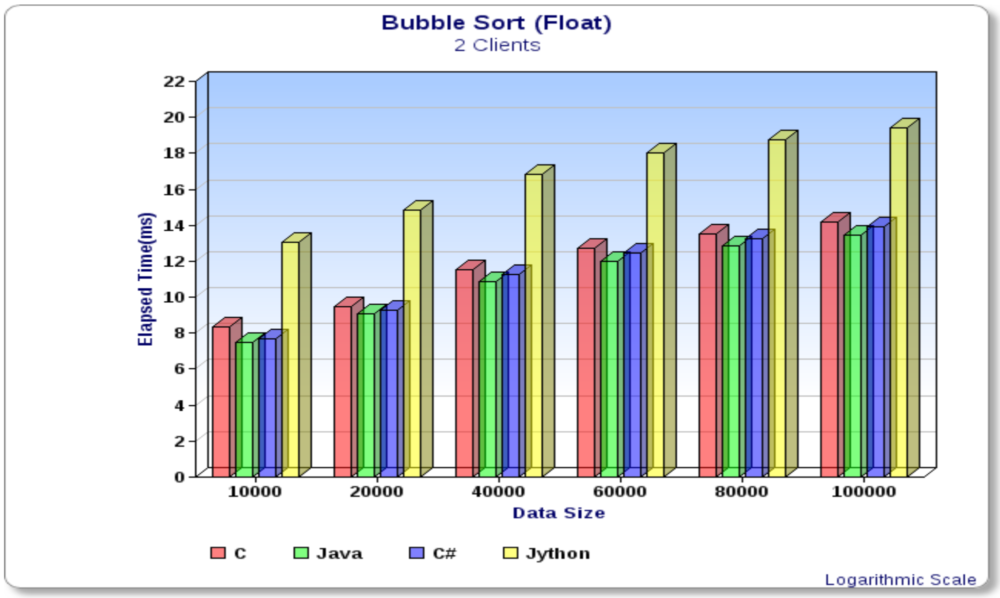

Figure 30: Bubble Sort (Float) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	85	88	101	4305
20000	351	276	309	16512
40000	1378	938	1230	60033
60000	3214	2074	2762	128439
80000	5408	3667	4893	226374
100000	10108	5692	7638	354626
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0073				

Table 21: ANOVA results for Bubble Sort (Float 4 clients)

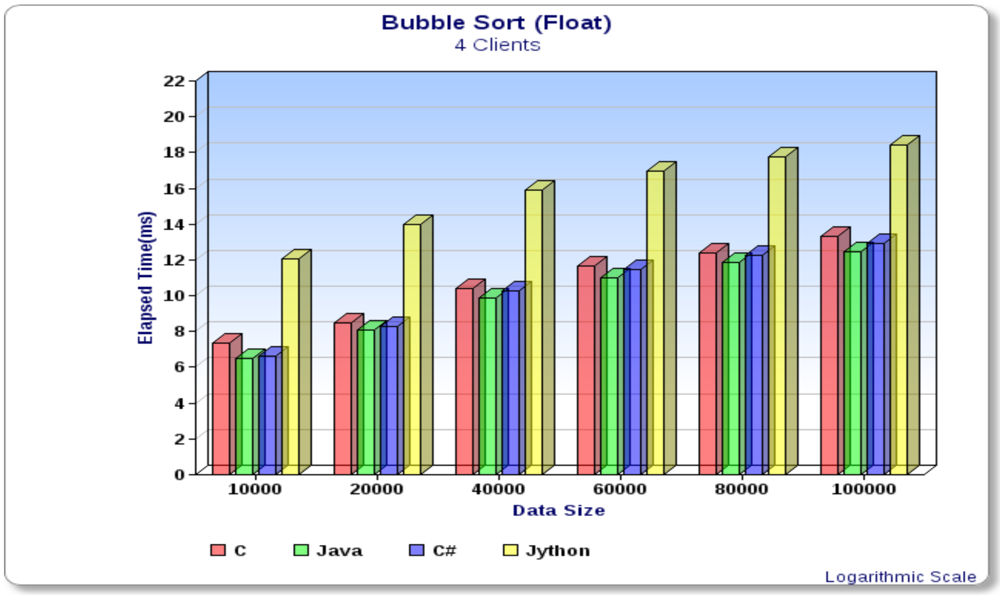

Figure 31: Bubble Sort (Float) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	54	44	51	1973
20000	179	138	155	7485
40000	743	468	615	33331
60000	1618	1037	1381	64683
80000	2914	1833	2442	113572
100000	6219	2846	3819	176629
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0067				

Table 22: ANOVA results for Bubble Sort (Float 8 clients)

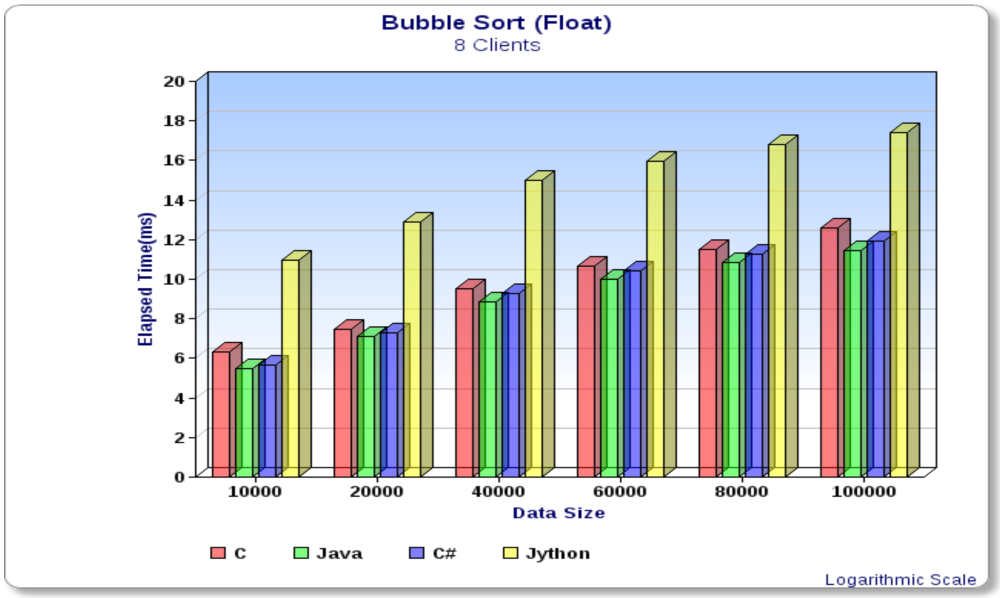

Figure 32: Bubble Sort (Float) Using Socket Communication for 8 Clients

6.3 Bubble Sort (String)

6.3.1 Evaluation of Bubble Sort (String)

Table 23 through 26 depicts the data obtained from the algorithm and the result of ANOVA tests. Figures 33 through 36 shows that, for this algorithm, the C implementation produced the best results, followed very closely by Java. String operations are slow in Java, because Java uses immutable, UTF-16 encoded string objects. This means that surplus memory is required, since a larger memory usage enhances the chances that parts of the program will be swapped out to disk. Moreover, swap file usage slows the speed. Finally, certain operations are more complex as compared to those with ASCII. C# implementation was slower than C and Java, but faster than Jython.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	614	1212	5310	13755
20000	2667	3613	20845	55409
40000	10896	14822	91913	221621
60000	24968	33277	216167	498060
80000	45510	57680	405606	924298
100000	72911	91337	681704	1349687
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.034				

