# Report on
# A Comparative Analysis of Standard Data Structures in Java and C++

Submitted to
Professor Martha Wall
for
English 316
Brigham Young University
Provo, Utah
March 29th, 2013

by
Andrew Jensen
Joey Cozza

# Brigham Young University

March 29th, 2013

Martha Wall, Ph.D.
4048 JKB
Brigham Young University

Dear Professor Wall,

Our technical report, A Comparative Analysis of Standard Data Structures in Java and C++, details original research that was conducted in order to quantify certain differences in Java and C++. We consider the use of abstract data structures in terms of algorithmic complexity, run-time performance, and maintainability. We also provide recommendations for programmers that are considering which programming language would be most appropriate for their applications.

Our report is written specifically for beginning programmers who have already taken at least one course in programming. Several graphs and diagrams have been included to help explain the purpose and implementation of abstract data structures.

We learned several interesting facts about the two languages while conducting our research. C++ code performed better than Java code, as we predicted; however, the difference was not as dramatic as we expected. Java code fluctuated quite a bit in its runtime performance, but the data structures from the built-in Java library were more consistent than our own implementations.

Thank you for the opportunity to perform this research. The process of preparing this report has been beneficial to the two of us and our findings will be useful for programmers in the future.

Respectfully,

Andrew Jensen
Joey Cozza

## Table of Contents

## List of Figures

## List of Tables

## Abstract

The choice of programming languages can have large effects on the end result of a program. Unfortunately, there has not been much research to compare individual programming languages with another, so at times it is difficult to know which language would be more effective to use. We decided it would be beneficial to programmers of all expertise levels to have a side-by-side comparison of Java and C++, two prolific programming languages. In order to narrow our research, we focused our study on the differences in Data Structures between the two languages. We began by making our own programs to time how long each language took to run some repetitive tasks. We also made our own implementations of some common data structures including Sets, Sorted Sets, Linked Lists, and Array Lists. We were expecting C++ to run more quickly, but Java to be easier to write for the programmer. In addition, we also assumed that our handwritten structures would always run slower than the standardized data structures. In general, the results were as we expected, but there were a few outliers that we hadn't anticipated. The most interesting output we saw was when our own implemented data structure effectively took no time to run, while the standard one took a full second to perform. We analyzed that output, and came to the conclusion that C++ was able to automatically optimize certain code that was handwritten, but not code that was included from a static library. This means that handwritten C++ data structures, on average, performed better than all of the other configurations we tested. We suggest that novice programmers create their programs in Java, using the Standard Java Library data structures for the sake of simplicity. We suggest that advanced programmers focusing on speed implement their own data structures in C++ to tailor their structures to their programs' needs and be as efficient as possible. Our research is available online and can be extended easily by other programmers in the future.

# A Comparative Analysis of Standard Data Structures in Java and C++

## I. Introduction

Fundamental differences between Java code and C++ code cause programmers to question which language they should use to create applications. These differences affect the speed and performance of the applications as well as the ease of programming them. This report outlines research performed to demonstrate the similarities and differences between code written in the two languages. The differences between these two programming language are complex and the issue cannot be completely addressed in a single paper. However, we have chosen to focus our research on a single, important facet of programming: data structures.

The use of **data structures*** is critical in modern programming. Data structures are "methods of organizing large amounts of data" (Weiss, 1997) in computer programs. Data must be organized into structures so that computers can manipulate it. These structures can be unordered, sequential, or sorted. These structures are simply abstract versions of structures that we interact with as part of daily life. However, computer programs are limited in the way they store information because they must store data as 1s and 0s. A primer on various types of data structures is provided in the following section.

Implementing data structures is a somewhat difficult and tedious process because they are such abstract concepts. Fortunately for programmers, programming languages have libraries of pre-written code that can be reused in many programs. Java comes with the **Java System Library** and C++ comes with the **Standard Template Library (STL)**. However, programmers must occasionally depart from these pre-written structures and make their own to fulfill a specific

*** For more information on bolded terms, see the glossary.**

requirement by an application. For this reason, it is essential for programmers to have a working knowledge of the different types of data structures and how to implement them. Because they are complex and abstract concepts, it is relatively easy to make mistakes while creating them. Therefore, we have considered **maintainability** as one of our standards of judgment in our comparison. Code is said to be maintainable when it is easy to write and easy to change and extend when necessary.

As previously stated, in order to save time and prevent unnecessary code duplication, a standardized and easily accessible library of functions and "tools" have been written for programmers to use at their pleasure. In order to make the best possible product, programmers need to know the limitations and benefits of these tools. By comparing two very prolific programming languages in terms of speed and algorithmic complexity, we give programmers a better understanding of how the choice of programming languages will affect their end product.

