Introduction:

In this assessment we were tasked with the implementation of spreadsheets using 3 different abstract data types, Arrays, Linked Lists and Compressed sparse rows. Each spreadsheet consists of the same functions used to modify and access the spreadsheet, some functions worked very well with certain abstract data types and were more complex to implement with other abstract data types. Following the implementation of the spreadsheets is the analysis which will be covered in this report and provide recommendations for which spreadsheet is the best over all and if picking one over the other has any benefit.

Implementation:

Array Spreadsheet:

The array spreadsheet was the easiest to implement out of the three abstract data types as python comes with its own python lists which were used to implement this. Using the python lists we made a 2 dimensional array where the outer array represents the rows and the inner array represents the values stored in each column. Python lists come with an append function as well as an insert function which makes the implementation of those 2 features for the spreadsheet much easier, along with that it also has a function for getting the length.

Theoretical time complexity:

The array spreadsheet consists of multiple arrays containing further arrays within which suggests a time complexity of $O(n^2)$.

Linked List Spreadsheet:

The linked list was slightly more difficult then array spreadsheet as there were no linked list data structures that come with python. When implementing the linked list spreadsheet we began by adding a linked list class which would help contain the list nodes. Similar to the 2 dimensional array we decided to make each linked list its own node by adding the next and previous linked list and the head and tail of the linked lists to the spread sheet this made the logic very clear as it wasnt much different to a 2 dimensional array. The implementation of each function was not too difficult but the code included more checks and loops to iterate through the linked list as we cannot index them like we can with arrays.

Theoretical time complexity:

Similar to the array spreadsheet the linked list consists of multiple nodes and the linked list itself is a node and most of the functions for the linked list use an additional loop along with a nested loop which increases the time complexity for those functions to $O(n^2) + O(n)$.

CSR (Compressed sparse rows) spreadsheet:

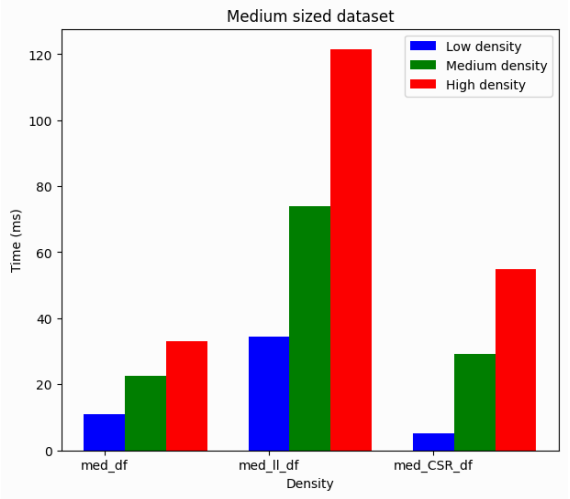
Out of the 3 abstract data types csr was the most difficult this was probably due to CSR spreadsheets only containing the non-none values making the rows and columns harder to keep track of as well as iterating through the spreadsheet. We found update the hardest function to implement as there were alot of things to consider like whether the desired cell has a value or if its a none value and once the value has been updated we must update the sum array as well. Because it was keeping track of the columns (more so than the rows) we decided to have a separate variable dedicated to keeping track of the columns because when appending a column, it was practically invisible to all the arrays.

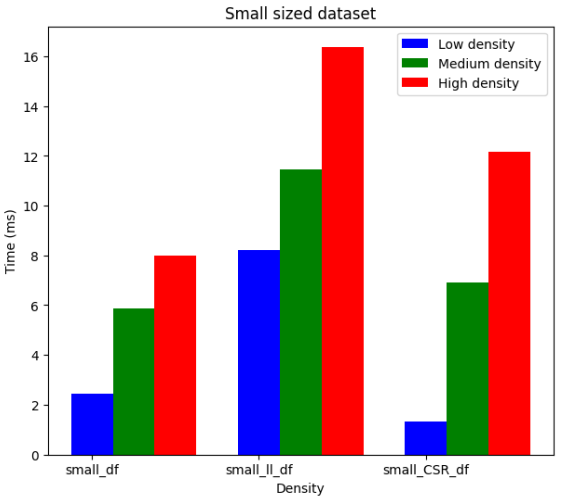
Theoretical time complexity:

