Andrew Kim
Kobe Lee
Celine Tran
CS 4080
Final Project

I. Abstract

Type coercion is described as the conversion of a type of object to another with similar content. Type coercion/conversion is common knowledge in the computer science field, but the specific effects it produces requires some research. In order to understand type coercion/conversion as more than just a convenient way to work with mixed-types, our project is designed to magnify its effects. We decided to utilize a bubble sort algorithm for our first algorithm and a very intense formula for the second algorithm. The former would highlight discrepancies in time efficiency in three different parts while the latter would highlight discrepancies in accuracy. For the bubble sort algorithm, we manipulated the way in which the input data was handled. One method allowed mixed types and used type coercion before sorting it, while one converted an input set of doubles into ints and the last would simply sort a double array. The second algorithm worked with doubles exclusively. But while one method converted all doubles to ints before calculating the formula, the second simply passed the doubles to the function. The experiment was conducted in three different programming languages: Java, Python, C++. Each language showed some key differences in how they handle mixed types. While one language's structure benefited from coercion/conversion, another would take a noticeably longer amount of time. The experiments produced sufficient results, showing both the pros and cons of type coercion/conversion, raising a question of whether it is helpful at times, or if it might prove to be an obstacle.

II. Introduction

The focus of our study is the effects number type coercion/conversion on time efficiency and accuracy of results when utilized in both numerical sorting algorithms as well as mixed type operations. In order to take a commonly accepted theory and magnify both its positives and negatives we have designed this experiment to be able to compare differences in time efficiency and accuracy. While using the programming languages of Java, Python, and C++, we will also be able to find unique features of each language and their methods of handling mixed type data sets.

III. Methods, Algorithms, and Concepts

The primary method by which we obtained data is by implementing two primary algorithms across our three languages. This was done in order to establish two important pieces of information: the time efficiency and the accuracy of type coercion.

1. Algorithm 1

The first algorithm that we implemented was used to establish the time efficiency differences of sorting with type coercion, mixed type operations, and no type operations. We created randomly generated arrays of sizes 250, 500, 750, and 1000; from there, we split the procedure into three parts. The first parts utilized type coercion, and employed a narrowing type conversion by explicitly typecasting the random values of float type to integers; from there, we used a bubble

sort to sort all of the data. The type coercion took place at each index within the sort function. The second part left the randomly sorted data as is, as type float, and used the same bubble sort to sort all of the data. The final part took an array of half ints and half doubles to test mixed-type time efficiency, and then used the same bubble sort. With each of these parts, we ran ten trials and calculated the average completion time to ensure that the data was reliable.

2. Algorithm 2

The second algorithm that we created was established to point out the loss of accuracy within type coercion. We created a generic formula, slightly resembling the quadratic formula, that goes as follows:

```
-a^2 + (b^2+c^2-4ac)/(2ac+cab)
```

From there, we created three randomly generated lists of floats of size 100. We plugged in the values to the formula in two halves once again. The first half simply plugged the values in as doubles into the formula without any type conversion involved. The second half type-casted each of the three arraylists' random values at the given index to an integer, and performed the function. We then took the absolute value of the difference between the two halves' values to determine the difference that we retrieved between type coercion and non-type-coercion.

IV. Experiments

As established within the previous section, the experiment was carried out with two main algorithms. Within the execution, there were a few variables that we made sure to remain weary of. The first of which was uniformity; in order to overcome issues resulting from different computers and therefore varied processing speeds, we decided to utilize the online compiler from Repl to run our trials. Another factor that we considered was the uniformity of randomized data ranges, as well as the general output. The randomly generated arrays were doubles of the range 1 to 1000. The output of Algorithm One was of the following format:

```
With Type Conversion --
Average time of 10 trials of size 250 is ___ seconds
Average time of 10 trials of size 500 is ___  seconds
Average time of 10 trials of size 750 is ___  seconds
Average time of 10 trials of size 1000 is ___  seconds
Without Type Conversion (Original List) --
Average time of 10 trials of size 250 is ___  seconds
Average time of 10 trials of size 500 is ___  seconds
Average time of 10 trials of size 750 is ___  seconds
```

```
Average time of 10 trials of size 1000 is ___  seconds
With Mixed Types (Type Coercion) --
Average time of 10 trials of size 250 is ___  seconds
Average time of 10 trials of size 500 is ___  seconds
Average time of 10 trials of size 750 is ___  seconds
Average time of 10 trials of size 1000 is ___  seconds
```

The output of Algorithm Two was established to be of this format (with 100 entries):

```
Original Value: _____
Typecasted Value: ____
Difference: ____
```

