Type Coercion and Efficiency

Andrew Kim, Kobe Lee, Celine Tran
Professor Talari

Introduction

Problem: How does each programming language tackle mixed-type operations as well as converting certain types into others?

Topics of Comparison

- time efficiency between different methods of handling inputs for bubble sort
- accuracy between different methods of handling inputs for custom formula

Languages used: Python, Java, C++

Methods/Algorithms/Concepts

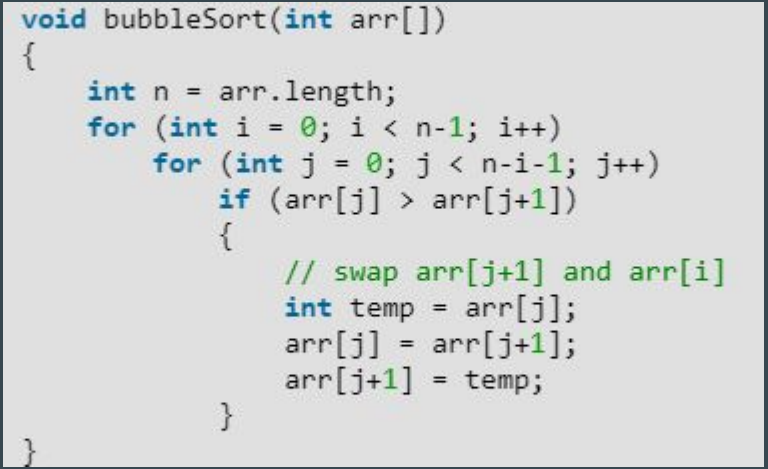
We chose to implement two primary algorithms across our three languages in order to establish the time efficiency and accuracy of type coercion.

Algorithm 1: Comprised of three parts. One part involves randomly generated arrays of doubles, one part involves an array of doubles that is typecasted to an int, and the last part involves an array of half integer half double that is used for mixed type operations. A bubble sort is then used on all three with sizes of 250, 500, 750, and 1000. We then take the average of 10 trials of each.

Algorithm 2: Create three randomly generated arrays of double type, and plug into a pre-established formula in two parts. The first part plugs in the value as doubles (the original), and the second part converts everything to integer. We then compare the two at each index to verify precision and/or the loss of data.

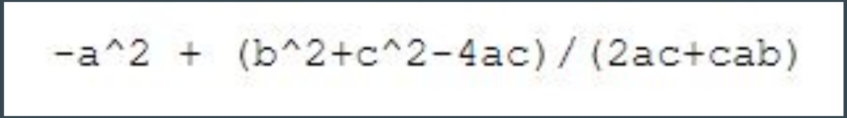
Experiments

Bubble Sort Algorithm (O(n^2))



```
void bubbleSort(int arr[])
{
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
            {
                // swap arr[j+1] and arr[i]
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

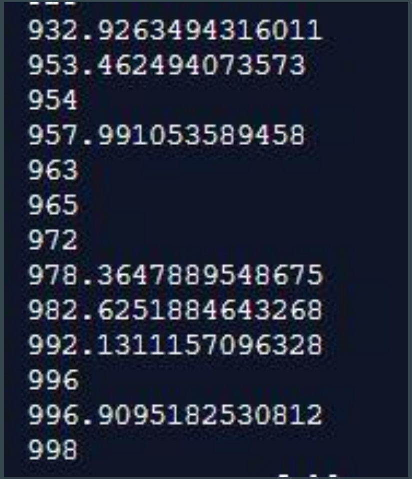
Algorithm 2 Formula



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

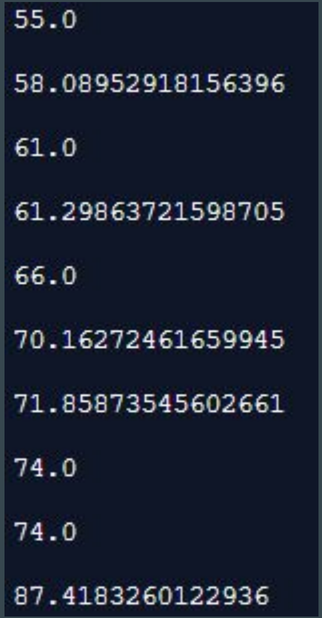
(All of the code was compiled through Repl.it)

Mixed Type Operation Differences Across Languages



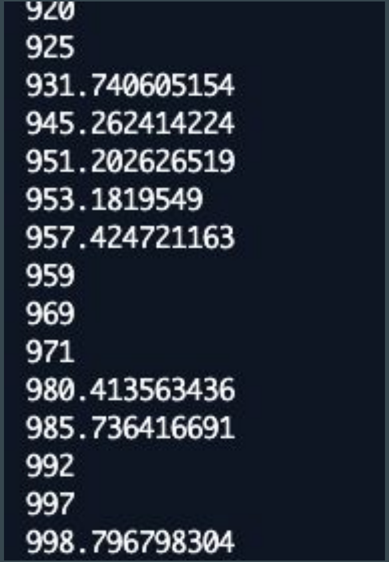
```
932.9263494316011
953.462494073573
954
957.991053589458
963
965
972
978.3647889548675
982.6251884643268
992.1311157096328
996
996.9095182530812
998
```

Python



