Performance Evaluation of C++ and Java Programs for Portable Devices

Venkata Sai Smaran Vallabhaneni

Abstract

In the world of software development for portable devices, performance optimization is crucial for responsiveness and general ease of use. This is particularly true for embedded systems and mobile devices like smartphones, where memory and processing speed are often limiting factors. This paper presents a comparative analysis of the Java and C++ programming languages, focusing on the runtime of a broad range of algorithms and emphasizes the range in runtimes between the algorithms. The study, conducted via the Termux terminal emulator, aims to highlight the performance differences between Java and C++ implementations across various algorithms. It also underscores the importance of metrics such as runtime in the context of software designed for portable devices. The findings reveal a significant performance edge for C++ over Java, as the results consistently demonstrate a speed factor of up to 200 times in favor of C++, underscoring the potential for utilizing performance enhancements in software implementing C++ in certain task-heavy algorithms such as regression for resource-constrained devices. In the end, this study contributes valuable insights for developers seeking to optimize software efficiency on portable devices.

Introduction

As the demand for sophisticated and resource-efficient mobile applications continues to grow, developers face the critical decision of selecting the appropriate programming language for Android development. C++ and Java emerge as two of the most prevalent and optimized choices. Over the history of their development, C++ has had advantages over Java and likewise. This study investigates the difference in efficiency between C++ and Java using several algorithms commonly required by the Android development platform.

This study attempts to provide quantitative results regarding the efficiency of each programming language, which can help in determining the utility in using C++ or Java to develop applications on the Android platform. While Java is still required for the underlying development of Android applications, substituting C++ for resource intensive or I/O heavy algorithms can be beneficial. The experiments conducted in this research employ a comprehensive set of benchmarks run in Termux, a terminal emulator for Android, measuring execution times across various algorithms commonly encountered in mobile application development.

A similar study by Farzeen Zehra et al. comparing C++ with Python (usually known as an even slower language) on a computer shows C++ dominance in terms of execution times. In sorting algorithms, a best case scenario for C++ yielded double the runtime for Python, while the worst case scenario showed quite the opposite. Memory consumption is where C++ excelled, with a worst case using only half the memory Python used. Sorting, data insertion, and data

deletion algorithms show C++ clearly outperforming Python in both memory usage and runtime [9]. Lower end devices (such as the ESP32 and Pi Pico) can also benefit from running C++ code, despite its un-user friendliness [4].

The choice of programming language is a critical factor in optimizing resource utilization, response times, and overall user experience on relevant Android devices. Understanding the performance characteristics of C++ and Java is essential for developers seeking to enhance the efficiency of their Android applications. This study provides statistical insights into the current discourse surrounding programming language choices in mobile application development.

Methods

To determine the runtime of a program on Android, Termux, a terminal emulator, was used to run the programs. The programs themselves were made with standard C++ and Java libraries to ensure maximum compatibility with the wide variety of devices and prevent any outlier runs.

Termux was chosen for running the programs because it does not add an extra performance overhead that other methods of running might add. In other words, the programs are running directly on the Android system and its compilers, and not through a custom compiler some IDEs might add. Furthermore, it eliminates the need of a computer to be connected, as Android Studio requires a PC to transfer and compile the program.

It is also a better choice to use Termux over Android Studio as the latter is more designed for running mobile applications and not regular programs, like we can on a computer. Using Android Studio would have also not allowed us to test C++ without Java and Kotlin interfering, as in the real world, C++ ideally works with Java.

The devices tested are the Samsung Galaxy Tab A (2015, Qualcomm Snapdragon 410), the Galaxy Tab S6 (2019, Qualcomm Snapdragon 855), the Galaxy J7 Nxt (2015, Samsung Exynos 7570 Octa), the Galaxy A30 (2019, Samsung Exynos 7904 Octa), the Lenovo Tab M8 (2019, MediaTek Helio A22), and the Vivo Y35 (2022, Qualcomm Snapdragon 680).

The program's runtime was determined using the chrono library in C++ and the java.time.Instant and java.time.Duration libraries in Java, and all of these libraries are part of the standard C++ and Java package. Furthermore, the algorithms were programmed manually rather than using in-built methods to make sure that implementations were the same in both languages and prevent any unintended advantages. Any special language-dependent features that make one language faster or slower than the other are not used. The unintended advantage can come from one method using special functions for a specific language or even from using a different algorithm. In other words, the programs were written as they would have been on a computer with no external libraries; only the algorithm itself, the variables containing the data,

the time tracking code, and the print statements. Those algorithms in both C++ and Java can be found here: https://github.com/happysmaran/ScientificResearch.