## LIBRARIES OF CODE

In computer science, a library is a collection of pre-written code that is available for re-use. Library code is written in such a way that it can be used easily by programs that are completely unrelated. Most compiled languages have a standard library which implement the majority of system services and basic functionalities. Many, if not all, experienced programmers will use specific libraries dealing with data structures in order to use good code that works correctly at no cost and no wasted time.

## STANDARDS OF JUDGMENT

In order to provide a clear comparison between data structures within Java and C++, we have decided to focus on two quantitative standards and to give our perspective on one qualitative aspect. The quantitative standards includes Big-O notation to measure complexity of algorithms and run-time performance to measure how long it takes a program to run. We also briefly explain the usability and maintainability of the two languages.

## II. PRIMER ON DATA STRUCTURES

### LINKED LIST

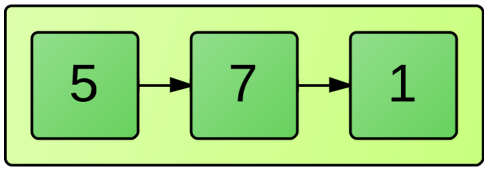


**Figure 1: A Linked List**
Nodes holding number values are shown in green.

A **Linked List** is a structure that is comprised of **Nodes** of computer memory (See Fig. 1). Each node holds a piece of information and points to the next node over. So like the name implies, it is a list that links several elements to each other inside of nodes (Langsam 186).

### ARRAY LIST

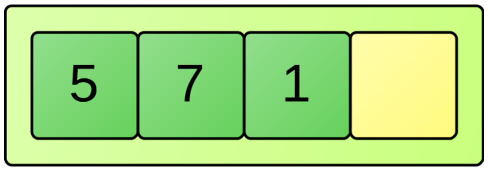


**Figure 2: An Array List**
There is an empty space at the end of the Array, represented by a yellow box.

An **Array List** (or **Vector** in C++ terminology) is the same type of structure as a Linked List, but it is implemented differently on the inside (See Fig. 2). Instead of having **Nodes** that connect to each other, an Array List contains a block of memory called an **Array**. The Array shown in Figure 2 has enough room for 4 elements. If five elements are added to the Array List, the first four will be added first. Then the program will pause to devote a new, larger block of memory to the List and copy the original elements in. Then it will add the fifth element in after them (Langsam 203).

## SET

A **Set** is an unordered collection of elements, like a Set in mathematics (See Fig. 3). There are multiple ways to implement an unsorted Set in a computer program, but they all involve a concept called a **Hash Table**. A Hash Table is an Array that holds references to elements of the Set. As elements are added to the Set, the proper position in the Hash Table is calculated by a **Hash Code**. This is done to decrease the time required to add and find elements in the Set (Langsam 488).

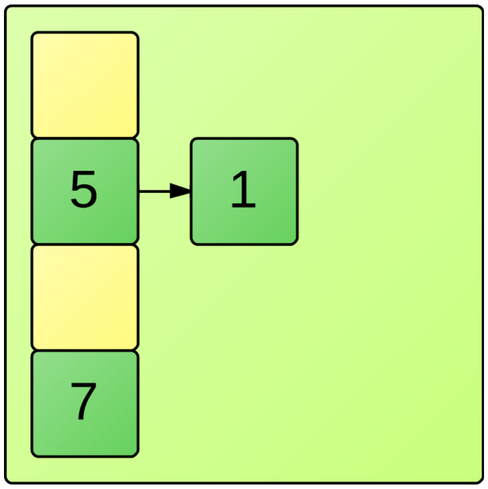


**Figure 3: A Set**
The Hash Table on the left contains Nodes with number values. Some spaces in the Hash Table are empty.

## SORTED SET

A **Sorted Set** has all the qualities of a Set (no duplicate objects, not indexed, etc.) but with an additional stipulation that the Set has a particular ordering to it (See Fig. 4). In general, this ordering provides more advanced functionality than a normal set, although it has its drawbacks as well (Langsam 249).

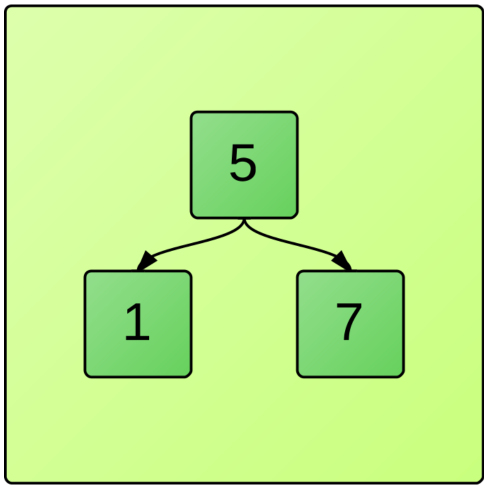


**Figure 4: A Sorted Set**
Each Node links to two “child” Nodes, forming a tree structure.

# III. RESEARCH METHODS

Due to the fact that little previous research has been done that directly compares Java and C++ with the standards of judgment we are using, we had to write our own tests to get the data we needed. One facet that we wanted to compare was how “handwritten” code stacked up against the pre-made libraries. We made our own data structures from scratch and implemented our own algorithms. This made it possible to see how well written the libraries of code are, as well as confirm suspicions we already had concerning bugs in hand-written code, and resulting performance issues.