```
55.0

58.08952918156396

61.0

61.29863721598705

66.0

70.16272461659945

71.85873545602661

74.0

74.0

87.4183260122936
```

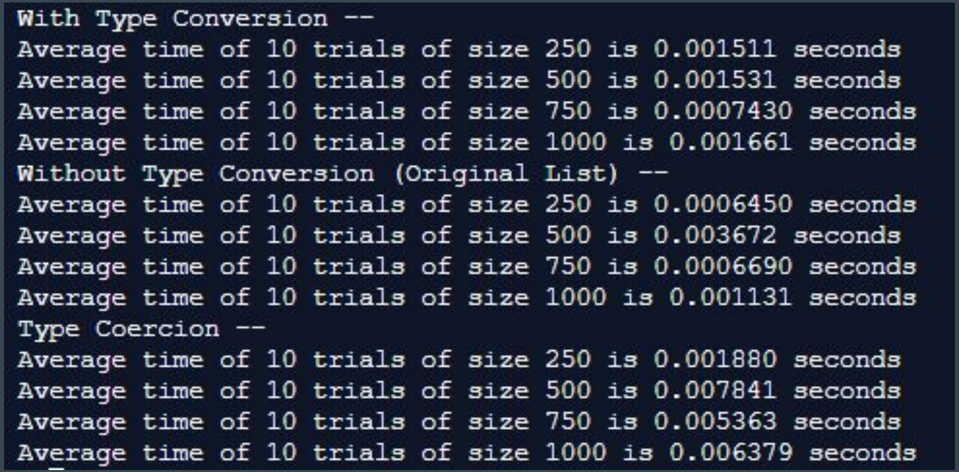
Java



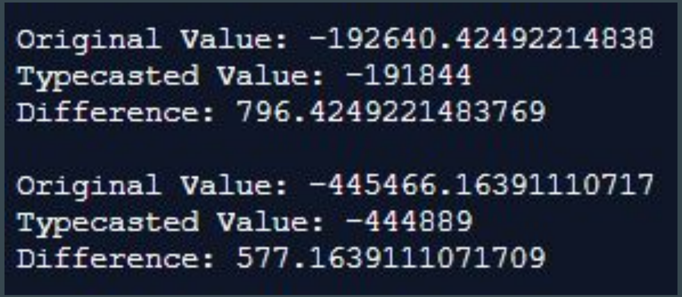
```
920
925
931.740605154
945.262414224
951.202626519
953.1819549
957.424721163
959
969
971
980.413563436
985.736416691
992
997
998.796798304
```

C++

Results: Java



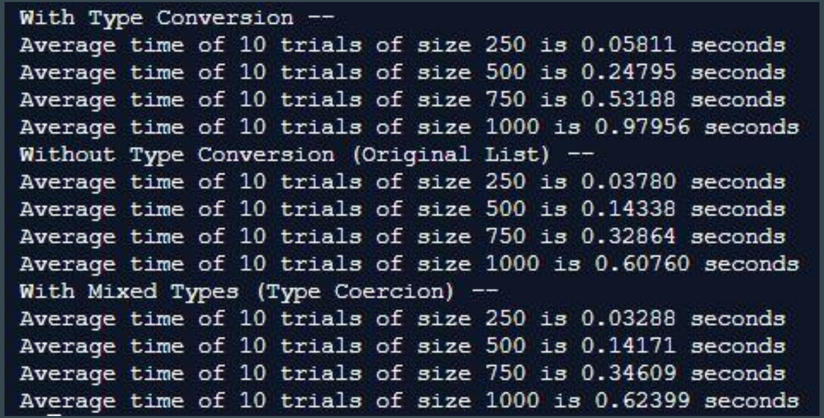
```
With Type Conversion --
Average time of 10 trials of size 250 is 0.001511 seconds
Average time of 10 trials of size 500 is 0.001531 seconds
Average time of 10 trials of size 750 is 0.0007430 seconds
Average time of 10 trials of size 1000 is 0.001661 seconds
Without Type Conversion (Original List) --
Average time of 10 trials of size 250 is 0.0006450 seconds
Average time of 10 trials of size 500 is 0.003672 seconds
Average time of 10 trials of size 750 is 0.0006690 seconds
Average time of 10 trials of size 1000 is 0.001131 seconds
Type Coercion --
Average time of 10 trials of size 250 is 0.001880 seconds
Average time of 10 trials of size 500 is 0.007841 seconds
Average time of 10 trials of size 750 is 0.005363 seconds
Average time of 10 trials of size 1000 is 0.006379 seconds
```



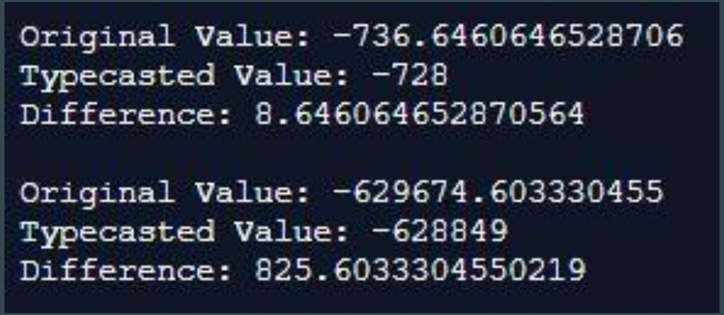
```
Original Value: -192640.42492214838
Typecasted Value: -191844
Difference: 796.4249221483769