V. Results

1. Python Findings

Python is a dynamically typed language. Within weakly typed languages, variables could be coerced implicitly into unrelated types; in strongly typed languages, this is not possible, as there needs to be explicit conversions that must take place. Within the mixed-type operation under algorithm one, within the bubble sort, the data of types int and float were not converted to be directly compared with one another. Python is known to be a "duck typed" language because it allows this to take place without the need for strong typing. For this reason, the mixed-typed and the original float arrays had roughly equivalent execution times within the first algorithm. This can be deduced from the fact that there was no type conversion of any kind that took place within either of the operations. As for the array that was converted into integers at each index from float values, it took about 30% longer than the other two types of arrays that were dealt with. This follows with the concept of dynamic typing, as this required explicit type conversions from float to integer. The "duck typed" aspect of the Python language is what allows for mixed type operations to be done so efficiently; integers and floats can be compared without any problem, whereas in Java, it would create an issue. For that reason, in real world usage, Python would never have to strongly type integers into floats or vice versa within comparison; this is beneficial, considering the inefficiency of strongly typed conversions.

Within algorithm two, the findings within Python were more or less the same as those discovered in the other two languages. It was found that the loss of accuracy that takes place within a widening conversion under the given circumstance varied immensely. The differences ranged from a mere few points to over a thousand points. This certainly makes sense, given that with all of the squaring and multiplying done, the loss of significant decimals can cause dramatically different results.

2. Java Findings

Java is a statically typed language, meaning every variable is bound to a type and an object. The language itself is very restrictive, requiring all variables, methods, and objects to have a specific type bound to it at all times. Java's restrictive nature was shown clearly when compared to Python and C++'s method handling mixed types. While Python allowed ints and doubles to be put into the same array/list without any complication, Java put a spin on it. First, the array itself had to be typed, so we typed it as a double so we would not have to worry about any loss of accuracy. Then we added in the doubles followed by the ints into the array of type double. When printed out, we noticed that while Python and C++ show the ints as ints and the doubles as doubles. But for Java, every single int was converted into a double through type coercion.

This finding explained why for algorithm 1, the type coercion method took the longest amount of time to run. This method involved having half ints and half doubles, which would then be put into an array of type double and finally go through a bubble sort algorithm. The other two methods that involved type conversion and no conversion were similar in times, with the no conversion method being faster sometimes. This may be due to both methods using the same input arrays because they both only work with doubles. We were able to conclude that for Java, type coercion slows down the time efficiency at a noticeable rate.

For algorithm 2, we found that regardless of language, there were very noticeable differences in the answers from not type-casting and type-casting. We found that for some cases, the difference was single digits, but some were four digits. Keeping in mind that we limited the random number generator to values up to 1000, a difference of four digits is quite alarming.

3. C++ Findings

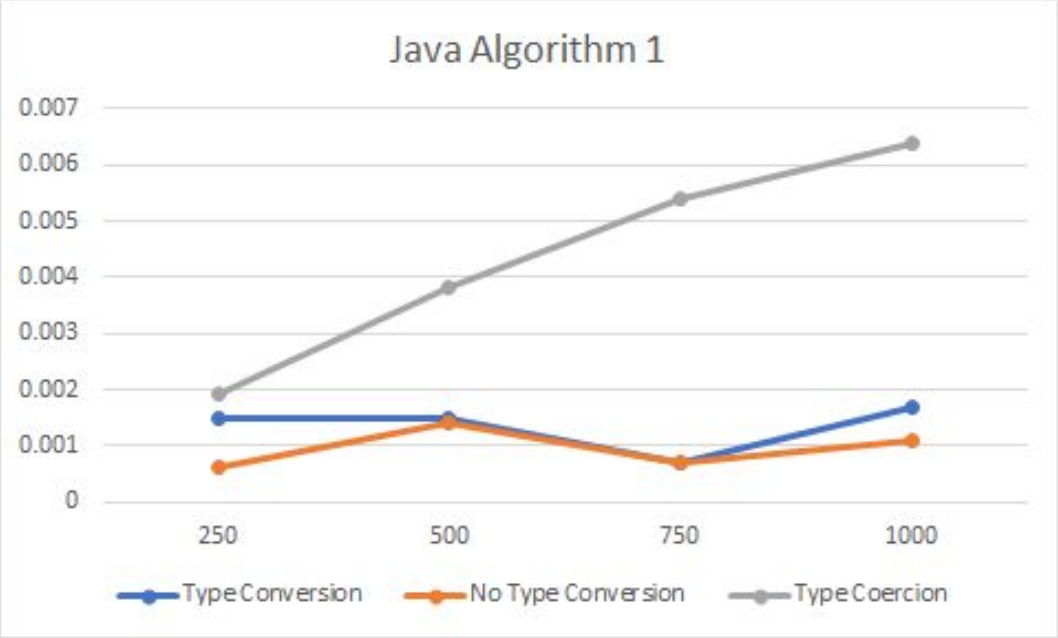
C++ is "not a strongly typed [language] because [it includes] union types, which are not type checked. Because C++ deals grealy with coercion and has many assignment type coercions, it's less reliable and its error detection is not as effective as other languages like Java or C#. C++, like Java, allows mixed-mode assignment only if the required coercion is widening, however, C++ applies coercion freely.