Table 23: ANOVA results for Bubble Sort (String 1 client)

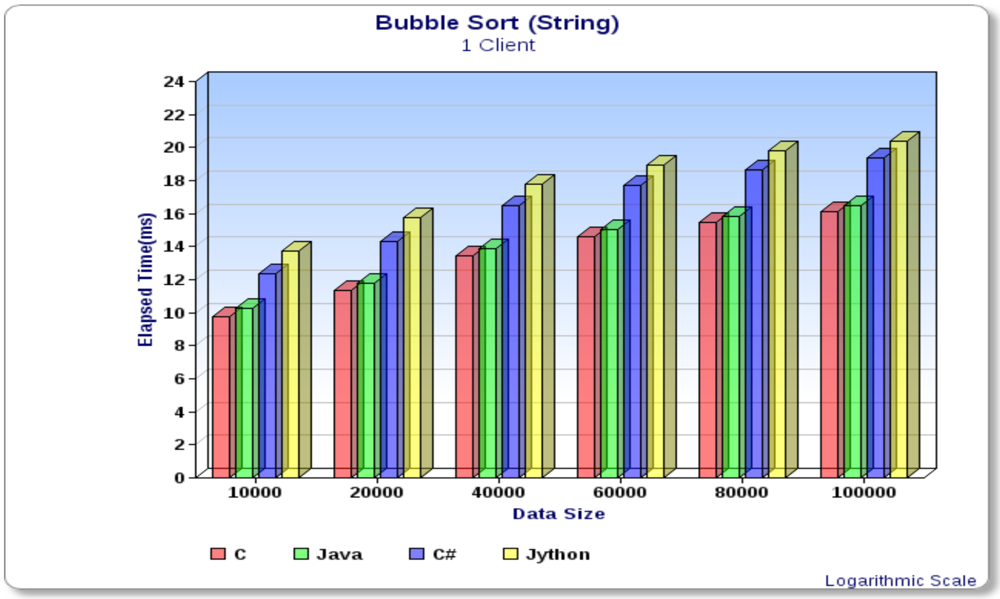

Figure 33: Bubble Sort (String) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	442	606	2650	6996
20000	1333	1806	10423	27586
40000	5448	7411	45957	112813
60000	12484	16638	108083	24882
80000	22750	28840	202803	442803
100000	36455	45668	340852	693216
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.11				

Table 24: ANOVA results for Bubble Sort (String 2 clients)

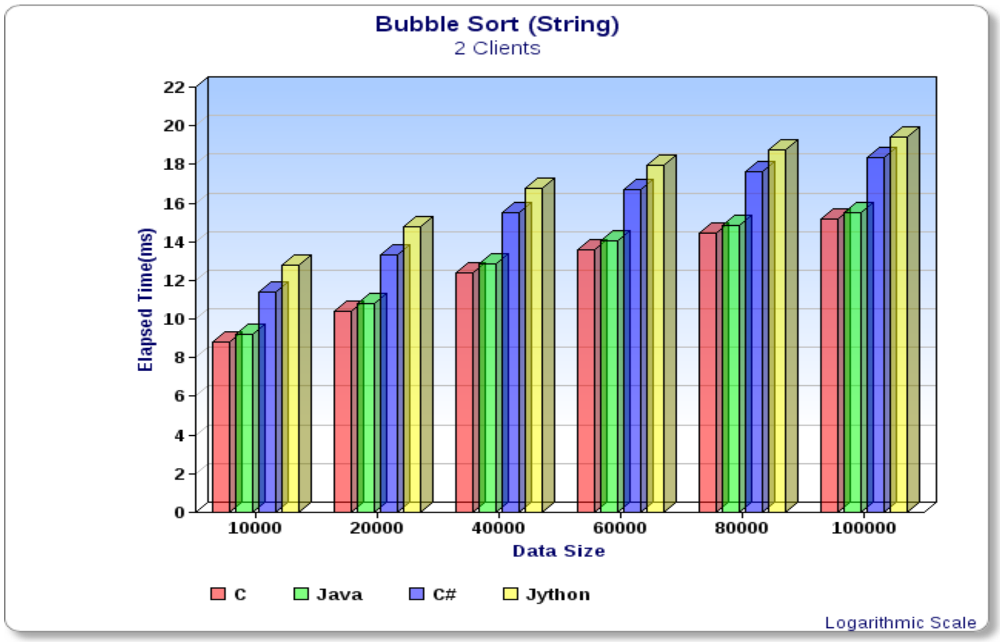

Figure 34: Bubble Sort (String) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	230	310	1325	3491
20000	668	922	5213	13762
40000	2854	3735	22979	55245
60000	6321	8413	54041	125246
80000	11432	15420	101401	223168
100000	18288	23856	170426	349441
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.037				

Table 25: ANOVA results for Bubble Sort (String 4 clients)