Original Value: -445466.16391110717
Typecasted Value: -444889
Difference: 577.1639111071709
```

- Java is a statically-typed language, meaning every variable is bound to a type and an object
- Type coercion led to longer runtimes
- Type coercion took the longest because Java automatically converts ints to doubles when added to an array of type double
- Type Conversion and No Conversion methods were similar faster than the Type Coercion method
- converting from type double to int in algorithm 2 produced very noticeable differences in answers

Results: Python



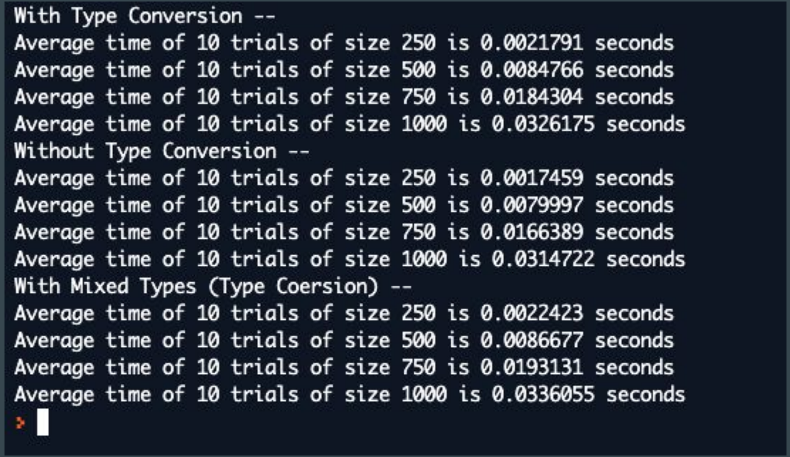
```
With Type Conversion --
Average time of 10 trials of size 250 is 0.05811 seconds
Average time of 10 trials of size 500 is 0.24795 seconds
Average time of 10 trials of size 750 is 0.53188 seconds
Average time of 10 trials of size 1000 is 0.97956 seconds
Without Type Conversion (Original List) --
Average time of 10 trials of size 250 is 0.03780 seconds
Average time of 10 trials of size 500 is 0.14338 seconds
Average time of 10 trials of size 750 is 0.32864 seconds
Average time of 10 trials of size 1000 is 0.60760 seconds
With Mixed Types (Type Coercion) --
Average time of 10 trials of size 250 is 0.03288 seconds
Average time of 10 trials of size 500 is 0.14171 seconds
Average time of 10 trials of size 750 is 0.34609 seconds
Average time of 10 trials of size 1000 is 0.62399 seconds
```



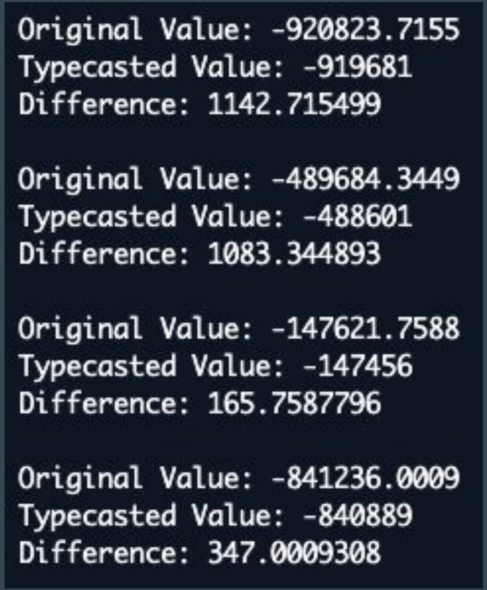
```
Original Value: -736.6460646528706
Typecasted Value: -728
Difference: 8.646064652870564