## STRUCTURE IMPLEMENTATION

While in our own algorithms, we didn’t actually code up the most powerful or efficient methods, it is important to note that any experienced programmer knows that optimizing code before it is absolutely necessary is a poor practice. We followed basic implementations regarding structures that use nodes and structures that use arrays.

## CLOCKING RUN-TIME

Since computers can perform tasks so quickly, it can be difficult to tell the difference in speed between two algorithms performing the same tasks. In our research, we found that the algorithms to manipulate our data structures needed to be run thousands of times each to produce a difference of milliseconds in time. Similarly, it is easier to tell the winner of a marathon than a 100 meter dash because the times will vary more. For this reason, we created programs to test data structures with large data sets and ran them thousands of times.

We measured the runtime performance of data structures in Java and C++ by creating a **profiler** – a computer program that works like a stopwatch. We profiled the run-time performance of each data structure by *adding* 600,000 random numbers into it, then *searching* for 100 elements inside. Our profiling procedure was similar to that of Bull, Smith, Pottage, and Freeman, who compared various algorithms of importance to scientists and engineers between the C++, Java, and Fortran programming languages (2001). Our research serves as a useful update to their work as computers have increased in power dramatically from the time their research was published. We considered four experimental groups in every experiment:

1. The data structure as implemented in the Java System Library
2. The data structure as implemented in the C++ Standard Template Library
3. Our own Java implementation
4. Our own C++ implementation

We chose to compare all four groups in order to illustrate the efficiency of the data structures that exist in the Java and C++ libraries, as well as to show differences in runtime performance of the two languages.

## COMPUTER SPECIFICATIONS

All tests were performed on a MacBook Pro laptop with a 2.6GHz Intel i7 processor and 8GB RAM. All Java code was compiled using javac version 1.6.0_43. All C++ code was compiled with the GNU G++ compiler, version 4.2.1, set to optimization level 1.

## IV. ALGORITHMIC COMPLEXITY

One of the most common ways to analyze computer algorithms is with **Big-O notation**. It can be misleading to consider a data structure's performance only in terms of run-time performance because one computer may process the same structure faster than another. Rather, we are interested in the corresponding change in the amount of time required to process the structure induced by a change in the number of elements in the structure "n" (Langsam, Augenstein, Tenenbaum, 2003). Big-O analysis tells us how complex the algorithm is, and how many steps the algorithm will take to finish. Results of a specific study concerning Big-O is found in Table 1.

| Input Size | $O(N^3)$ | $O(N^2)$ | O(N log N) | O(N) | O(1) |
|---|---|---|---|---|---|
| **N=100** | 1,000,000 | 1,000 | 200 | 100 | 1 |
| **N=1,000** | 1,000,000,000 | 1,000,000 | 3,000 | 1,000 | 1 |
| **N=10,000** | 1,000,000,000,000 | 100,000,000 | 40,000 | 10,000 | 1 |
| **N=100,000** | 1,000,000,000,000,000 | 10,000,000,000 | 500,000 | 100,000 | 1 |

**Table 1: Number of Calculations performed with several algorithmic complexities (Weiss, 1997).**

To put Table 1 into other words, if a program was run with a slow O(N3) algorithm with an input size of 100,000, it would 11 days to finish, but a little better algorithm of O(N2) would only take 1/10th of a second!