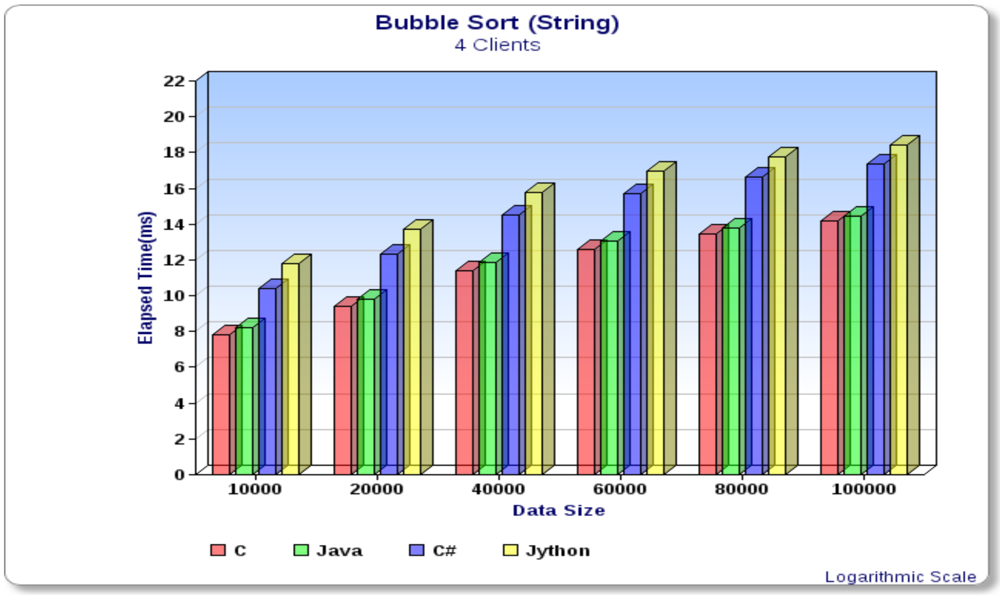

Figure 35: Bubble Sort (String) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	110	152	663	1740
20000	334	456	2607	7002
40000	1362	1852	11489	27653
60000	3121	4159	27020	62131
80000	5689	7210	50700	110846
100000	9114	11417	85213	173358
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.036				

Table 26: ANOVA results for Bubble Sort (String 8 clients)

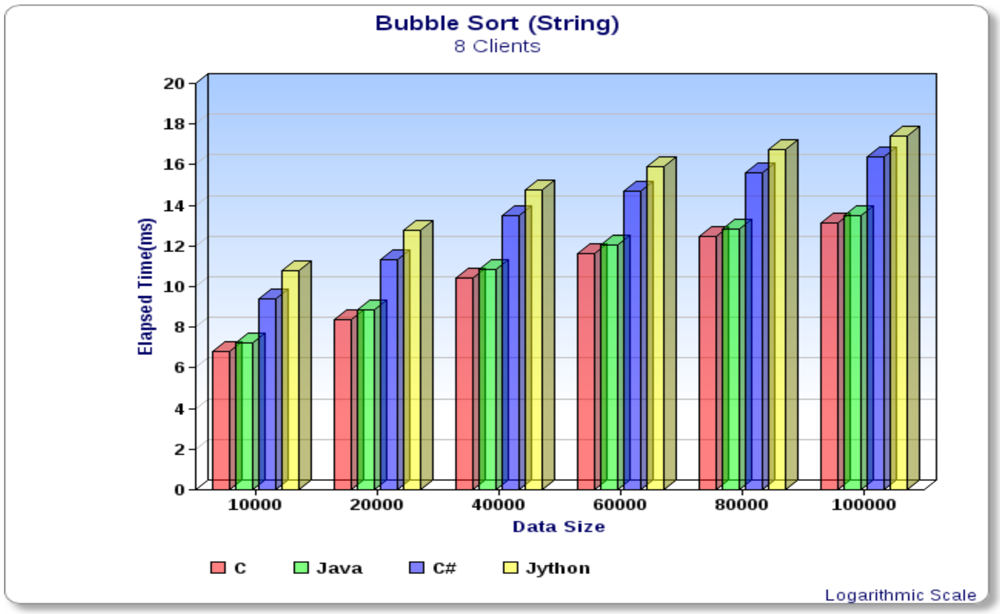

Figure 36: Bubble Sort (String) Using Socket Communication for 8 Clients

6.4 Quick Sort (Integer)

6.4.1 Evaluation of Quick Sort (Integer)

Table 27 through 30 depicts the data obtained from the algorithm and the result of ANOVA tests. Figures 37 to 40 show that C was the leader and proved to be the fastest solution. One of the surprising results is that C# was second, thus beating Java. Fourth was Jython.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	5	99	10	568
20000	7	137	20	1362
40000	16	165	42	2572
60000	24	190	53	3881
80000	32	209	69	4927
100000	41	212	99	5999
ANOVA: Results: The probability of this result, assuming the null hypothesis, is less than .0001				

Table 27: ANOVA results for Quick Sort (Integer 1 client)

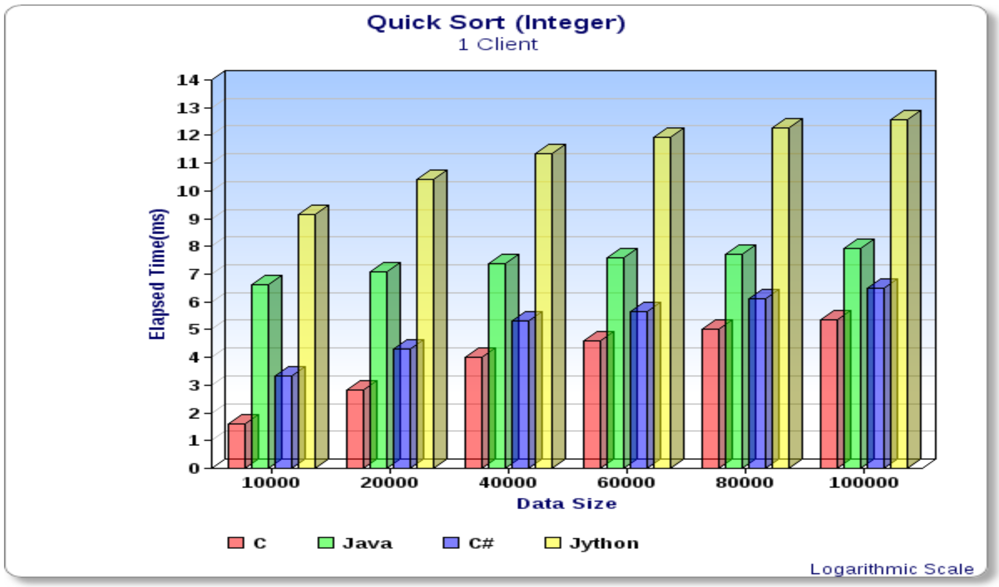

Figure 37: Quick Sort (Integer) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	4	46	7	284
20000	6	68	16	681
40000	14	85	25	1286
60000	22	95	34	1945
80000	30	105	41	2463
100000	38	110	62	2999
ANOVA: Results: The probability of this result, assuming the null hypothesis, is less than .0001				

Table 28: ANOVA results for Quick Sort (Integer 2 clients)

