Sorting Algorithms

Lou George
CMPS 112
Winter 2016

What I did:

Wrote 4 sorting Algorithms in Haskell, Java, and C.

- Quicksort
- Bubblesort
- Mergesort
- Insertionsort

Why these algorithms and Languages:

These algorithms have a good variety of run times.

The languages are very different from one another.

How I went about comparisons....

Randomly generate 10,000 random integers between 0 and 99.

Each time I generate 10,000 numbers I ran those numbers through all 3 versions of the given algorithm.

They all ran on the same system (2015 macbook pro 2.9 Ghz i5 dual core)

Tried to use what makes each language “unique”, and tried to make it readable, didn’t try to conserve space to sacrifice readability.

Quicksort

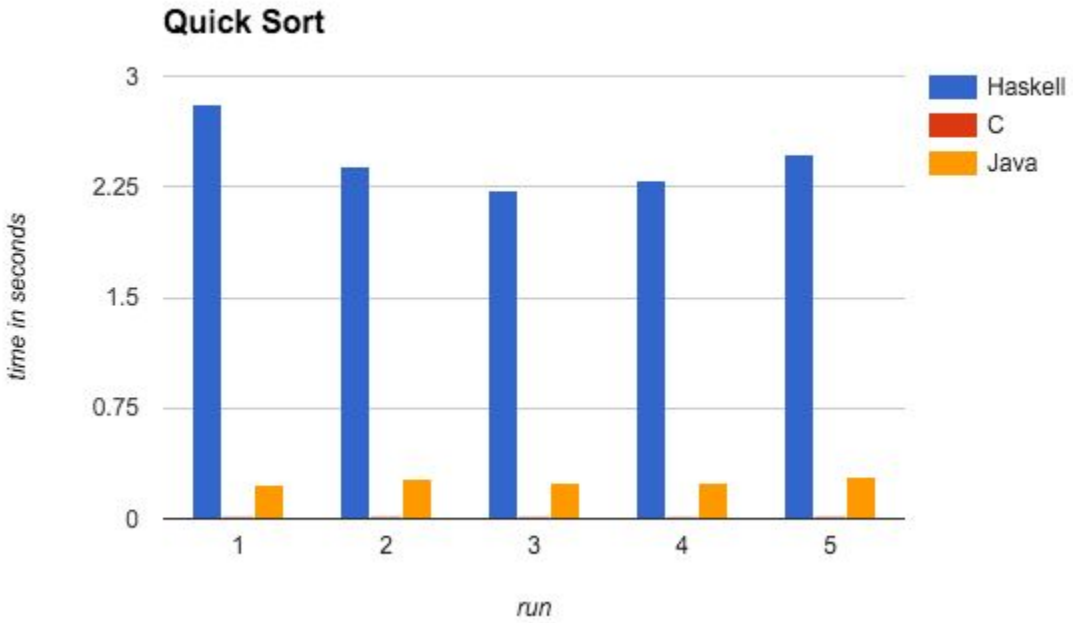
Not very consistent algorithm since at rare moments it can go up to squared time. But mostly (n*(log(n)) time

Average Runtimes:

Haskell: 2.4300 seconds
C: 0.0132 seconds
Java: 0.2552 seconds

Run	Haskell	C	Java
1	2.81s	0.013s	0.237s
2	2.39s	0.014s	0.272s
3	2.23s	0.013s	0.242s
4	2.29s	0.012s	0.245s
5	2.47s	0.014s	0.280s

O(n^2) but generally O(n(log(n))



Quicksort, looking at the code:

- Only 6 lines of code in Haskell!
- Less than 30 lines in Java.
- Got it 32 lines in C.

The algorithm is very compact, and ran very quickly across the board. and felt very natural for Haskell.

File Size

Haskell: 238 bytes
C: 1,251 bytes
Java: 1,701 bytes

```haskell
quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
    let smallerSorted = quicksort [a | a <- xs, a <= x]
        biggerSorted = quicksort [a | a <- xs, a > x]
    in  smallerSorted ++ [x] ++ biggerSorted
```

