Assignment 3 – Divit Koradia and Ethan Griffee

Introduction – This report is meant to outline the pros and cons of the 6 selected assignments, using various tests to back up various assertions and conclude which one of the assignments would be the best data adaptor to use moving forward. The 6 teams we selected are; SnowySong, Anonymous, Purgatory, TeamBaobab, Prizes and Chunky Boys.

Originally team OHE was going to be used, but our edge case test produced a Segfault.

Description of the analysis performed – We decided to test the 6 assignments with two different SoR files:

- 100row.sor: a relatively small file containing around 100 rows of data.
- default.sor: a large file containing more than 100,000 rows of data.

All the assignments were tested using these files to run two different functions; print_col_type and print_col_idx. For the sake of accuracy, we ran the test 3 times for each of the function and then plotted their averages in the graphs below.

Comparison of the products' relative performance

Ordered list in terms of preference:

1. SnowySong
2. Anonymous
3. Chunk Boys
4. Purgatory
5. TeamBaobab
6. Prizes

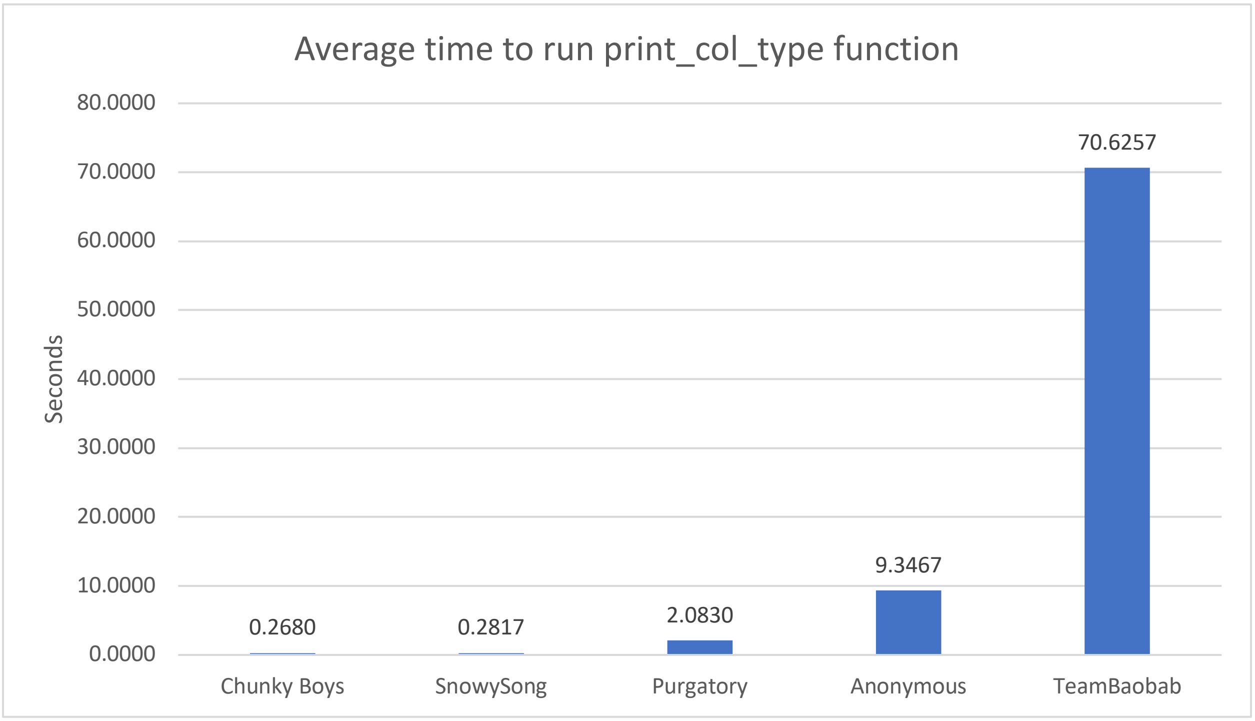

Fig 1. Average time to run the function print_col_type on default.sor file.