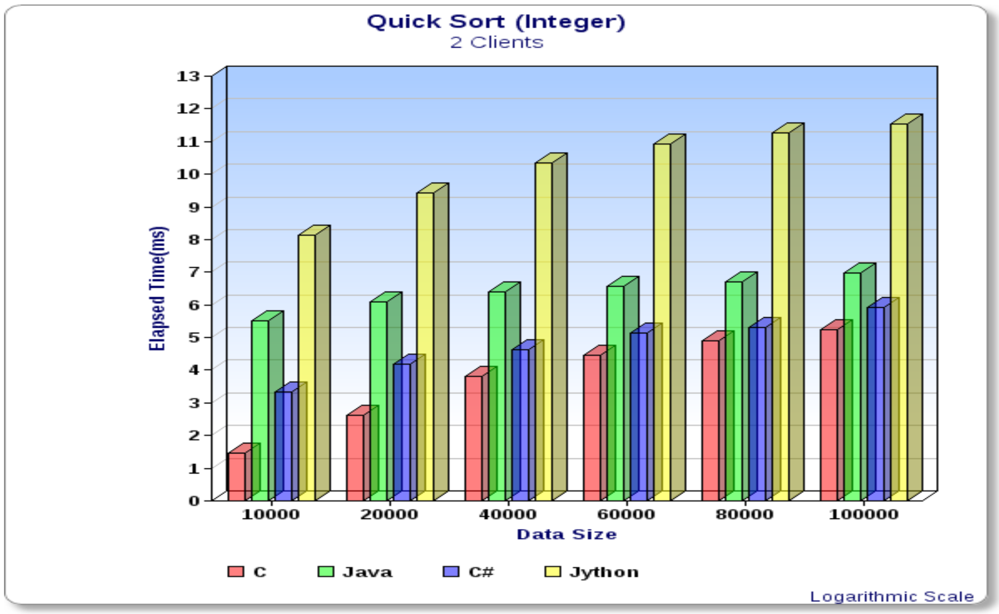

Figure 38: Quick Sort (Integer) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	3	23	5	142
20000	7	34	14	345
40000	12	43	20	645
60000	14	48	25	973
80000	24	58	30	1232
100000	33	68	40	1519
ANOVA: Results: The probability of this result, assuming the null hypothesis, is less than .0001				

Table 29: ANOVA results for Quick Sort (Integer 4 clients)

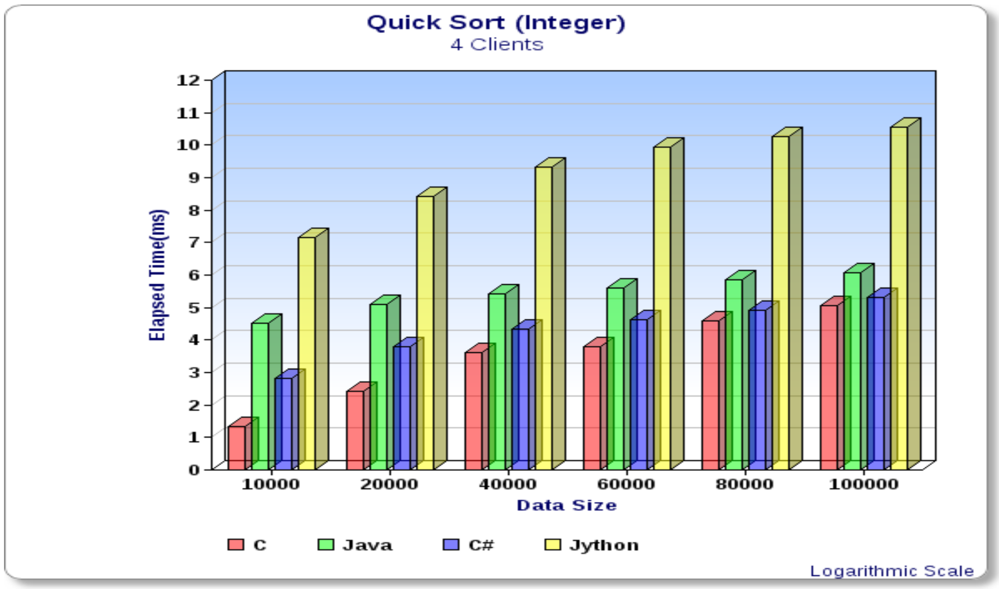

Figure 39: Quick Sort (Integer) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	2.5	17	4	75
20000	5	20	12	173
40000	6	24	15	322
60000	9	27	20	487
80000	14	34	25	616
100000	21	46	30	750
ANOVA: Results: The probability of this result, assuming the null hypothesis, is less than .0001				

Table 30: ANOVA results for Quick Sort (Integer 8 clients)

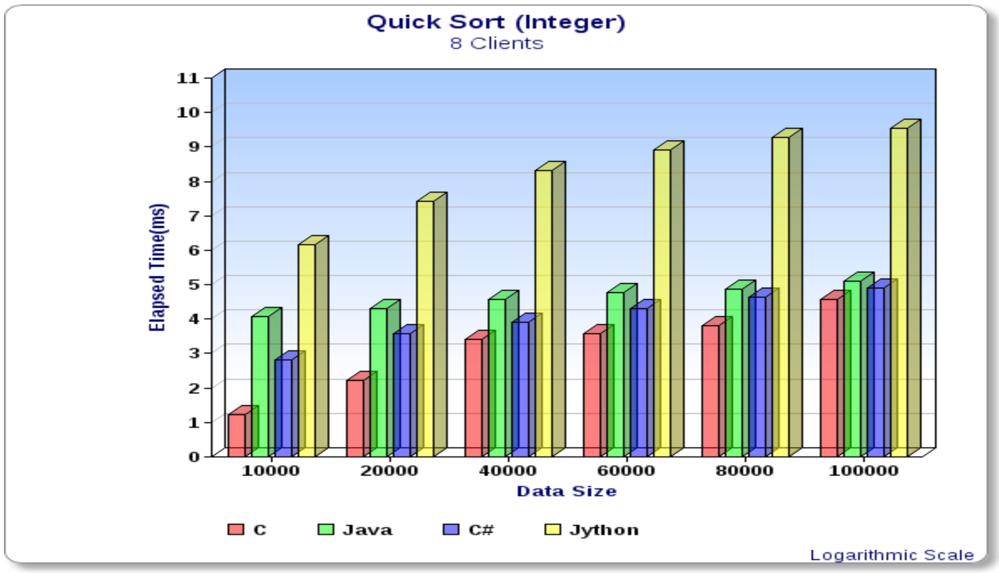

Figure 40: Quick Sort (Integer) Using Socket Communication for 8 Clients

6.5 Quick Sort (Float)

6.5.1 Evaluation of Quick Sort (Float)

Table 31 through 34 depicts the data obtained from the algorithm and the result of ANOVA tests. Figures 41 to 44 show that in this algorithm, C was the language that produced the fastest solution. Jython posted the slowest time since it is an interpreted programming language with low execution speed. In third place was Java, which runs on the virtual machine. This not only makes programming simpler but also more portable; however, the downside of this method is performance loss, in comparison to the compiled code in C, which is mainly noticeable at calculations comprising floating-point numbers. C# was second, but performed nearly as well as C.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	10	218	15	1218
20000	18	262	30	2363
40000	39	290	51	5268
60000	57	349	72	9559
80000	77	358	100	13287
100000	99	403	130	16481
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0003				

