Approval Voting in Different Languages

Daven Chohan
301401324
dca120@sfu.ca
Comparative Programming Languages
CMPT 383 D100
Fall 2021

Performance

I ran an experiment of how efficient each program is by timing how long each program takes to analyze 10 000 ballots, 100 000 ballots and 1 000 000 ballots. I started each timer after I enter the file name. Racket was tested using a built-in function called current-inexact-milliseconds, this returned the amount of time the program takes to go from the starting point which I assigned to the ending which I also assign. Haskell was timed using :set +s which is something entered into terminal that calculates how long it takes to run a function. However, Haskell was run on a Linux subsystem on Windows 11 which may have affected efficiency. Finally, Python was tested by importing the time module, as this was an easy and efficient way to time how long a program takes. All timing is in seconds.

Ballots	10 000			100 000			1 000 000		
Python (sec)	0.08	0.08	0.08	0.40	0.41	0.41	3.60	3.61	3.54
Racket (sec)	0.12	0.12	0.12	1.27	1.42	1.31	15.12	16.13	14.86
Haskell (sec)	0.19	0.17	0.18	1.88	1.83	1.86	18.17	17.08	18.33

From the data above it is clear that Python performs far more efficiently than Racket or Haskell.

Ease of development

Python

The first language I had used to develop a program for approval voting was Python. Python is a popular mainstream language as it is taught as a beginner language to many new coders. Python's simplicity is what intrigues new users and its why I chose the language to implement approval voting. Unlike Racket or Haskell, my Python implementation used no functions. Python's file reading and for loop efficiency allowed me to traverse through each ballot with ease. I first used a for loop to traverse through each ballot, finding empty ballots, counting total ballots, appending all candidates from a duplicate free ballot to a list and then appending unique candidates to another list. This way I was able to have a list of all votes, a list of all potential candidates to vote for, a count of empty votes and a count of ballots. I then counted the amount of each candidate in the list of all votes and assigned the candidate's name and corresponding number of votes in a dictionary. To get full votes I checked the length of each ballot after removing duplicates with the length of a full vote and incremented a variable for each ballot of equal length. All of this can be done without the need of functions because of Python's useful built-in functions.

Developing a program for approval voting in Python took me about 2.5 hours. It took me another hour for debugging as I was having issues with my dictionary that stores each vote count, but once I figured out my issue it was fairly frustration free. Finally, I spent about 20 minutes testing as I wanted to test my program exhaustively which meant I had to create new test cases. I enjoyed coding in Python because of my familiarity with the language. I had knowledge in reading files with Python prior to this project which helped me tremendously. The simplicity of the language also made things easier as I was able to create straight forward solutions to tasks I need completed.

Racket

The second language I developed an approval voting program in was Racket. Having only written a few programs in Racket prior to this one, I was not as knowledgeable as I was of Python. Unlike with Python I used functional programming in my Racket implementation. I found it easier to use functions to remove duplicate votes in a ballot or find/remove empty ballots. Trying to reuse my Python implementation I initially used a for loop to remove duplicates and empty votes from each ballot in the file. Then just like Python I traversed through the file once more to find full ballots. At first this worked without issues, but as soon as I went to test a file with 100 000 ballots, I noticed a detrimental flaw. The for loop I was using required each ballot to be modified before appending it to a list and then setting the list equal to the appended version. This constant setting of the list made it traverse through each element of the list each time it was being set, which meant it was running at n^n time. For a file with 10 ballots this was not an issue but after 100 000 ballots took a few minutes and 1 000 000 ballots never finished I knew I had to make a change. By reviewing my knowledge of Racket, I realized that instead of a for loop to make changes to each element in the file, I could simply use map or filter. I filtered out empty lists and mapped my remove duplicates function to the list of all ballots in the file. This performed the task much quicker and after hours of debugging I was able to complete my Racket implementation.