In total, there are five algorithms: selection sort, quick sort, and insertion sort for showing worst case sorting tests, merge sort for showcasing ideal sorting, and linear regression for mathematical operations. Each of these algorithms were made in Java and C++, and each program was run a total of nine times, meaning that each device had a total of ninety runs: 45 in C++ and 45 in Java. There is a one second delay between each run, with C++ and Java programs running separately with a sizable gap between a C++ session and a Java session. This is to control the thermal load on the devices, especially older models with slower processors and subpar cooling.

One thing to note was that in all the programs, the same dataset was used. For the sorting algorithms, the same list of 1000 unsorted numbers was used. For the regression algorithm, an adaptation of the 1000 item list was made, with the x and y coordinate list containing that same list. This is to make sure that the test cases are as equal as possible for comparison.

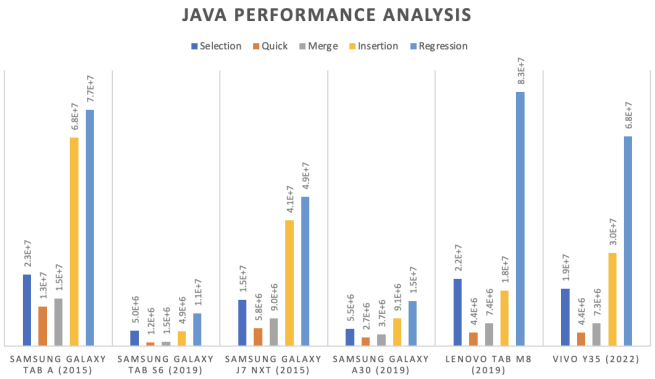
After the data was collected, the average runtime for C++ and Java in each program was calculated for that particular device, where the individual runs were averaged for general comparison between the two languages (figures 1 and 2), and those averages were then averaged again for the performance difference value (figure 5). The standard deviation of the original results were also calculated (figures 3 and 4). After that, the difference in performance was calculated as the performance of C++ divided by the performance of Java, since C++ ran faster in every test.

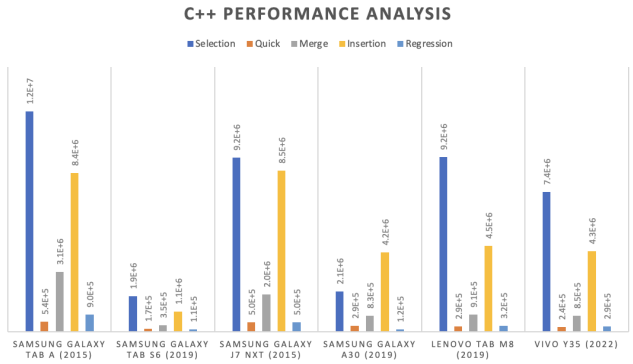
Results

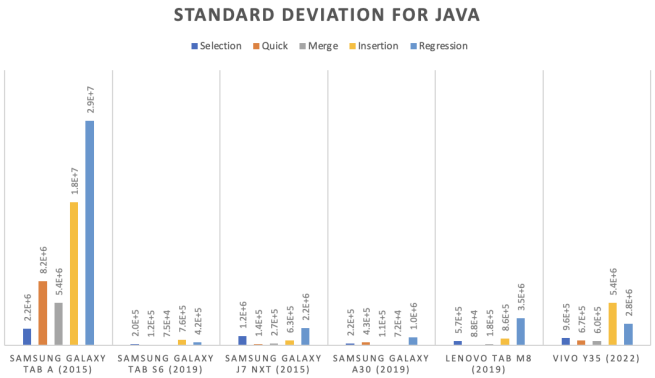
The results from each individual test showed a significant performance advantage for C++ over Java on all devices. On average, C++ programs outperformed (in terms of runtime speed) Java counterparts by factors ranging from 2 to 200. The difference in performance was particularly pronounced in algorithms involving heavy computation or data manipulation, such as sorting and regression.