Mergesort

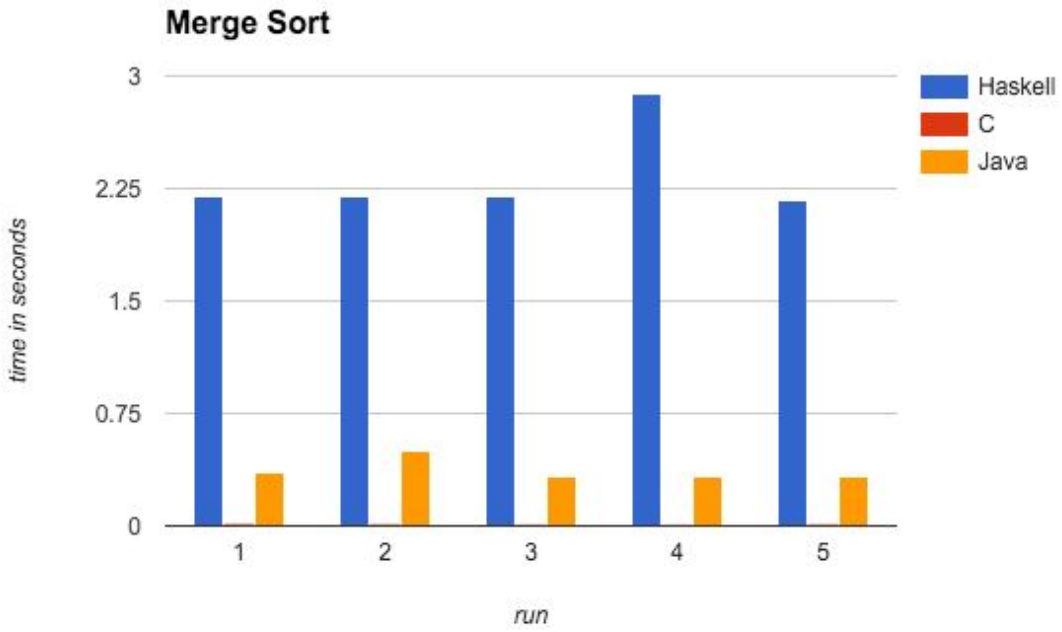
Divide and conquer algorithm, which relies on recursion. Good, and very consistent run time.

Average Runtimes:

Haskell: 2.3300 seconds
C: 0.0122 seconds
Java: 0.3714 seconds

Run	Haskell	C	Java
1	2.20s	0.015s	0.360s
2	2.20s	0.013s	0.505s
3	2.20s	0.011s	0.328s
4	2.88s	0.010s	0.331s
5	2.17s	0.012s	0.333s

O(n(log(n))



Mergesort, looking at the code:

- 16 lines of code in Haskell
- Less than 40 lines in Java.
- Less than 40 lines in C as well.

The algorithm is just very clever to begin with. I expected Haskell to do very well with this one, since it relies so heavily on recursion, and on average it was Haskell's fastest algorithm

File Size

Haskell: 539 bytes
C: 1,546 bytes
Java: 1,787 bytes

Insertion Sort

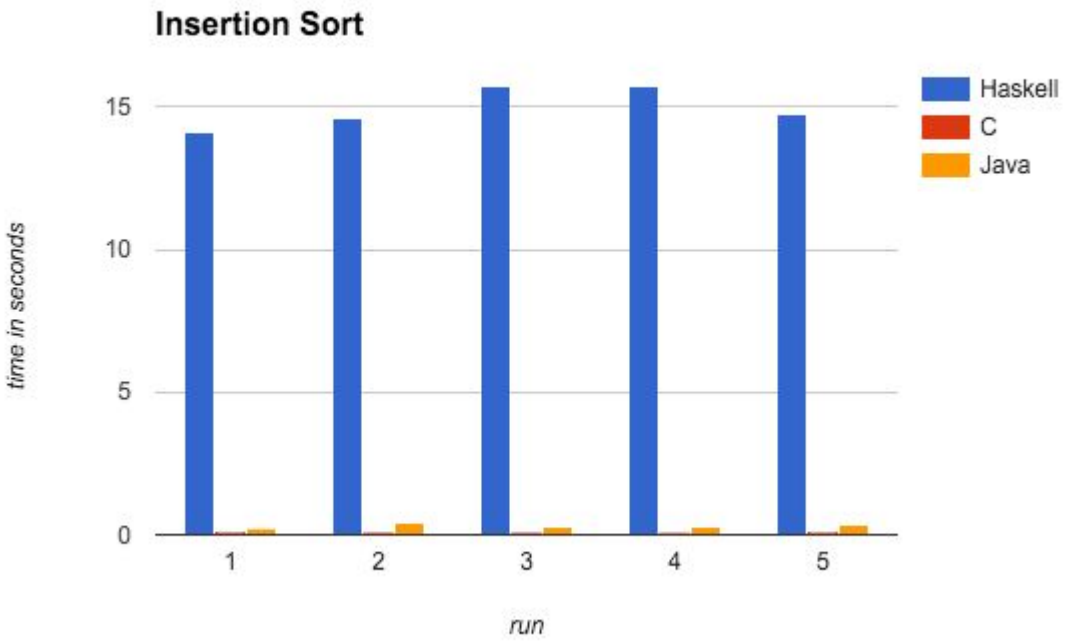
The more successful sibling of bubble sort. But none the less, not close to mergesort, and quicksort.