Original Value: -629674.603330455
Typecasted Value: -628849
Difference: 825.6033304550219
```

- As can be seen, Python does not type integers/doubles when doing mixed operations; they can be directly compared
- For this reason, the mixed type and original lists had nearly identical execution times
- The version with type conversion took longest because it required dynamic typing, as Python is a dynamically-typed language
- In a weakly typed language like Python, variables can be implicitly coerced to unrelated types, whereas in a strongly typed language they cannot, and an explicit conversion is required

Results: C++



```
With Type Conversion --
Average time of 10 trials of size 250 is 0.0021791 seconds
Average time of 10 trials of size 500 is 0.0084766 seconds
Average time of 10 trials of size 750 is 0.0184304 seconds
Average time of 10 trials of size 1000 is 0.0326175 seconds
Without Type Conversion --
Average time of 10 trials of size 250 is 0.0017459 seconds
Average time of 10 trials of size 500 is 0.0079997 seconds
Average time of 10 trials of size 750 is 0.0166389 seconds
Average time of 10 trials of size 1000 is 0.0314722 seconds
With Mixed Types (Type Coersion) --
Average time of 10 trials of size 250 is 0.0022423 seconds
Average time of 10 trials of size 500 is 0.0086677 seconds
Average time of 10 trials of size 750 is 0.0193131 seconds
Average time of 10 trials of size 1000 is 0.0336055 seconds
>
```



```
Original Value: -920823.7155
Typecasted Value: -919681
Difference: 1142.715499

Original Value: -489684.3449
Typecasted Value: -488601
Difference: 1083.344893

Original Value: -147621.7588
Typecasted Value: -147456
Difference: 165.7587796

Original Value: -841236.0009
Typecasted Value: -840889
Difference: 347.0009308
```

Algorithm 2

Typecasting the solution introduces ambiguity into our arithmetic expressions. We lose the precision in the answer so when dealing with large numbers like we did in the hundred thousands, the difference in value is significant enough to make typecasting our solutions insufficient.

Algorithm 1

- C++ is not as strongly typed as Java because there are union types that are not type checked
- As seen in slide 5, while looking similar to Python, C++ does implicitly convert types when necessary, however, C++ default does not print the floating point so we have to implement that. Because of this extra step of computation, it averages around the same time as type conversion.
- Type conversion leads to longer runtimes, therefore, the trial without type conversion is the quickest.

Future Works

- Testing with larger data sizes, because we were limited to an online IDE
- Type theory helps model the structure of data types in programming languages so while strongly typed systems like Java allow for better error detection, it is not as flexible as weakly typed languages such as Python
- Future works would include finding the balance between the two
    - "Researchers have been exploring type systems capable of capturing a greater range of programming errors such as the public exposure of private data" (Type Theory Comes of Age, Wright)
    - Security type systems - applying semantic checking to determine the information
    - Type checking also allows for hybridizing type systems and theorem provers
        - Refining certain logical restraints

Conclusion

This 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 noticeably 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.