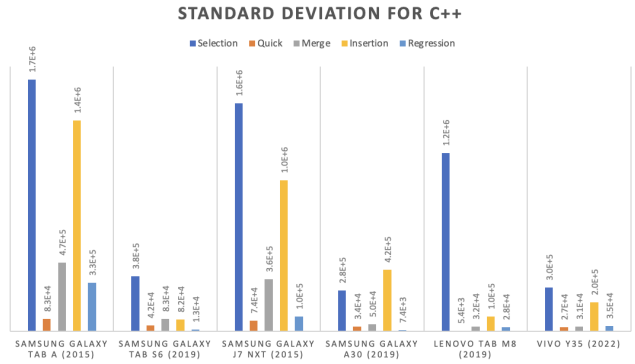
For instance, in the selection sort algorithm, C++ exhibited an average runtime that is 2.68 times faster than Java across all devices. Similarly, in the quick sort algorithm, C++ achieved an average runtime approximately 7.13 times faster than Java. Regression achieved an average runtime speed difference of 178.83. However, it is important to note that algorithms that are already optimized (in other words use strategies such as divide and conquer to solve tasks), such as merge sort and quick sort, saw a smaller difference compared to less optimized algorithms, such as insertion sort. Quick sort managed to have the least amount of variance

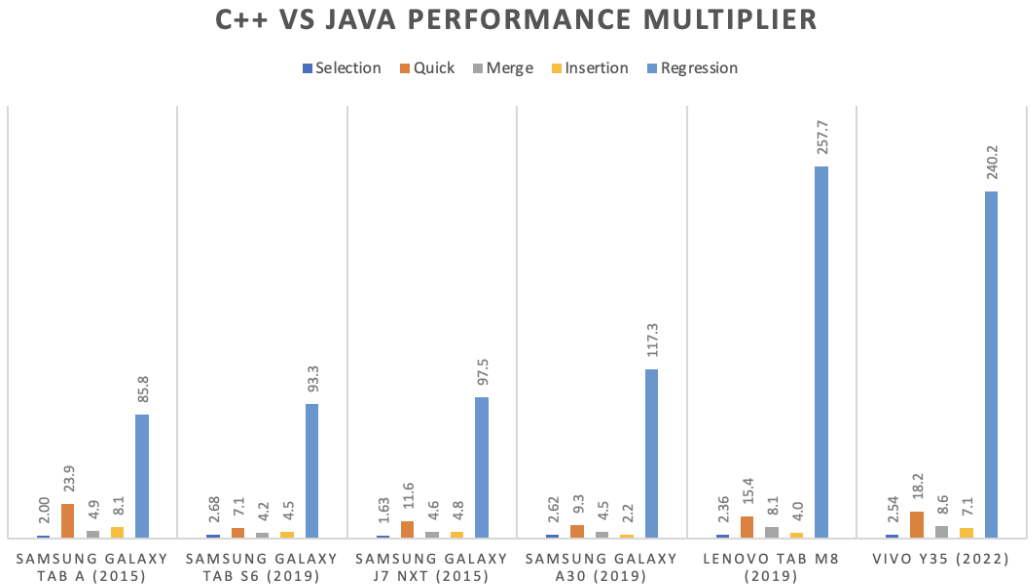
because of the list's small size. Larger lists would widen the gap between mergesort and quicksort.











However, the standard deviation analysis reveals that C++ programs generally exhibited higher variability in runtime compared to Java programs, indicating greater consistency in favor of Java across multiple runs. This is because Java, despite having a similar way of calling code and using memory, it has a generally stable and secure virtual machine for running code, as Gayathri Kandasamy Sengottaiyan mentions in his paper on Memory management in C++ and Java: "Like in C++ java is also using the call stack to load the run able subroutine. Java is fully object oriented so it have several class. In C++ compiler will be responsible for stack allocation, here it [is] taken care by "Java Virtual Machine (JVM)" [6]. The Java will load the corresponding byte code which is a run-able code and it allocate memory as structured memory."

Java's stability via JVM can also help in larger datasets. In Omar Khan Durrani's paper comparing the same algorithms but against Python on a computer, he revealed that while Python is faster in smaller datasets, it gets much slower with larger ones, with Java outpacing it by several factors [2].

Moreover, Java can get closer to C++ in terms of speed and efficiency with the help of third party libraries. Bogdan Oancea et al. shows that with a Java library specifically developed for Matrix computations, Java manages to run faster [5].

However, C++ can also be made to run more securely in terms of memory management. Kostya Serebryany et al. mentions that memory tagging can be used to improve C and C++ memory safety, with cooperation between compiler and runtime of course [7].

Furthermore, if testing was done with larger data sets, then this difference would be more uniform and possibly bigger with less efficient algorithms such as selection sort. For this study, lists of 1000 integers were used, since Java's arr[] datatype did not accept a list of 10000 elements. If more efficient/versatile data types were used, then a larger list could be tested.