O(n^2)

Average Runtimes:

Haskell: 14.966 seconds
C: 0.0888 seconds
Java: 0.2998 seconds

Run	Haskell	C	Java
1	14.12s	0.096s	0.241s
2	14.61s	0.090s	0.390s
3	15.70s	0.088s	0.252s
4	15.69s	0.080s	0.250s
5	14.71s	0.093s	0.366s



Insertion sort, looking at the code:

- Haskell got it in 8 lines
- got it in 14 lines in Java.
- got it in 14 lines in C as well.

This algorithm is essentially brute force. Thus it’s consistently slow, although it still wasn’t the worst...

File Size

Haskell: 258 bytes
C: 878 bytes
Java: 1,215 bytes

Bubblesort

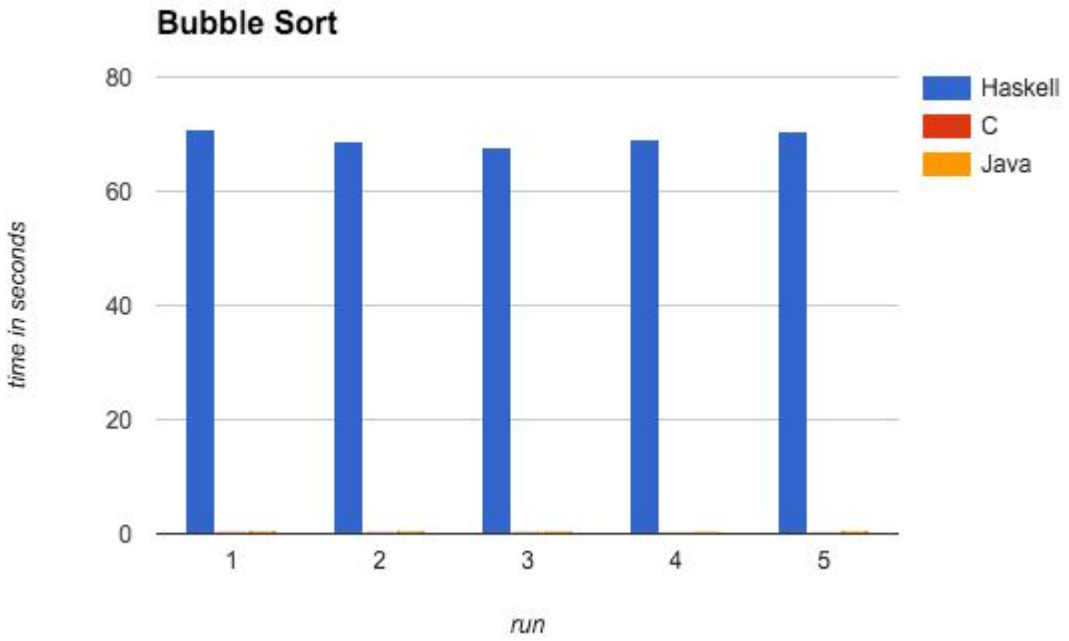
Slowest algorithm in the group. Drastic performance drop for Haskell, due to ridiculous amounts of recursion.

Average Runtimes:

Haskell: 69.388 seconds
C: 0.3106 seconds
Java: 0.4778 seconds

Run	Haskell	C	Java
1	70.88s	0.338s	0.494s
2	68.76s	0.325s	0.504s
3	67.68s	0.331s	0.485s
4	69.23s	0.282s	0.411s
5	70.39s	0.277s	0.495s

O(n^2)



Bubblesort, looking at the code:

- Haskell got it in 7 lines
- got it in 19 lines in Java (3 functions too!).
- got it in 18 lines in C (in 2 functions).

Well, in terms of Haskell this one was a disaster. It took over a minute every run. It was also the only algorithm to push C to over 1ms. It's an interesting algorithm, just not very good.

File Size

Haskell: 309 bytes
C: 940 bytes
Java: 1,395 bytes

Final Reflections

C is really really fast and efficient, but often a pain to code in, leading to frustration.

Java is really consistent and very straight forward and the code felt the most “clean”, and easy to read.

Haskell can code massive complex code structures in very few lines. A completely different way of coding, but often can be much less efficient.