Like my Python implementation my racket implementation only took about 2.5 hours. But this was when I was using for loops, during testing when I realized this was inefficient and had to debug it took me upwards of 6 hours to make my program be able to test 1 000 000 ballots at a reasonable time. At first, I found Racket to be unenjoyable, the frustration of my buggy software was having me pull my hair out. However, after I realized Racket's potential and correctly made use of map/filter, I found it very interesting. If I only had thought of using them at the beginning, I would have enjoyed it a lot more. I'm glad I am more familiar with the language after completing a challenging task like approval voting.

Haskell

The final language I used to implement approval voting was Haskell. I already found Haskell a little less enjoyable than Racket so after my experience with Racket, I was not looking forward to Haskell's implementation. Just like Racket I used functional programming here. I began with following the implementation from Professor Toby Donaldson, I found it very useful and quite like my implementation in Racket and Python. After that I started researching on how to read files and read user input, this was straight forward. After completing the project without much issue using inspiration from Toby's code and my own Racket code, I realized during the debugging stage that my approval voting does not work with candidate names longer than a single character. This annoyance caused me to have to rework much of the way I manipulate the lines read from the file while still trying to use the same algorithm as my previous 2 implementations. This took a lot longer than expected.

My Haskell program for approval voting took me about 3 hours. This was because much of it was taught already by Toby and by making use of his teachings with credit, I was able to complete the task relatively easily. However, during the debugging stage when I realized my program fails for candidates that are longer than one character, I had to rework much of my code. This debugging/testing process took me another 3 hours. I found Haskell more enjoyable than Racket as bugs were often easily fixable because of the brilliant explanations provided when attempting to compile broken code. Although Haskell was nowhere as simple as Python, I did not find it incredibly frustrating building my approval voting program, instead enjoyable as I

was learning many new things. To be fair part of my enjoyment was because of having a reference of approval voting from Toby, even though I did have to make changes to it.

	Python	Racket	Haskell
Development (hrs)	2.5	2.5	3
Debugging/Testing (hrs)	1.33	6	3
Enjoyability (1-5)	4	2	3

Quality of Source Code

I consider a line of code to be a line that is standalone from the previous or next line, it does something without needing the next or previous line to rely on. If I separated a line to 2 lines because the line was too long, this still counts as 1 line. Although if statements can be done on 1 line, I still classify them as multiple as it hurts readability to force them to be one. Blank lines and comments are not lines of code. I consider Python to have the worst readability because of it not being done using functional programming. Although it was easy to implement without functions it makes it terribly difficult to understand. Not only that but the precise indentation in Python makes it more difficult to read as it often makes me misunderstand how a certain piece of code works. Following Python in readability is Racket. Although Racket uses functional programming it suffers from the large number of brackets used. Having a ton of brackets everywhere just clutters the code causing it to be challenging to read even with functions. However, the use of functions makes code more concise and with adequate function names easier to read. Finally, Haskell has the best readability. This is because although Haskell is complex, its complexity allowed me to create concise code and complete tasks in fewer lines than the previous languages. Furthermore, Haskell uses functional programming but does not make use of many brackets unlike Racket, thus allowing it to be easier read.

	Python	Racket	Haskell
# of Lines of Code	46	64	53
# of named Functions	0	12	13
Readability (1-5)	2	3	4

Recommendation Section

In conclusion, Python is the language that should be chosen to implement an approval voting program. Python uses less lines of code than the other 2 languages but suffers from poor readability because of the lack of functions used. Although it may be difficult to read, its efficiency over Haskell and Racket implementation is staggering. Python is 4-5x faster than Racket or Haskell when it comes to 1 000 000 ballots, this difference will only grow exponentially as the ballot size increases. Moreover, I would recommend Python because of its ease of development, of the 3 languages it was far easier and took almost half the time. Racket and Haskell are not bad languages by any means, but they can be more challenging for a beginner. Although I have experience with Python, its simplicity and familiarity to other languages makes it easier to learn. Overall, Python's performance, ease of development and quality of source code makes it the best language to write a approval voting program.