These findings suggest that C++ does offer substantial performance advantages over Java for these resource-constrained devices, with the cost of performance stability, as shown in figures 3 and 4 (C++ Standard deviation is much worse compared to Java). Furthermore, since C++ requires developers to manage all aspects of code, including compilation, a compiler that is not as performant can inefficiently compile the code, removing the speed advantage [8]. Along with this, if there is any case where resources are at such a constraint that OOP concepts may not be viable, C++ as well as other OOP languages may suffer performance issues [1]. Developers seeking to optimize software performance on such devices may benefit significantly from utilizing C++ for algorithms requiring intensive computation or memory usage. This can possibly be done by sharing data with Java programs and C++ programs in an app, and any resource-intensive calculations that need to be done can be passed off from Java to C++ through cached files. Alternatively, a more easier method is to use the Android NDK platform that Google provides, which allows the use of C++ directly in Java/Kotlin for its benefits while still having access to Java/Kotlin and their APIs.

Discussion

In conclusion, the comparative analysis between C++ and Java programs conducted in this study provides valuable insights into the performance dynamics crucial for software development on portable devices. The results consistently showcase a notable performance edge for C++ across various algorithms, ranging from 2 to 200 times faster runtime compared to Java counterparts. This was most apparent in tasks involving heavy computation, such as sorting and regression. However, it's essential to note the higher variability in runtime is exhibited in C++ programs, indicating a trade-off between performance and stability. Furthermore, security is sacrificed in C++ as well as memory integrity, making C++ vulnerable to exploits from bugs or viruses, while Java's JVM protects it. Nonetheless, these findings underline the significance of selecting the appropriate programming language, with C++ emerging as a favorable choice for resource-constrained devices, especially when optimizing for performance in algorithmically intensive tasks.

Moving forward, this study provides insight to the ongoing discourse surrounding programming language choices in mobile application development. By offering quantitative assessments of runtime and memory usage, developers gain valuable insights into the practical implications of selecting C++ or Java for Android development. The results suggest that while Java remains integral for Android application development, integrating C++ for resource-intensive algorithms can significantly enhance overall efficiency. Hence, future research pertaining to Java and C++ comparisons can focus on how memory stability and security differs between the two. It can also be done with a wider variety of hardware and more runs, some with less available memory to see how Java can perform. How the Object Oriented system is developed between the two languages can also be checked, as they can also cause performance discrepancies by one structure possibly being "better" than the other [8].

It should also be mentioned that Java's performance loss has the opportunity of being exaggerated on older, lower-end devices simply due to the fact that there are so many other items contributing to Java's runtime: The JVM, the garbage collector, heap sizes, memory allocated, even the input can all affect a Java program's runtime performance [3].

It can also be beneficial to analyze this performance difference on devices that do not support Java natively. As an example, iPhones do not have JVM in favor of Apple's native Swift language. JVM, however, can be installed on these mobile devices through jailbreaking.

It should also be mentioned that optimization plays a critical role in development. Certain features available exclusively on either C++ or Java can give an advantage over the other. Furthermore, optimization made through general programming techniques such as memoization instead of recursive algorithms and nlogn efficiency programs can reduce this performance gap to give Java a level playing field. Furthermore, research can be done towards library compliance with C++, as currently many essential libraries are platform dependent. Java is not.

References

[1] Chatzigeorgiou, Alexander. *Performance and power evaluation of cpp object-oriented programming in embedded processors*. U of Macedonia, 2015.

[2] Durrani, Omar Khan and Sayed AbdulHallan. *Performance measurement of popular sorting algorithms implemented using java and python*. Ghousia College of Engineering, pages 5–6, 2023.

[3] Georges, Andy, Dries Buytaert, et al. *Statistically rigorous java performance evaluation*. Ghent U, 2007.

[4] Plauska, Ignas. *Performance evaluation of c/c++, micropython, rust and tinygo programming languages on esp32 microcontroller*. Kaunas U of Technology, 2022.

[5] Oancea, Bogdan, Ion Gh. Rosca, et al. *Evaluating java performance for linear algebra numerical computations*. Nicolae Titulescu U, 2010.

[6] Sengottaiyan, Gayathri Kandasamy and Tarik Eltaeib. *Memory management in c++ and java*. U of Bridgeport, pages 3–5, 2015.

[7] Serebryany, Kostya, Evgenii Stepanov, et al. *Memory tagging and how it improves c/c++ memory safety*. Google, page 5, 2018.

[8] Wu, Pei-Chi and Feng-Jian Wang. *On efficiency and optimization of c++ programs*. National Chiao Tung U, 1996.

[9] Zehra, Farzeen, Maha Javed, et al. *Comparative analysis of c++ and python in terms of memory and time*. NED U of Engineering and Technology, 2020.