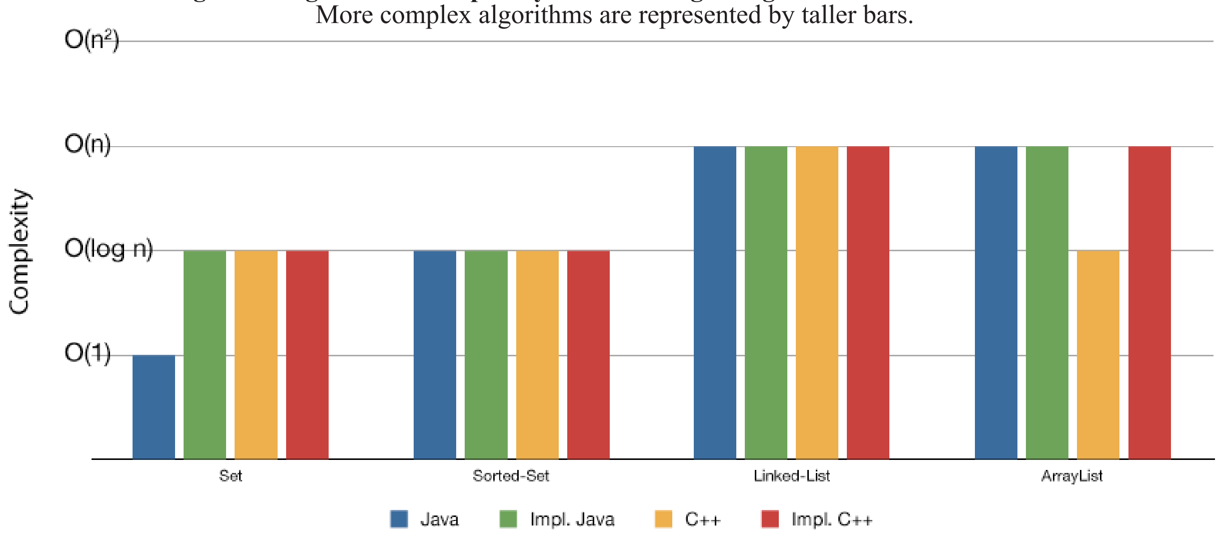
### JAVA VS C++

The complexity of the data structure algorithms was mostly equal between Java and C++ code (Sedgewick, R. 470). This was expected because standard data structures have implementations that are widely understood and taught in textbooks. At Brigham Young University, for example, the Data Structures course focuses on these standard implementations. However, as we will show in the following section, there were noticeable differences in runtime performance between Java and C++ code. This is not because the algorithms in the two languages are drastically different, but because of the **Java Virtual Machine (JVM)** and **garbage collection**, both characteristics of Java code that cause it to run slower than C++ code.

## IMPLEMENTED ALGORITHMS

In order to be realistic in the code that we wrote, we decided to follow good coding practice and write simple working versions of the code we needed. It is always better to write a "slow" working program than to spend inordinate amounts of time writing an amazing program that never quite works. In the programming community it is put this way, "premature optimization is the root of all evil". It turns out that the 2 functions we implemented by hand, "add" and "find", are quite simple, and our algorithms are on par with the standardized algorithms. If we had tried sorting our data structures, our algorithms would have been much slower than the standard ones. A complete comparison of algorithmic complexity for the data structures in our experiment is shown in Figure 5. In general, searching for elements in Sets and Sorted Sets is faster than searching through Lists.  This is also apparent in our run-time performance results (See Fig. 7). Although data structure implementation is well-documented and fairly standard, there were still a few differences, as seen in Figure 5. These few changes were a surprise to us, but they reinforced our suspicions of the efficiency gap between Java and C++. They are also reflected in our measurements of run-time performance.

**Figure 5: Algorithmic complexity for searching though various data structures**
More complex algorithms are represented by taller bars.



## V. RUN-TIME PERFORMANCE

**Run-time** is the period when a program is actually being executed. In terms of computer science, a shorter run-time is better than a longer run-time. Nobody wants to wait for a computer program to finish; they want it to be done as quickly as possible. This facet of programming is the most obvious to the average computer user, and as such is one that we will weigh more heavily on than the other standards. The way that we decided to measure our run-time accurately was to use a Profiler program.

### JVM VS. NATIVE C++ CODE

There are two major differences between code compiled from Java and code compiled from C++. These differences make Java easier to program and more compatible across multiple operating systems, but make it slower to execute than C++ code. One of the main aims of our research was to demonstrate this difference in run-time performance.

