Introduction

The goal of this report is to ends-based study of performance and code quality that will allow us to select a data adapter for our project. We accomplish this goal by analyzing the run time and memory usage of six data adapters by feeding the data adapters different .sor files. The differences in run time and memory usage as well as robusticity of the code to handle different cases was assessed in order to make our final decision.

Analysis Performed

The analysis was performed on a 2016 MacBook Pro with a 2.7GHz quad core Intel i7 and 16GB of 2133MHz DDR3 RAM. We used the compiler provided by the data adapter teams due to the assumption that the teams had optimized the compilation of their programs. However, two of the groups had Python 3 files which were already compiled and executables. In these cases, we used the provided sorer executable.

We tested the performance of all groups and recorded run times and memory usage through a Homebrew package, gnu-time. We used the command:

```
gtime --verbose ./sorer - f example.sor
```

to record maximum resident set size (memory), user time, system time, and elapsed (wall clock) time (total time). For all groups that provided a way to compile the program, we compiled it and ran against our .sor files. We repeated this process three times, cleaning the compiled executable between each process. As we mentioned above, two groups only provided an executable and not a compiler. Therefore, for these groups, we could only test by running the program against our .sor files three times without recompiling in between.

In order to test the robusticity of the program, we ran the programs with six different .sor files. To summarize, we had short and long files as well as misformed files where the rows have a different number of columns. The short files are 400-440 bytes and the long files were roughly 30-35MB. We chose to create different “shapes” with our files since the number of columns versus the number of rows may affect run time or memory usage despite the files being the same size. Thus, for short.sor, long_square.sor, misformed_small.sor, and misformed_large.sor, the shape of the file is roughly square in terms of number of rows versus size per row (i.e. 6 rows with each row being roughly 6MB). For long_hrect.sor, the shape of the file is a horizontal rectangle with only 3 rows but each row was roughly 10MB. For long_vrect.sor, the shape of the file is a vertical rectangle with 2,223,738 rows but each row is 2 columns or less. We have included these .sor files with this MEMO.

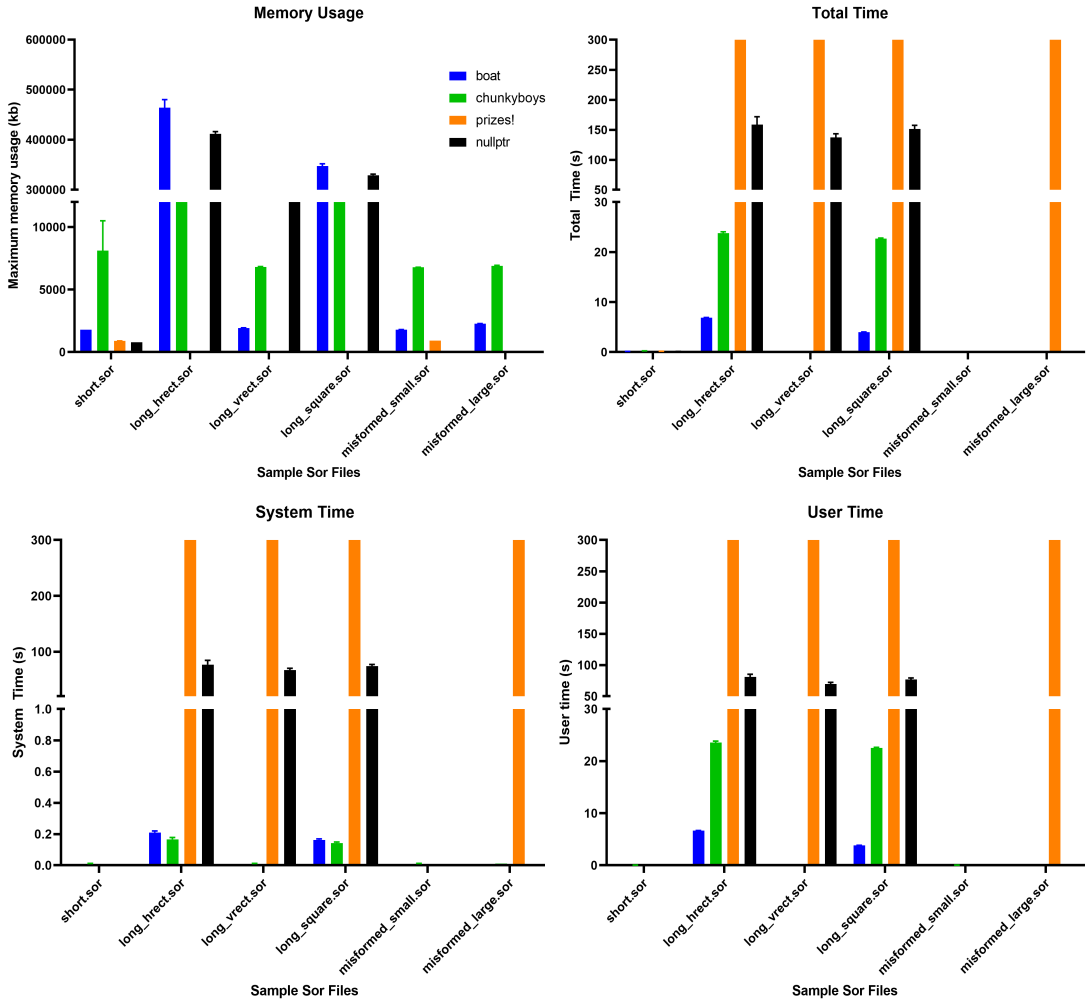
Comparison of Products Relative Performance