Table 31: ANOVA results for Quick Sort (Float 1 client)

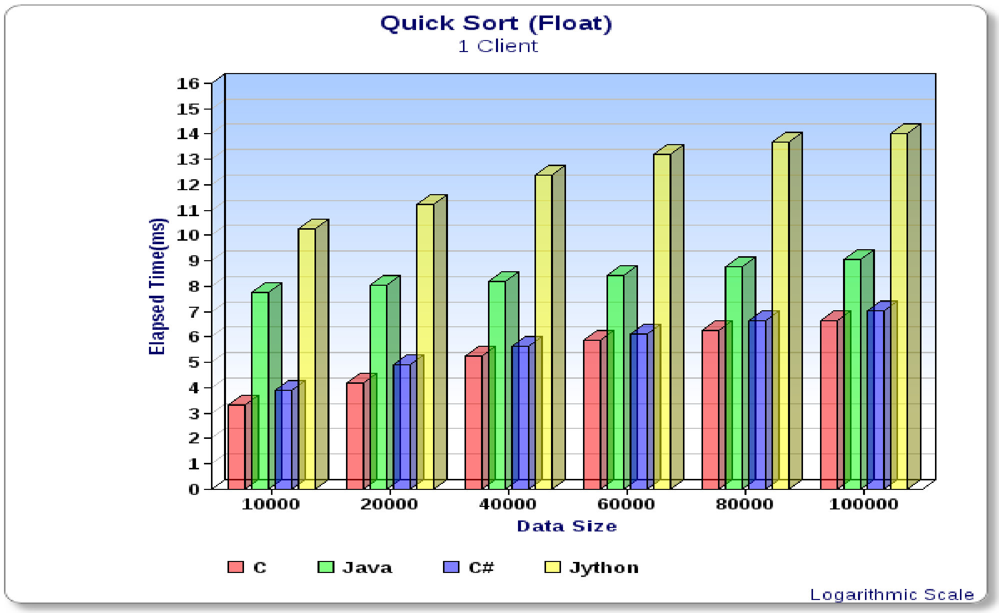

Figure 41: Quick Sort (Float) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	7	109	10	610
20000	12	131	28	1282
40000	20	145	35	2786
60000	30	170	40	4875
80000	41	189	50	6645
100000	52	202	65	8356
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0003				

Table 32: ANOVA results for Quick Sort (Float 2 clients)

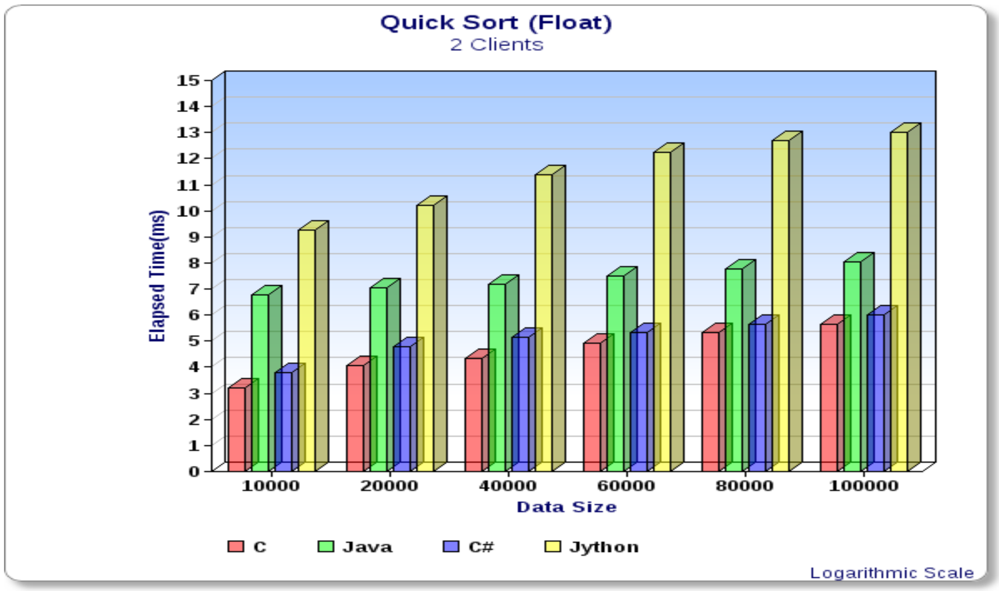

Figure 42: Quick Sort (Float) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	5	55	8	305
20000	9	66	18	591
40000	15	78	24	1317
60000	28	90	32	2389
80000	35	99	38	3322
100000	41	110	46	4120
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0003				

Table 33: ANOVA results for Quick Sort (Float 4 clients)

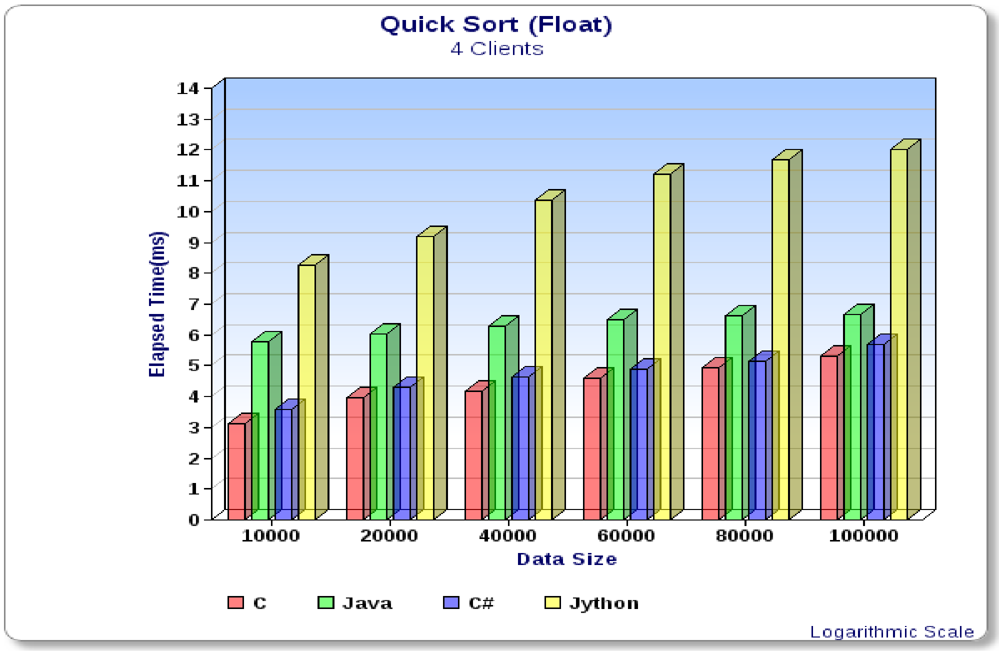

Figure 43: Quick Sort (Float) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	3	28	6	167
20000	6	33	10	319
40000	9	39	16	659
60000	16	45	23	1234
80000	24	51	29	1664
100000	35	57	38	2120
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0003				