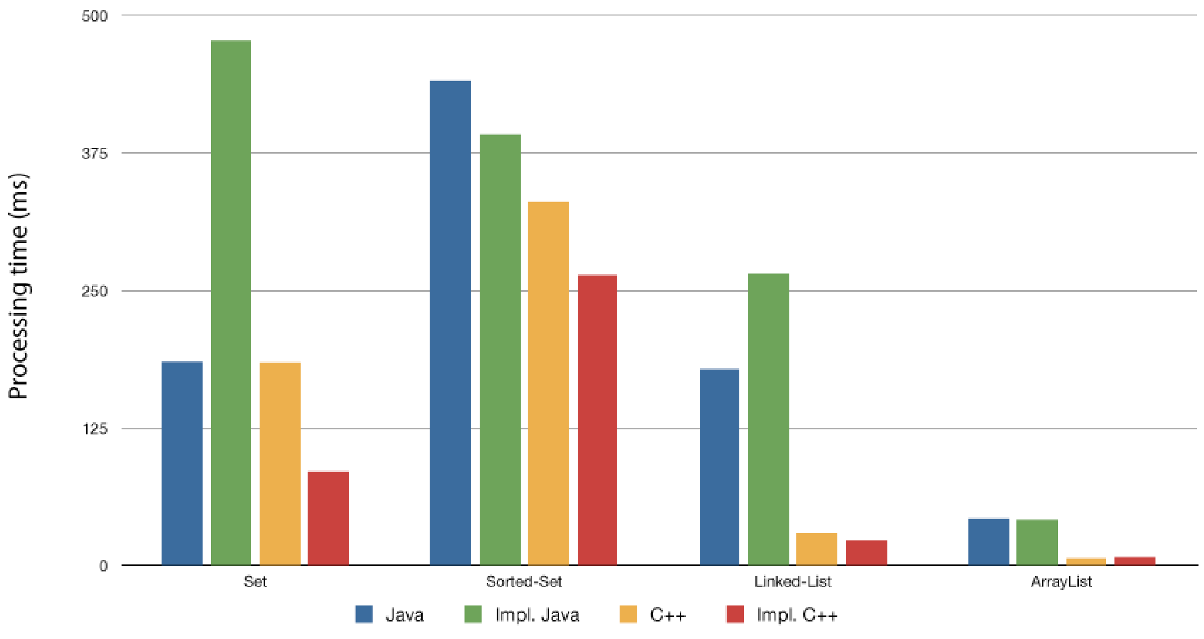
The first major difference between Java and C++ is the JVM. C++ code is compiled directly into machine code (1s and 0s) that can only be run by a specific kind of processor while Java code is compiled into byte code, a machine-readable language that is interpreted by the JVM when the program is executed. The JVM handles the conversion from byte code to machine code, meaning the programmer does not need to create multiple versions of the program for different operating systems, but this code will run slower than C++ code. For a long time, many programmers had a negative attitude towards Java because of the JVM's slower running time. In a study performed in 2001, it was shown that the JVM was being improved, and they conclude that they expected the differences in run-time between Java and other "faster" programming languages would continue to decrease (Bull, J. M., Smith, L. A., Pottage, L., & Freeman, R.).

Garbage Collection is another characteristic of Java that separates it from C++. Memory allocation in C++ is unregulated, meaning it is the programmer's responsibility to free up the memory that a program has used. This makes C++ a difficult language for beginners to use. It is easy to forget to free up memory and the resulting decrease in performance is called a "memory leak." If the memory leak is large enough, the user will start to notice a difference in the computer's responsiveness. Java manages memory allocation so the user does not need to. The JVM keeps a log of what data has been created by the program and frees up the parts that are no longer needed. This makes Java much easier for beginners to learn. However, Java's garbage collection is occasionally slow to delete unused data, and takes time from the regular program from running. This typically makes Java code slower to execute than C++ code.

## IMPLEMENTED RUN-TIME

Analyzing the data of our implemented structures showed surprising results. We were confident that our handwritten code would be outperformed by the standard libraries in Java and

**Figure 6: Run-time performance for adding elements**
Sets and Sorted Sets performed roughly equally well at adding.
Linked Lists and Array Lists where significantly slower in Java than C++.



C++, but that wasn’t always the case. Our own Java implementations were either outperformed or equal to the Java library, but we were amazed to see that our C++ code beat the C++ library in terms of run-time (See Fig. 6). After studying the results and analyzing the code that we had written, we came to the conclusion that our handwritten C++ code was being automatically optimized when it was compiled into its final form (See Fig. 7 and 8). The computer was not able to access and optimize the library code because it has already been compiled. We were not expecting that from our results, but we feel that it is pertinent to our study because automatic optimization is a useful tool that should almost always be used.

**Figure 7: Run-time performance for finding elements**
We were surprised to note that our implemented data structures outperformed the structures in the system libraries. A zoomed graph is shown in Fig. 8, showing find times for Sets and Sorted Sets.