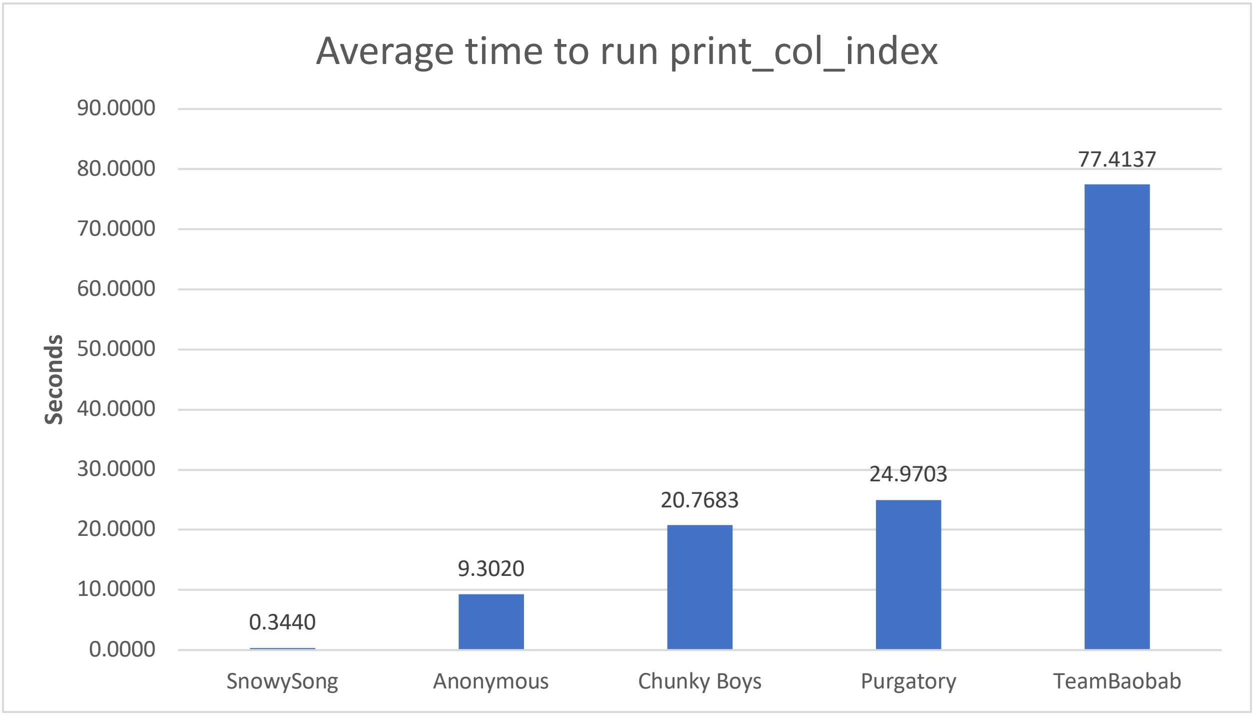

Fig 2. Average time to run the function print_col_index on default.sor file.