Table 34: ANOVA results for Quick Sort (Float 8 clients)

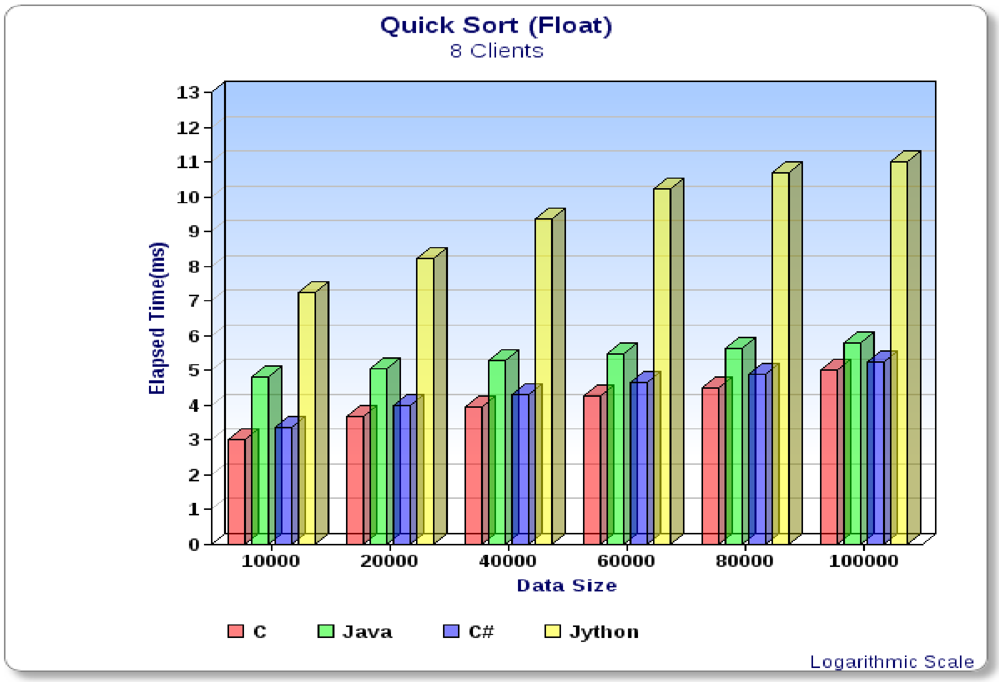

Figure 44: Quick Sort (Float) Using Socket Communication for 8 Clients

6.6 Quick Sort (String)

6.6.1 Evaluation of Quick Sort (String)

Table 35 through 38 depicts the data obtained from the algorithm and the result of ANOVA tests. Figures 45 through 48 shows that C# was the leader for character being the fastest. String types in both Java and C# exhibit a similar behaviour with slight differences in their execution speed. Second was Java, which was expected to be slower for strings since string operations are immutable in Java. Therefore, it requires extra memory and memory access. Third was C. Therefore, while the allocations/freeing are a slow process, if the data size of the input files were smaller, C would have performed much better than Java. Fourth was, again, Jython.

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	9	11	7	741
20000	24	19	10	173
40000	64	25	22	313
60000	111	32	24	457
80000	181	44	26	657
100000	269	85	28	804
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0009				

Table 35: ANOVA results for Quick Sort (String 1 client)

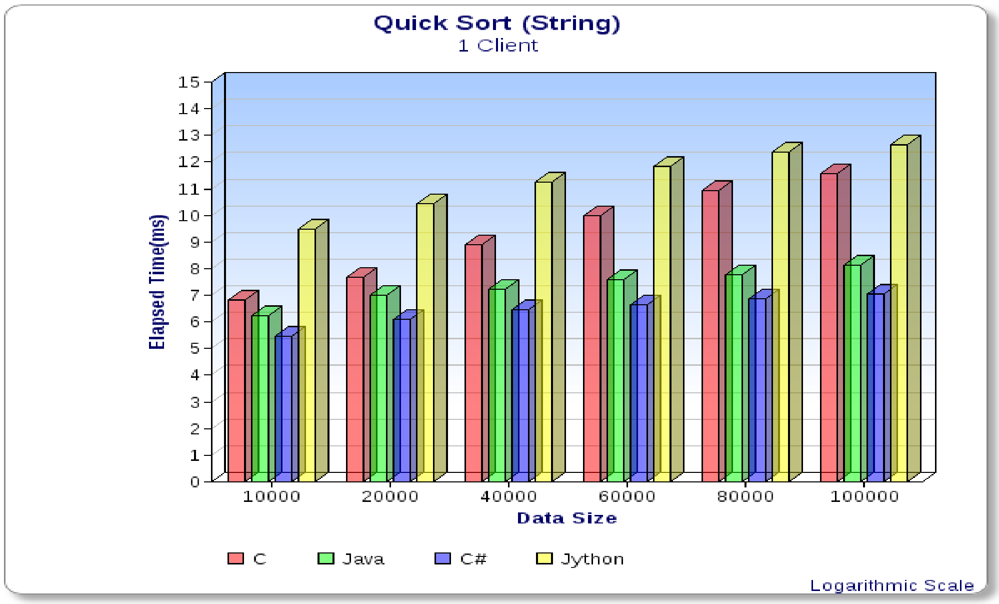

Figure 45: Quick Sort (String) Using Socket Communication for 1 Client

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	13	44	23	371
20000	67	69	30	689
40000	131	88	75	1209
60000	553	109	93	1819
80000	762	148	102	2789
100000	1513	222	150	3213
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0009				

Table 36: ANOVA results for Quick Sort (String 2 clients)