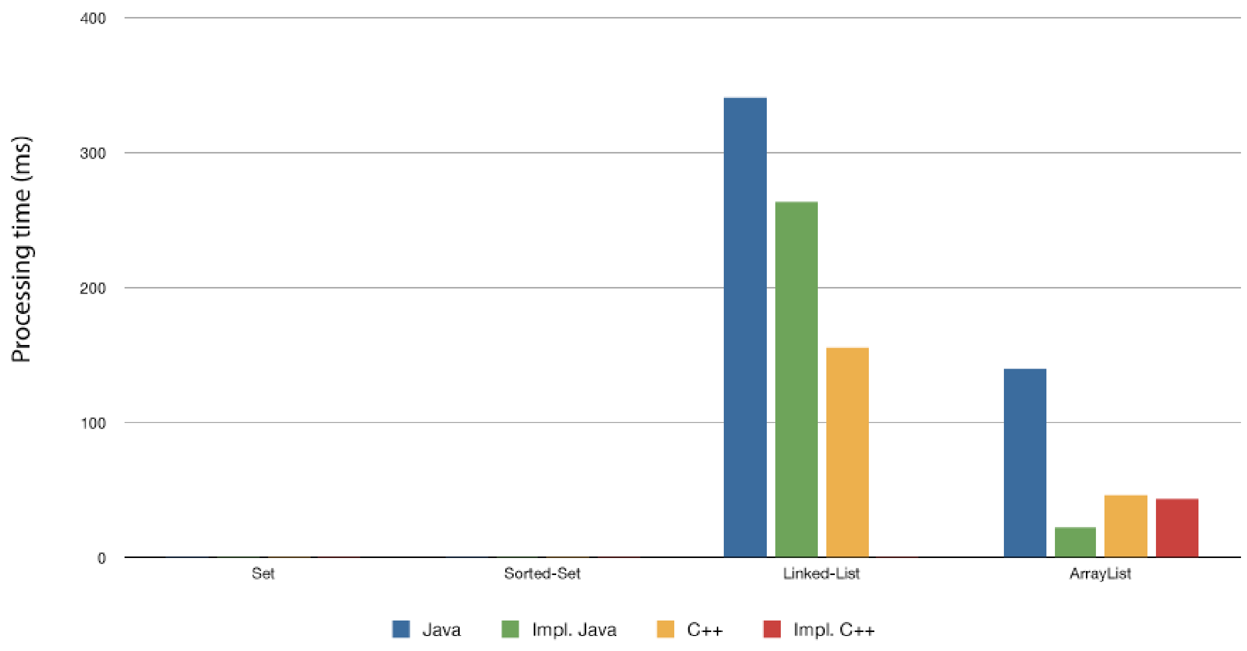
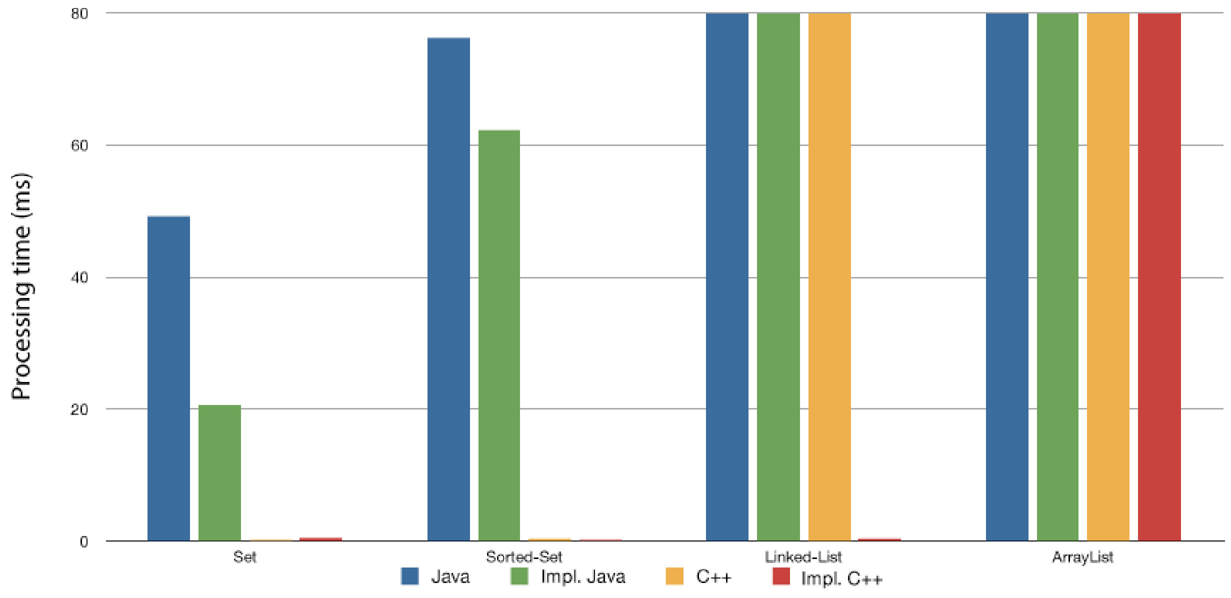


**Figure 8: Run-time performance for finding elements (zoomed)**
C++ Structures performed significantly faster than Java structures. It is worth noting that our implemented Set was less efficient that the Java System Library's Set in adding, but more efficient when finding elements. Our C++ implementation of a Linked List is an anomaly, likely caused by the optimizing compiler circumventing our code.