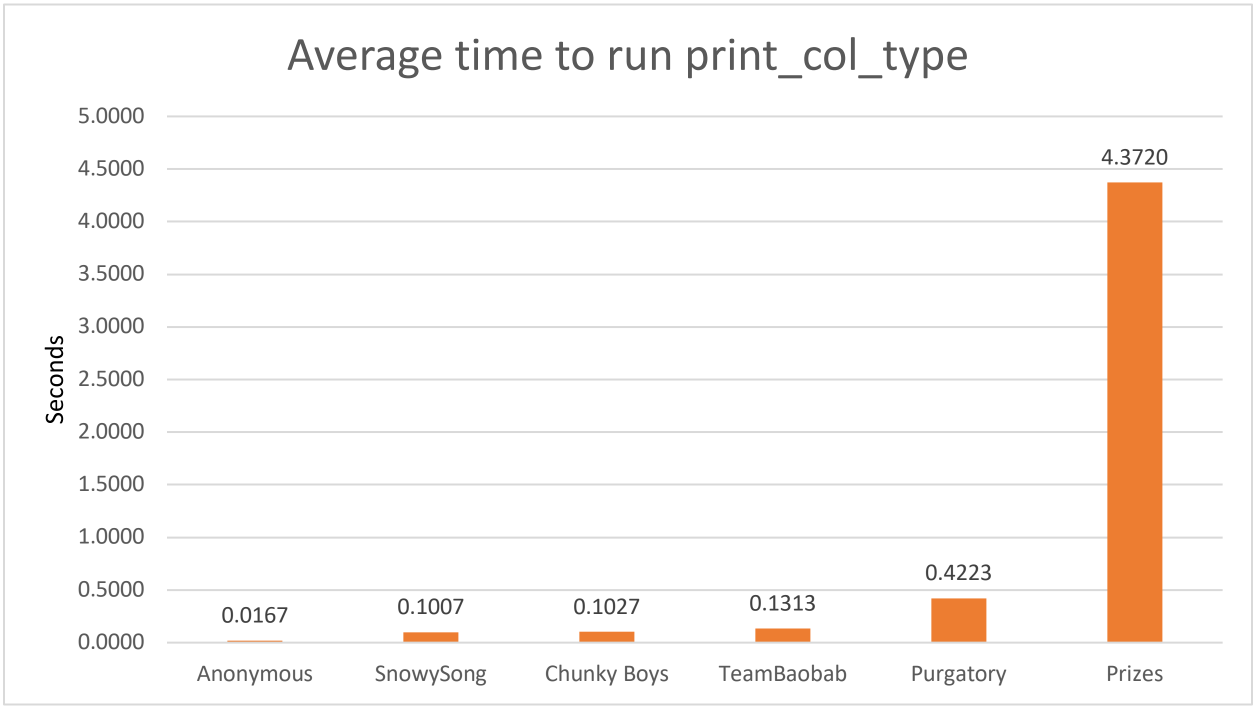

Fig 3. Average time to run the function print_col_type on 100row.sor file.