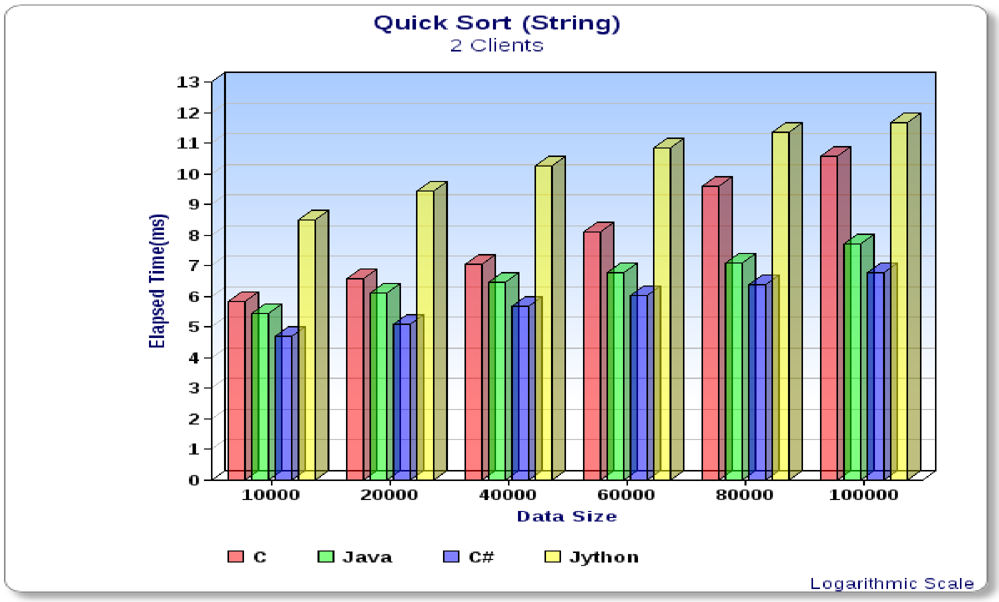

Figure 46: Quick Sort (String) Using Socket Communication for 2 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	12	24	10	130
20000	26	36	15	346
40000	75	45	36	605
60000	291	55	52	909
80000	392	78	56	1313
100000	583	160	62	1607
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0009				

Table 37: ANOVA results for Quick Sort (String 4 clients)

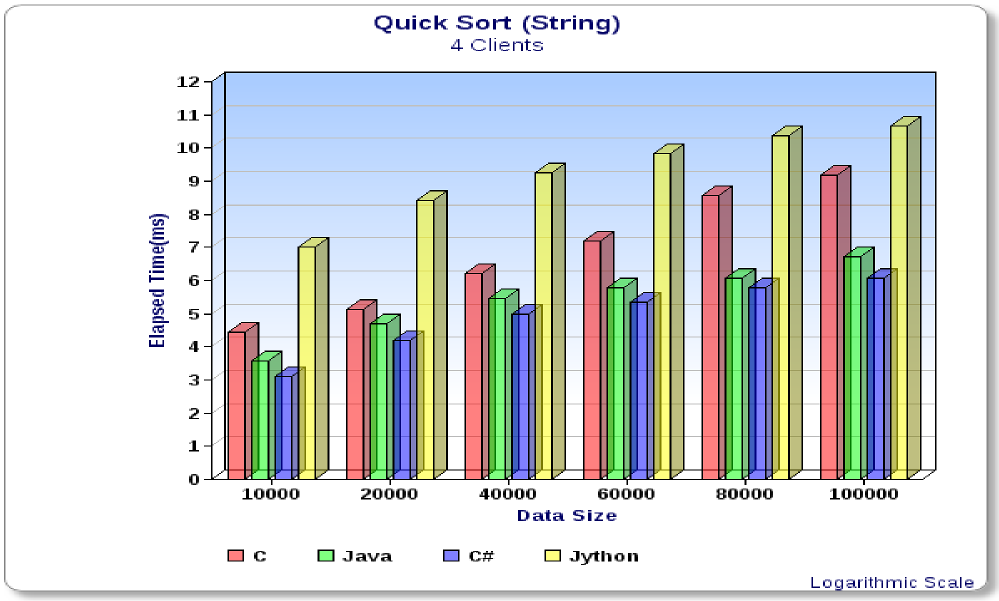

Figure 47: Quick Sort (String) Using Socket Communication for 4 Clients

Data Size	C(ms)	Java(ms)	C#(ms)	Jython(ms)
10000	9	11	7	71
20000	24	19	10	173
40000	64	25	22	313
60000	111	32	24	457
80000	181	44	26	657
100000	269	85	28	804
ANOVA: Results: The probability of this result, assuming the null hypothesis, is 0.0006				

Table 38: ANOVA results for Quick Sort (String 8 clients)

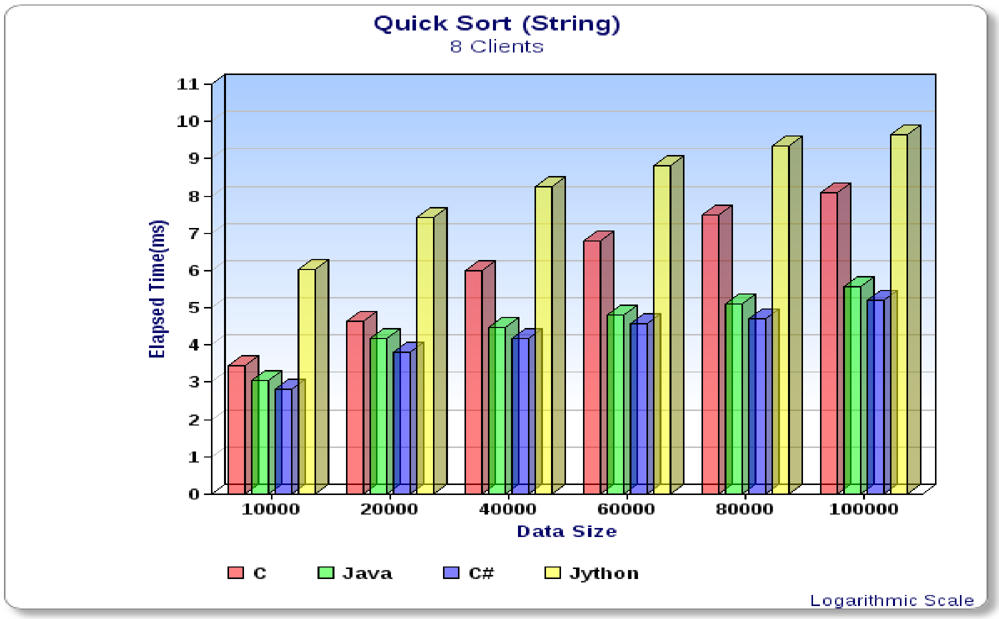

Figure 48: Quick Sort (String) Using Socket Communication for 8 Clients

Chapter 7

CONCLUSION

The main purpose of this research was to provide a comparison of the performance of different programming languages, when used to implement several classical algorithms and measurement criteria on a common platform. In this study, all of the results are based on measurement values taken from the three categories: CPU Utilization, Memory Usage and Run time in a distributed system using TCP sockets.