## VI. QUALITATIVE STANDARDS

### EASE OF USE

As we implemented each of the four data structures in Java and C++, we noted the more difficult steps required to write the program. The number of lines of code required for each class can be seen in Table 2.

| | Linked List | Array List | Set | Sorted Set |
|---|---|---|---|---|
| Java | 79 | 73 | 197 | 115 |
| C++ | 110 | 77 | 230 | 150 |

**Table 2: Lines of code required to implement data structures**
We recommend that novice programmers use the data structures implemented in the Java System Library and STL instead of implementing their own; these lines of code need not be written again.

If a programmer chooses to use data structures from the Java System Library or the Standard Template Library, none of this code would be necessary. Furthermore, the structures created in our experiments were only partially implemented: functionality for removing elements and retrieving elements at specific locations were not implemented and memory management was not considered. Adding these typically necessary features would require approximately 100-150 additional lines of code per structure. Simply put, using the data structures provided in standard libraries saves hundreds of lines of difficult and abstract code. If ease of use is the most important standard of judgment in a project, the programmer should choose to use the data structures already provided.

### MAINTAINABILITY

We discovered a problem as we prepared to run the profiler on our data structures: we had to determine what kind of elements would be put in each of the structures before we tested them. Programming languages use **type checking** to ensure that elements of one type are not confused for another. This is analogous to a shopping list and a to-do list. Shopping lists contain groceries to buy like “apples” and “oranges” while to-do lists contain actions to perform like “wash the car" or "paint the fence." If a shopping list included "wash the car," it would not be a valid shopping list.

Computer programs enforce this type of type checking automatically inside of data structures. If the programmer tries to put objects of the wrong type into a data structure of another type, the computer will report an error. It is possible to create dynamic data structures that can accept arbitrary types of objects inside; this method is called **Generic Programming** in Java terminology or **Templating** in C++ terminology (Kak, A. C. 547). One report from 2003 explained the difficulty with Java's generic programming is that one has to work within the constraints of a predefined collection hierarchy, and extend it in a useful manner. C++ on the other hand is extensible to any type of container or algorithm without those constraints (Saiedian, H., & Hill, S. 133).

By utilizing generic programming, or templating, maintainability is much easier in the long run. The problem is that initially setting it up for handwritten structures is a complicated and error prone process. We feel it is imperative to understand what the program will be used for in order to make a decision concerning this level of maintainability and complexity.

# VII. CONCLUSION

## SUMMARY OF RESULTS

Our research provides strong evidence that the JVM has increased significantly in speed over the last decade and that run-time performance is becoming more uniform in Java and C++ programs. The research also illustrates the advantages and disadvantages of creating data structures instead of using the structures from standard libraries.

We found the implementation process to be quite difficult. As we created the data structures, we had to refer to our sources for specific details on programming language syntax, especially syntax required for generics and on the methods for creating hash tables.

We saw dramatic differences in run-time performance in C++ data structures by using an optimizing compiler. In some cases, the compiler circumvented the code in our implemented data structures to increase speed. Our results add to a wealth of evidence that advances in compilers and runtime environments is making programming languages more and more similar in terms of run-time performance.

## RECOMMENDATIONS

For novice programmers, we recommend using data structures from the Java System Library. One of the main reasons why Java is simpler than C++ is because the entire issue of memory allocation is handled by the Java Virtual Machine. Furthermore, the standard data structures needed for most programs are included in the Java System Library and Standard Template Library and need not be implemented again. For novice programmers, using system code will be the most effective way to proceed on a project.

For experienced programmers whose primary focus is their program's run-time performance, we recommend implementing data structures in C++. During our profiling, we were shocked to see the difference in run-time performance as we compiled our code with an optimizing compiler. Programmers have greater control over their program when they create more of it themselves. If the requirements for a data structure are clear, then the programmer creating it should tailor it to his or her own needs to make it as efficient as possible.

## IMPLICATIONS FOR THE FUTURE

Our research is useful in showing the differences in run-time performance between four different experimental groups. There are several possible ways to extend this research in ways that would lead to useful statistical data. The source code to our experiments is publicly available online and can be downloaded at https://github.com/andrewjensen/profiling-structures-java-cpp . Our data structures and profiler should be easily extensible and will make it easier for programmers to run additional tests in the future.

One useful way to extend our research would involve adding varying numbers of elements into data structures. For each of our experiments, 600,000 elements were added to the structures. Tests could be written that add 100 elements, 1,000 elements, 10,000 elements, and so forth. This would help to show the effect of the Java Virtual Machine's delayed garbage collection on increasingly large sets of data.

It would also be interesting to note the improvements made to the JVM and to hardware in the future and see how that affects the performance of Java and C++ respectively.