This is exemplified in algorithm 1, when printing out the mixed sort vector. While the printed out mixed sort vector looks similar to Python, C++ does like Java implicitly convert types when necessary. So the integer values are converted doubles and then put into a vector of doubles. However, the outstream in C++ by default does not print the floating point so there is an extra step needed to add the precision points. Because of this, the time efficiency results are similar to that of Java where there was not a noticeable difference in time between sorting mixed types and sorting after types are converted. Because we did use an online IDE, we had to settle for smaller data sizes due to the slower compiler. With a greater data size, there would be a greater difference between the result of sorting without type conversion and with mixed types.

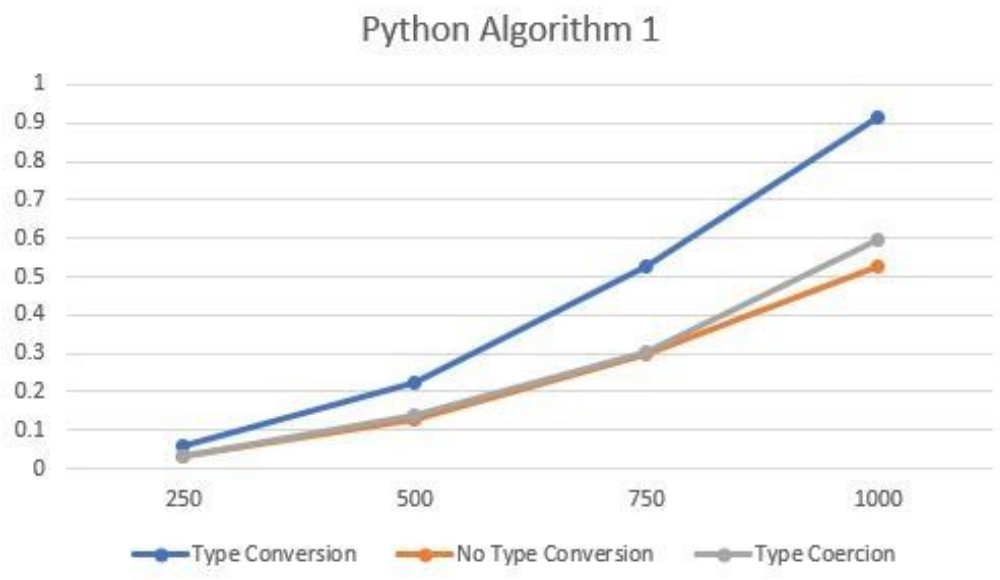
Mixed types is the slowest of the three because it has to sort the doubles requiring a floating-point comparison and also doing a conversion.

Algorithm 2 demonstrates the loss of precision when typecasting in mixed-operations. When computing large numbers in the hundred thousands like in our experiment, the difference in results are significant enough to make typecasting insufficient.