From best to worst for performance, our rankings are:

1. boat
2. chunky_boys
3. Prizes!/nullptr

We excluded two groups, Purgatory and OHE, due to errors during run time. These errors will be discussed in more detail in recommendation to management section. Prizes!/nullptr were also ranked equally due to different types of issues we encountered which will be discussed below.

In terms of run time (user time, system time, total time), boat was repeatedly at the top of the list for every .sor file we tested. We can note that between memory usage boat and chunky_boats, chunky_boys was more efficient in some cases (long_hrect.sor and long_square.sor), but boat had better memory usage for the other files (short.sor, long_vrect.sor, and the misformed .sor files). However, we felt that a quick run time was more important than memory differences between the two solutions.



In the Memory Usage graph, we did not include the memory usage for teams (nullptr and Prizes!) which were unable to parse the file given either due to failure to catch edge-cases or timed out. In the graphs regarding run time, we included teams (Prizes!) that timed out as 300 seconds (5 minutes), but teams (nullptr) that failed to process a file are recorded as 0 seconds. It is important to note, some teams' run times were quick enough that we recorded 0 seconds. Please refer to the tables at the end of this if necessary.

Interestingly, for both of the top two data adapters (boat and chunky_boys) as well as nullptr's data adapter, these solutions took longer to process the long_hrect.sor file

than the long_vrect.sor file. As suspected, the long_square.sor file tended to take a median amount of time compared to the other long files. From this, we inferred the run-time of these teams' solutions was more dependent on the number of columns rather than the number of rows.

As mentioned above nullptr and Prizes! were ranked equally. Although nullptr was able to run the long files in under three minutes, they were unable to handle misformed schemas or schemas containing edge-cases such as "<<>" which were specified in the specs. However, Prizes!'s solution was unable to process any of the long files in less than five minutes. Both teams performed equally well on the short files, and their performances were comparable to the top two groups.

To mitigate sources of non-determinism, we recompiled (if possible) after each run. Each run consisted of running all six .sor files once. We repeated the process of compiling and running three times. We were unable to necessarily standardize testing iterations; however, we did pass in different files which would theoretically result in different number of calls to different functions.

Threats to Validity

We ran on a local machine rather than a standard Docker image thus the performance results are dependent on the machine's hardware and software. Another issue is that we used each teams' given compiler rather than compiling the program ourselves to standardize the optimization the teams chose to use.

Recommendation to Management

From our results, we found that boat's data adapter is most preferable in terms of run time and memory usage as well as robusticity of code. Our order of preference is:

1. boat
2. chunky_boys
3. nullptr/Prizes!
4. nullptr/Prizes!
5. Purgatory
6. OHE

Over all, we favored a more efficient run time versus memory usage as we believe that as long as there are not major memory leaks (which we should have caught/noticed via testing with the different long .sor files), the memory usage differences were not major between the two data adapters.

As mentioned above, we had excluded Purgatory and OHE from our performance analyses due to runtime errors. Purgatory's program did not follow the spec and was missing cases such as "<';;;;;;'>" which should be processed as a string. When running a .sor file containing this case, we got a traceback error stating that the program had tried to process the string as an int. When running OHE, we had an error on all long files stating, command terminated by signal 11, which is a memory issue. Additionally,

after compiling OHE using their provided Makefile, there were 9 compiler warnings. We chose to rank Purgatory's program above OHE's as their error was due to a missed case rather than a memory leak or memory issue.

We noted that the top two data adapters both used Python 3. The third and fourth place programs used C++. Purgatory used Python 3 and OHE used C++. However, we considered the language secondary to performance when making our decision.

In conclusion, **we recommend using the data adapter from the boat team** as it had no major memory usage issues and was able to process files of different sizes as well as mis-formed schemas the most efficiently.