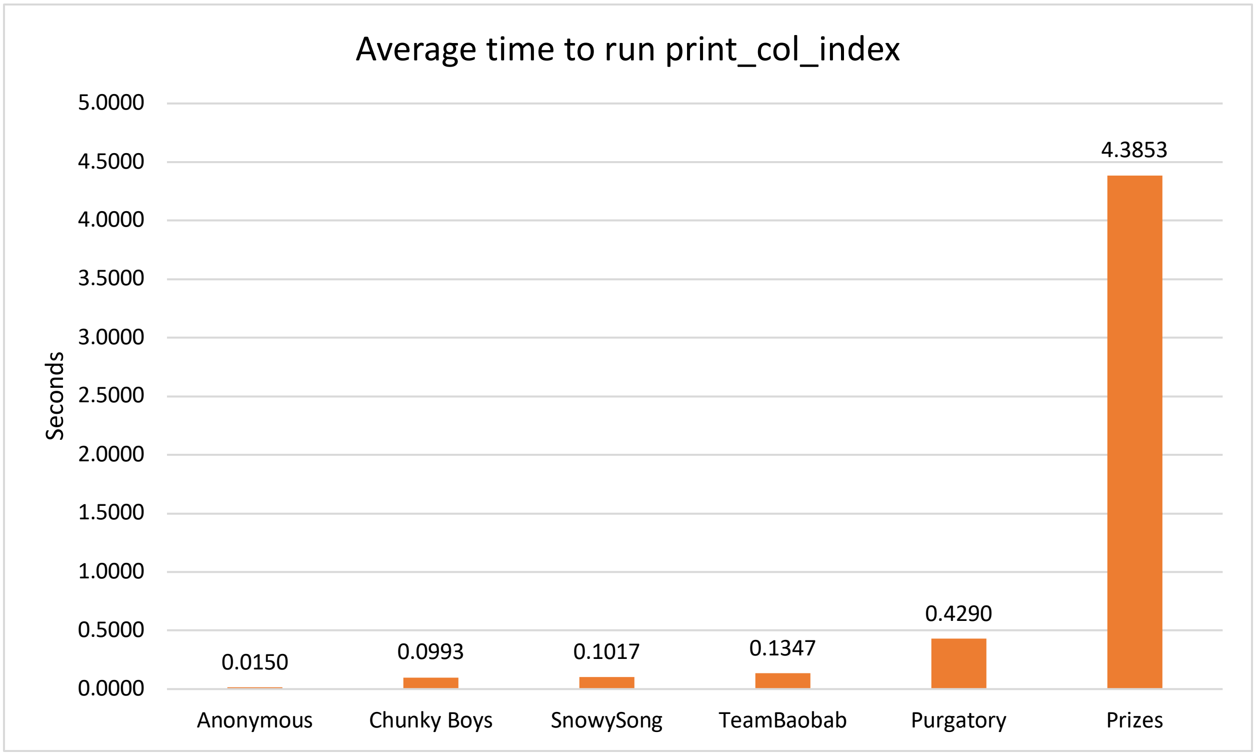

Fig 4. Average time to run the function print_col_index on 100row.sor file.

Disclaimer: In the first two graphs we decided to not include team Prizes because their code was taking upwards of 20 mins to run on our large default.sor file.

Comparison of the products' interpretation of edge case

One test we ran included the string <"dsfsaD><Dsafsa">. The behavior of this was not explicitly defined in the Project guidelines, but we were interested to see how each project handled it.

TeamBaobab, Prizes - Parses into two Strings: dsfsa, Dsafsa"

This does not make much sense because it removes the beginning " and the D at the end of the first string, when it should be checking if the D is a "

Anonymous - Parses into two empty strings:

This means that the code probably sees it as two strings "dsfsaD and Dsafsa." But, they decide it's "malformed," and makes it an empty string. This behavior does not make sense as they are throwing out data, and was not specified in the Assignment.

Chunky Boys - Parses into two strings: "dsfsaD, "Dsafsa""

This does not make much sense as the first string is not given quotes, while the second string is.

Snowy Song - Parses into two strings: ""dsfsaD", "Dsafsa""

This makes sense as it prints them like strings, adding quotations around it.

Purgatory - Parses into two strings: "dsfsaD, Dsafsa"

This makes sense as they are not changed, and interpreted as strings.

We believe that printing “dsfsaD><Dsafsa” would have made the most sense, however none of the products tested resulted in this option.

Threats to validity The tests were performed on a Mac OS with the charger plugged in and no other processes running. This would lead to the tests to be biased toward programs that perform better on macs. Also, our .sor files tested included the interesting edge cases many times. If the program was optimized to take longer on this unusual, unlikely input, and faster on more expected inputs, then it would be penalized harshly in our tests.

Comparison of Parsing

One major difference between different programs was whether the program parses the whole file and places it into columnar format, or if it just skips to the line given after the first 500 lines are read. This is a major difference that could cause certain programs to take much shorter, and give the correct answer, while not actually having the data we need.

Team Name	Parses Every Row
SnowySong	NO
Anonymous	YES
Chunk Boys	NO
Purgatory	To the row specfiied
TeamBaobab	YES
Prizes	YES

Purgatory does not skip rows but it does not parse rows after the row specified. This basically means that the time results for the 100k file for snowy song and chunky boys should be basically be thrown out as they don't do the required work. Purgatory's should be doubled as they do around half the work necessary.

Recommendation to management

1. Purgatory – While it wasn't the fastest to run, it was determined to be the fastest that handled the edge case well. Anonymous was a close runner up due to its very quick speeds while parsing every row. However, their handling of what we considered a valid input made us question whether the speed up was due to it being too itchy to throw out data.

Also, their code was the hardest to understand due to the odd names of variables and files (ie. helper1.h, helper2.h) and the decision to use a hashmap to store keys of row indexes. Concerns with purgatory include them having to move the parsing, and not end the parsing with the input row. However they store all the values in a 2d array, so with these changes it would be easy for them to send the whole array, or columns of it. Team Baobab's much slower performance turned us off of putting it higher.

2. Anonymous
3. Team Baobab
4. Snowy Song
5. Chunky bois
6. Prizes