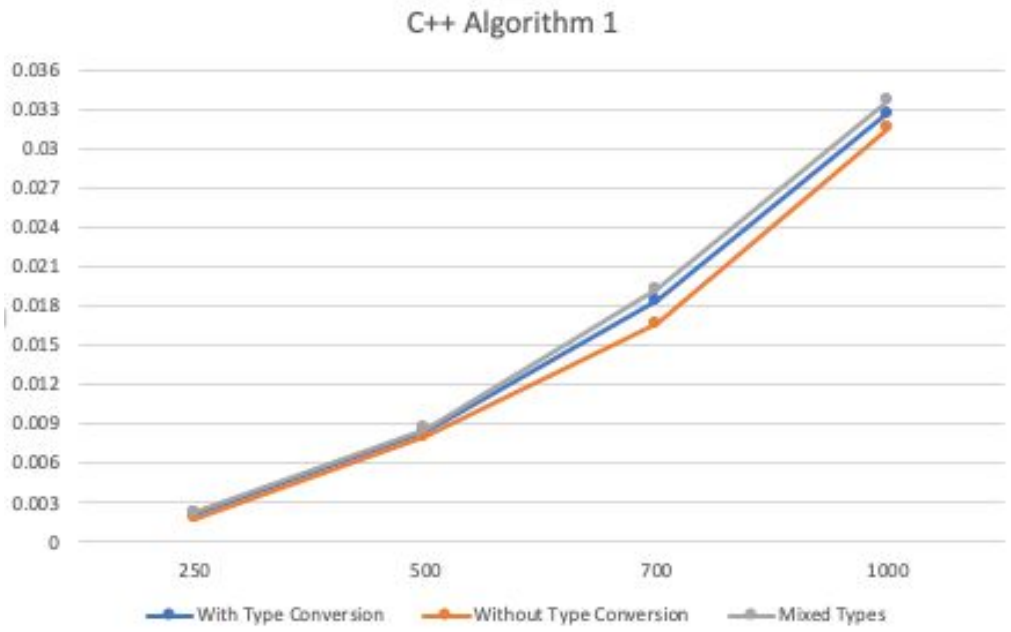
4. Results



Size	Type conversion	No Type Conversion	Type Coercion
250	.0015s	.0006s	.0019s
500	.0015s	.0014s	.0038s
750	.0007s	.0007s	.0054s
1000	.0017s	.0011s	.0064s



Size	Type Conversion	No Type Conversion	Type Coercion
250	0.05894s	0.02976s	0.03354s
500	0.22335s	0.12972s	0.1368s
750	0.52804s	0.29986s	0.30176s
100	0.91268s	0.52849s	0.59813s



Size	**Type Conversion**	**Without Type Conversion**	**Mixed Types (Type Coercion)**
250	0.0021791s	0.0017459s	0.0022423s
500	0.0084766s	0.0079997s	0.0086677s
750	0.0184304s	0.0166389s	0.0193131s
1000	0.0326175s	0.0314722s	0.0336055s

VI. Future Work

Due to our limitations, using an online IDE, our largest data size was only 1000, which can easily be handled in a modern CPU. For more visible differences in time, we'd want to test with much larger data sizes.

Type theory is a "broad area of study in mathematics, logic, computer science, and philosophy"(Sebesta 2019 494). Type theories in computer science defines the types and type rules of programming languages. Type theory helps model the structure of data types in programming languages so while static type systems like Java allow for better error detection, it is not as flexible as dynamic type languages such as Python. Future works would include finding the balance between the two, strongly typed and weakly typed programming languages.

In an article written by Alex Wright, he makes known that "researchers have been exploring type systems capable of capturing a greater range of programming errors such as the public exposure of private data"(Wright 2010). In our increasingly interconnected age, the

problem of data security has posed a real question about how reliable cybersecurity is. Wright discusses the development of a security type system that could “apply the same principle of semantic checking to determine the owner of a particular piece of information”, different from the traditional type systems which enforces rules by assigning values to data types (Wright 2010). Wright also proposes the application of type checking in hybridizing type systems and theorem provers, allowing for certain logical restraints to be refined.

VII. Conclusion

While our experiment scrapes the surface of type theory, our results clearly show the difference between strongly typed and weakly typed programming languages. Our project looks at how different languages handle explicit and implicit type conversions, looking at their efficiency in sorting and accuracy in mixed operations. In a strongly typed language like Java, where type coercion does occur, it takes a significantly longer time to sort the data when comparing it to when we do not use type conversion. Whereas in a weakly typed language like Python, the time difference between sorting with type coercion and without type conversion, is not significant due to the fact that Python does not implicitly convert data types. With C++, while it is not a strongly typed language, it is more static than Python, causing it to be less reliable when handling type coercion. The results exemplify that when type conversion is used, we compromise efficiency. Similarly, when we typecast the values, we lose precision. These are all factors that play into type theory and its role in each programming language.

Sources Cited

G. D. Plotkin, "Type theory and recursion," [1993] Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science, Montreal, Quebec, Canada, 1993, pp. 374-, doi: 10.1109/LICS.1993.287571.

M. Abadi, L. Cardelli, B. C. Pierce, and G. D. Plotkin. Dynamic typing in a statically-typed language. In phPOPL, pages 213--227, 1989

M. Salib. Starkiller: A static type inferencer and compiler for python. In Master's thesis, MIT, 2004.M. Salib. Starkiller: A static type inferencer and compiler for python. In Master's thesis, MIT, 2004.

Sebesta, Robert W. *Concepts of Programming Languages*. 12th ed., Pearson, 2019.

Wright, Alex. "Type Theory Comes of Age." *ACM*, 1 Feb. 2010, cacm.acm.org/magazines/2010/2/69367-type-theory-comes-of-age/fulltext.