Data in Table Format

User Time (s)

Team Name	short	long_hrect	long_vrect	long_square	misformed_small	misformed_large
boat 1	0	6.7	0	3.85	0	0
boat 2	0	6.66	0	3.84	0	0
boat 3	0	6.63	0	3.87	0	0
purgatory 1	0.01	9.58	-	-	-	-
purgatory 2	0.01	-	-	-	-	-
purgatory 3	0.01	-	-	-	-	-
chunkyboys 1	0.04	23.36	0.03	22.66	0.02	0.05
chunkyboys 2	0.02	23.44	0.03	22.49	0.03	0.05
chunkyboys 3	0.02	23.89	0.03	22.52	0.03	0.05
OHE 1	0.02	-	-	-	-	-
OHE 2	-	-	-	-	-	-
OHE 3	-	-	-	-	-	-
prizes! 1	0	300	300	300	0	300
prizes! 2	0	300	300	300	0	300
prizes! 3	0	300	300	300	0	300
nullptr 1	0	78.07	67.82	74.8	-	-
nullptr 2	0	78.76	72.91	79.87	-	-
nullptr 3	0	85.87	68.1	75.76	-	-

System Time (s)

Team Name	short	long_hrect	long_vrect	long_square	misformed_small	misformed_large
boat 1	0	0.22	0	0.16	0	0
boat 2	0	0.21	0	0.17	0	0
boat 3	0	0.2	0	0.16	0	0
purgatory 1	0.04	0.43	-	-	-	-
purgatory 2	0.02	-	-	-	-	-
purgatory 3	0.02	-	-	-	-	-
chunkyboys 1	0.01	0.16	0.01	0.14	0	0.01
chunkyboys 2	0.01	0.16	0.01	0.14	0.01	0.01
chunkyboys 3	0	0.18	0	0.15	0.01	0.01
OHE 1	0.02	-	-	-	-	-
OHE 2	-	-	-	-	-	-
OHE 3	-	-	-	-	-	-
prizes! 1	0	300	300	300	0	300
prizes! 2	0	300	300	300	0	300
prizes! 3	0	300	300	300	0	300
nullptr 1	0	73.05	65.2	71.34	-	-
nullptr 2	0	71.8	70.69	77.2	-	-
nullptr 3	0	85.51	65.64	73.98	-	-

Elapsed wall clock time (s)

Team Name	short	long_hrect	long_vrect	long_square	misformed_small	misformed_large
boat 1	0.08	6.93	0.01	4.02	0.01	0
boat 2	0.16	6.88	0	4.02	0	0
boat 3	0	6.84	0	4.04	0	0
purgatory 1	0.26	10.03	-	-	-	-
purgatory 2	0.03	-	-	-	-	-
purgatory 3	0.03	-	-	-	-	-
chunkyboys 1	0.27	23.54	0.05	22.82	0.04	0.07
chunkyboys 2	0.04	23.77	0.05	22.65	0.04	0.06
chunkyboys 3	0.03	24.1	0.04	22.69	0.04	0.06
OHE 1	0.19	-	-	-	-	-

OHE 2	-	-	-	-	-	-
OHE 3	-	-	-	-	-	-
prizes! 1	0.11	300	300	300	0	300
prizes! 2	0.17	300	300	300	0	300
prizes! 3	0.11	300	300	300	0	300
nullptr 1	0.08	151.21	133.07	146.23	-	-
nullptr 2	0.13	150.62	144.32	158.14	-	-
nullptr 3	0.15	173.98	134.84	149.8	-	-

Memory (KB)

Team Name	short	long_hrect	long_vrect	long_square	misformed_small	misformed_large
boat 1	1788	481380	1896	345304	1800	2284
boat 2	1788	461304	1956	343652	1788	2260
boat 3	1788	450080	1880	352652	1788	2260
purgatory 1	4668	244632	-	-	-	-
purgatory 2	4108	-	-	-	-	-
purgatory 3	4144	-	-	-	-	-
chunkyboys 1	10876	172072	6840	91792	6756	6868
chunkyboys 2	6772	176060	6768	91484	6780	6944
chunkyboys 3	6680	174476	6772	101624	6780	6824
OHE 1	102332	-	-	-	-	-
OHE 2	-	-	-	-	-	-
OHE 3	-	-	-	-	-	-
prizes! 1	900	-	-	-	912	-
prizes! 2	884	-	-	-	912	-
prizes! 3	884	-	-	-	912	-
nullptr 1	788	413980	290256	328468	-	-
nullptr 2	788	414836	290244	331480	-	-
nullptr 3	788	406916	290244	326992	-	-

% CPU usage

Team Name	short	long_hrect	long_vrect	long_square	misformed_small	misformed_large
boat 1	2	99	36	99	20	71
boat 2	1	99	66	99	50	71
boat 3	50	99	50	99	66	71
purgatory 1	23	99	-	-	-	-
purgatory 2	94	-	-	-	-	-
purgatory 3	94	-	-	-	-	-
chunkyboys 1	23	99	90	99	87	92
chunkyboys 2	90	99	78	99	88	92
chunkyboys 3	87	99	88	99	88	92
OHE 1	26	-	-	-	-	-
OHE 2	-	-	-	-	-	-
OHE 3	-	-	-	-	-	-
prizes! 1	6	99	99	99	71	99
prizes! 2	4	99	99	99	83	99
prizes! 3	5	99	99	99	71	99
nullptr 1	4	99	99	99	-	-
nullptr 2	2	99	99	99	-	-
nullptr 3	2	98	99	99	-	-