## Bibliography

Bull, J. M., Smith, L. A., Pottage, L., & Freeman, R. (2001). *Benchmarking Java against C and Fortran for scientific applications*. Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande, 97-105. Retrieved from http://dl.acm.org.erl.lib.byu.edu/citation.cfm?id=376656.376823& coll=DL&dl=ACM&CFID=184458434&CFTOKEN=16720360

Kak, A. C. (2003). *Programming with objects: A comparative presentation of object-oriented programming with C++ and Java*. (1st ed.). Hoboken, NJ: John Wiley. Retrieved from http://search.lib.byu.edu/byu/id:byu5122679

Langsam, Y., Augenstein, M. J., & Tenenbaum, A. M. (2003). *Data structures using Java*. (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Saiedian, H., & Hill, S. (2003). *A comparative evaluation of generic programming in Java and C++ : a semantic theory for an embedded coordination language. Software—Practice & Experience*, 33(2), 121-142. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/spe.499/references

Sedgewick, R. (2002). *Algorithms in Java*. (3rd ed., Vol. 1-4). Indianapolis, IN: Addison-Wesley Professional.

Weiss, M. A. (1997). *Data structures and algorithm analysis in C*. (2nd ed.). Menlo Park, CA: Addison-Wesley Professional.

## Glossary

**Array** – A block of memory allocated by a computer program. Arrays are used by data structures to connect elements together.

**Array List** – An implementation of a sequential list where elements are stored in an Array of memory. Note that Array Lists and Linked Lists are used the same way by other programs, but they are implemented differently internally.

**Big-O notation** – A type of notation used to express the complexity of a computer algorithm. Algorithm complexity is expressed in terms of a problem size n. For example, our research involves adding 600,000 elements to each data structure, so n=600,000. Algorithm complexity is independent from run-time performance, so Big-O notation provides an objective way to compare the efficiency of various data structures without considering other variables like system memory or processor speeds.

**Data structure** – Weiss defines data structures as "methods of organizing large amounts of data" (1997). Data structures can be unordered, sequential, or sorted. Our research focuses on four basic types of data structures: Linked Lists, Array Lists, Sets, and Sorted Sets.

**Garbage Collection** – Java's built-in method of memory management. As data is removed in a program, the JVM will free the memory dedicated to holding the data, leaving room for other data to take its place. Garbage Collection makes Java programming easier than C++ programming, but makes it less efficient in terms of run-time performance.

**Generic Programming** – A method of programming data structures that allow a type of structure to be reused with different types of elements, like a template. Generic programming makes data structures more usable, but is somewhat difficult to implement and largely unused aside from data structures.

**Hash Code** – A numeric code generated by elements of a data structure that allow them to be placed inside of a Hash Table. The use of Hash Codes allows data retrieval to occur extremely fast in Sets compared to other data structures.

**Hash Table** – A table in a Set that contains Nodes holding elements. Each element's position in the Hash Table is determined by a Hash Code.

**Java System Library** – The library of data structures, algorithms, and other useful code in Java. Re-using code from the Java System Library greatly increases maintainability in programs, and the code in the library includes optimizations that programmers do not need to consider as they create applications.

**Java Virtual Machine (JVM)** – An interpreter that take compiled Java byte code and turns it into machine code (1s and 0s). The JVM is an added layer of processing that handles the inconsistencies of different operating systems. In other words, a programmer can compile a single version of a Java program and it will run normally on Windows and Mac OS, while a programmer must compile different versions of the same program to be compatible with the two. The disadvantage to this cross-compatibility is slower run-time performance: C++ runs faster than Java because no extra interpreting is necessary at runtime.

**Linked List** – An implementation of a sequential list where elements are stored inside Nodes that are linked to each other. Linked Lists are efficient for adding elements because they are not limited by finite Arrays of memory.

**Maintainability** – Code that is maintainable is easy to write and easy to change and extend when necessary. Maintainability is critical in programming because programmers are pressed to develop new programs under tight deadlines. We consider maintainability as a standard of judgment in our research.

**Node** – Nodes hold information and point to other Nodes. Because computers are limited in the way they store data, Nodes are essential in data structures that grow and shrink.

**Profiler** – A computer program that works like a stopwatch, showing how much time is required to perform an algorithm. A timer is started before the algorithm is run, then stopped after the algorithm has finished. Profilers are extremely accurate because they count time according to the processor speed of the computer.

**Run-time** – The period of time when a computer algorithm is being executed. Shorter run-times are better than longer run-times.

**Run-time performance** – The efficiency of a computer algorithm. Our research considers run-time performance as a standard of judgment.

**Set** – An unordered data structure, like a set in Mathematics. Sets store and retrieve elements quickly in Hash Tables.

**Sorted Set** – An ordered data structure where elements are automatically sorted as they are added. This allows for quick searching through the structure.

**Standard Template Library (STL)** – The library of data structures, algorithms, and other useful code in C++. See Java System Library.

**Templating** – The C++ terminology for Generic Programming.

**Type checking** – Compilers do not allow objects of one type to be added into a data structure of a different type.

**Vector** – The C++ terminology for Array List.