Some languages performed as anticipated and others performed with some unexpected outcomes. The research provided a well-defined perspective of how each language performed based on the memory consumption, CPU utilization, and runtime for client-server communication using socket API of different languages (C, C#, Java and Jython), on algorithms (Bubble Sort, Quick Sort, Linear Search and Binary Search), for data types (Integer, Float, String). The overall best performers in case of memory usage and CPU usage were C and Jython, undoubtedly with C being the leader. In the case of computing CPU utilization in Bubble Sort, Jython was fourth, which was quite obvious since interpreted languages like Jython do not perform efficiently in a completely CPU bound algorithm.

In the case of computing runtime in a Distributed System using TCP Sockets for Bubble Sort, Java and C# performed almost equally. The third best performer among the four programming languages here was C. Fourth was Jython, as expected, which proved that a simple language structure in terms of coding could have complex scores in this area. For Bubble Sort using string data types, C was the best performer due to Java’s slow speed of string operations.

For Quick Sort, C was the fastest solution. Second and third were C# and Java, respectively. Java has more tools accessibility across all the platforms, although there are many tools accessible for .NET on Windows platforms, increasing acceptance of C#. Jython is last for a language recommendation for network communication, based solely on the measurements evaluated. In addition, in the case of Quick Sort using string data type, C# and Java were first and second respectively, with C being in the third position. Fourth was again Jython, which was quite expected.

To presume that Jython is not suitable for an application because of the fact that it is excessively slow is not correct. There are not many applications these days for which execution speed is such a huge concern. In the event of a bottleneck in the code, we can simply move that area of code to Java as necessary. As programmer/developer, we have to be concerned with the speed of development rather than the rate of execution, and in this scenario, Jython is quick.

After reviewing these results, it is obvious that programming languages on a common platform using similar coding styles does have an impact in the performance of different algorithms. The main goal of this study was to illustrate these differences, which is quite evident from the measurements obtained from the calculations.

Tables 39, 40 and 41 show the top 2 programming language implementations in Bubble Sort, Quick Sort, Linear Search, and Binary search in terms of CPU Utilization, Memory Usage and Run time in distributed system using TCP sockets (for 1,2,4 and 8 client) for Integer, float and String data types.

Comparison of Programming languages C, Java, C# AND Jython on Integer data			
	Criteria		
Algorithm	CPU Utilization	Memory Usage	Run Time using TCP Sockets
Bubble Sort	C and Java	C and Jython	Java and C#
Quick Sort	C and Jython	C and Jython	C and C#
Linear Search	C and Jython	C and Jython	N/A
Binary Search	C and Jython	C and Jython	N/A

Table 39: Top 2 programming language implementations for Integer data type

Comparison of Programming languages C, Java, C# AND Jython on Float data			
	Criteria		
Algorithm	CPU Utilization	Memory Usage	Run Time using TCP Sockets
Bubble Sort	C and Java	C and Jython	Java and C#
Quick Sort	C and Jython	C and Jython	C and C#
Linear Search	C and Jython	C and Jython	N/A
Binary Search	C and Jython	C and Jython	N/A

Table 40: Top 2 programming language implementations for Float data type

Comparison of Programming languages C, Java, C# AND Jython on String data			
	Criteria		
Algorithm	CPU Utilization	Memory Usage	Run Time using TCP Sockets
Bubble Sort	C and C#	C and Jython	C and Java
Quick Sort	C and Jython	C and Jython	C# and Java
Linear Search	C and Jython	C and Jython	N/A
Binary Search	C and Jython	C and Jython	N/A

Table 41: Top 2 programming language implementations for String data type

Chapter 8

FURTHER RESEARCH

Further writings and experimentation on this subject could include comparing programming languages across platforms using this study's approach. Perhaps a developer could compare C# results obtained using the same categories in a Microsoft environment to those obtained here in the LINUX, and see how the programming language behaves differently.

REFERENCES

Print Publications

[Bates04]
Bates, B. (2004), “C# as a First Language: A Comparison with C++”.Journal of Computing Sciences in Colleges, volume 19 issue 3, January 2004, page 89-95.

[Berlin03]
Berlin, K, J. Huan, M. Jacob, G. Kochhar, J. Prins, B. Pugh, P. Sadayappan, J.Spacco, C. Tseng, “Evaluating the Impact of Programming Language Features on the Performance of Parallel Applications on Cluster Architectures”, Proc. LCPC 2003.

[Feurer82]
Alan R. Feurer and Narain H. Gehani, “A Comparison of the Programming Languages C and Pascal”, Bell Laboratories, Murray Hdl, New Jersey 07974, 1982.

[Gillespie12]
Gillespie, Tarleton “The Relevance of Algorithms”, forthcoming, in Media Technologies, Tarleton Gillespie, ed. P. Boczkowski, & K. Foot. Cambridge, MA: MIT Press 2012.

[Harel85]
Elie C. Harel and Ephraim R. McLean “The Effects of Using a Nonprocedural Computer Language on Programmer Productivity, “Journal MIS Quarterly Volume 9 Issue 2, June 1985, Pages 109-120.

[Pajjuri00]
A Pajjuri ., “A Performance Analysis of Java and C“, Feb 15, 2000.

[Prechelt05]
Prechelt, L. (2005), “An Empirical Comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a Search/String-Processing Program”. Technical Report 2005.

[Sestoft10]
Peter Sestoft (sestoft@itu.dk), “Numeric performance in C, C# and Java”, IT University of Copenhagen Denmark Version 0.9.1 of 2010-02-19.

Electronic sources

[Bell09]
Rob Bell 2009, “A Beginner’s Guide to Big O Notation”http://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation, last accessed October 22,2014.

[Squared05]
M Squared Technologies (2005), RSM Downloads. Resource Standard Software Source Code Metrics For C, C++, C#, Java and Visual BASIC. http://www.msquaredtechnologies.com/m2rsm_demo.php (2005, July 2).

VITA

Poonam Goyal received the Bachelor of Engineering degree in Information Technology from Rajeev Gandhi Proudyogiki Vishwavidyalaya, India. She expects to receive a Master of Science in Computer and Information Science from the University of North Florida in December 2014. During 2012 – 2013, she served as the Vice President of Upsilon Pi Epsilon (UPE) chapter at University of North Florida, which is an ACM/IEEE-CS International Honor Society for the Computing and Information disciplines. Her research interests are in the area of programming languages.