Sovial Sonzeu

Professor Tsintsadze

CS 420

30 April 2023

Lab 5

Lab 5 programs consisted of merge sort and quick sort algorithms based on a random list of 1000 integers. Both programs should be timed and output the amount of time elapsed in seconds. For this lab, I wasn't able to download Haskell on my computer so I had to use an online compiler, here is the link to the code: https://onecompiler.com/haskell/3z75k5g6a. In this paper, I will be answering the questions mentioned in the lab 5 prompt on Canvas. Here is a link to the youtube video for code demo - https://www.youtube.com/watch?v=toVpFTxgX88&ab_channel=Sovi.

Code Readability, Modularity, and Maintainability

Haskell is a functional coding language. It has very unique ways of implementing several behaviors and functionalities such as generating random integers and doing list comprehensions similar to the Python programming language. It is somewhat readable, but not as readable compared to the imperative languages that are more commonly used today such as C++ and Python. May include some bias, but I think most people would probably prefer imperative languages since it is very intuitive in terms of the syntax and how variables and parameters are passed and declared. Code modularity is somewhat similar to Python and Haskell since both hold a set of instructions through function calls. With the function made, calling the function allows the user to use such instructions for specific purposes for outputting a specific result in a calculation or even printing results of calculation. In terms of code maintainability, I would say

both Haskell and Python offered good maintainability. I saw it was not too difficult to make changes when there were some errors in my code, thus I would say both are rather similar.

Quick Sort vs Merge Sort

The results between merge sort and quick sort showed that merge sort appeared to be faster on both Python and Haskell. This is expected since the algorithm's worst time complexity is O(nlogn). Quick sort was definitely slower on both the functional programming language and the imperative programming language. The algorithms were definitely faster on the sorted array compared to the random numbers array and reverse sorted order array. In the Python version though, the reversed order quick sort implementation was slower on the reversed sorted order array compared to the random integer input array. This was very surprising since this was the only case and I did not expect this to occur in only Python and not in Haskell.

Haskell vs Python

Some of the similarities between the two languages would be the ways of list slicing. Creating lists through slicing and list comprehensions is very similar to Python as it is in Haskell. In the Haskell code, I use similar syntax in making the sortedNumbers array along with the reverseSortedNumbers array as well. In addition, some of the functions are very similar such as used the built in reverse function in Python and Haskell. Some programming languages do not have such built-in functions but I am surprised Haskell does. Moving on, Python has a dynamic type system for variables and their types meaning they can change amongst runtime. On the other hand, Haskell is a strong, static, and expressive type programming language where variable types cannot change. In addition, Haskell uses lazy evaluation which means that expressions are only evaluated when their values are required. On the other hand with Python,

it uses eager evaluation which means expressions are evaluated immediately when met during runtime.

Outputs

Python

```
Testing quick sort on random array of 1000 integers:
Time elapsed: 0.0010006427764892578 seconds


Testing quick sort on ordered array of 1000 integers:
Time elapsed: 0.0005004405975341797 seconds


Testing quick sort on reversed array of 1000 integers:
Time elapsed: 0.001501321792602539 seconds


Testing merge sort on random array of 1000 integers:
Time elapsed: 0.0020017623901367188 seconds


Testing merge sort on ordered array of 1000 integers:
Time elapsed: 0.0010001659393310547 seconds


Testing merge sort on reversed array of 1000 integers:
Time elapsed: 0.0010011196136474 61 seconds
```

Haskell

```
Quick sort on randomNumbers:
Time elapsed: 2.265e-6 seconds


Quick sort on sortedNumbers:
Time elapsed: 1.57e-6 seconds


Quick sort on reverseSortedNumbers:
Time elapsed: 1.59e-6 second


Merge sort randomNumbers:
Time elapsed: 1.49e-6 seconds


Merge sort on sortedNumbers:
Time elapsed: 1.248e-6 seconds


Merge sort on reverseSortedNumbers:
Time taken: 1.52e-6 seconds
```