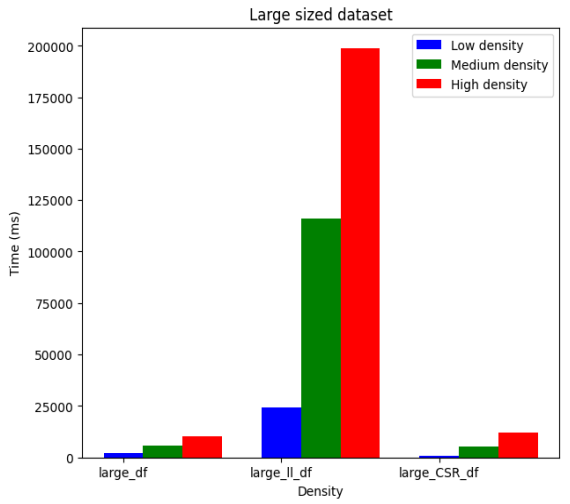
The CSR spreadsheet uses nested for loops of most of its functions which indicates that it has a time complexity of $O(n^2)$.

Data Generation & Analysis:

Data generation was done for different sizes which were then used for different densities. The sizes were 50x50, 500x500, and 1000x1000, denoted as small, medium and large respectively, and our densities were 0.1, 0.5 and 0.9, denoted as low, medium and high respectively. Using the generated data we tested each function in all 3 of the spreadsheets and used time as a measure of performance. This was all done on the same device so that the performance isn't affected. Using the measured time taken for each function the total was calculated and used to produce the following graphs







Each of the above graphs (going left to right, top down) shows different densities, different sizes and different implementations of the spreadsheets, the first graph shows small array based, linkedlist based, and CSR based spreadsheets with 0.1 density in blue, 0.5 density in green and 0.9 density in orange/red, then the second graph shows the medium sized spreadsheets, and the last one shows the large spreadsheets

After seeing the results of the graphs above a thought occurred to me, that since changing the array size requires the creation of a completely new array that maybe if more function calls were made then the linked list could have shown better performance as it only requires changing the values of the before and after nodes. But in the following table we can see that even though build spreadsheet is taking up most of the runtime, the insert column and insert row functions show that the array spreadsheet is still performing better with those functions compared to the table for the array spreadsheet insert row and insert column. The cause of this is the need to iterate through each preceding value in the linked list spreadsheet which makes the time taken longer compared to indexing.

	low_density	med_density	high_density
build_spreadsheet	22945.1656	114002.9863	196624.1558
append_col	2.0124	1.8332	1.8793
append_row	1.2131	1.4690	1.3595
insert_col	464.6435	565.9118	647.8697
insert_row	358.4450	475.8435	594.8127
col_num	0.2614	0.2825	0.2502
row_num	0.1328	0.3821	0.3481
entries	323.9194	455.5146	589.4819
update	0.0064	0.0085	0.0113
find	320.9875	431.5040	526.5185

The table above represents the runtime of each function for a linked list spreadsheet with a large data set for each density

	low_density	med_density	high_density
build_spreadsheet	966.0675	2287.0662	2820.5789
append_col	50.6758	192.3795	641.8793
append_row	0.8547	0.4791	1501.8637
insert_col	170.0171	262.8869	326.0630
insert_row	127.6908	225.9981	278.5943
col_num	0.0023	0.0030	0.0019
row_num	0.0008	0.0010	0.0006
entries	657.1338	2683.6053	4088.9804
update	0.0058	0.0042	0.0029
find	146.0155	258.6770	336.3159

The table above represents the runtime of each function for array spreadsheet with a large data set for each density

Observations/findings:

It was observed that linked lists took more time than both csr and array based spreadsheets and this increased at $O(n^2)$ as the data size and density got larger. In the graphs above you can see that with a small sized data set the performance was not too different but as the data set increased to medium sized the difference was much more evident and even more so when we used a large dataset. It was observed that each abstract data type increased in time taken when the dataset was more dense but the only difference was CSR spreadsheet which with less dense data performed better than the array spreadsheet. This performance difference is

due to the CSR spreadsheet not storing none values which decreases the amount of data scanned each run of the function.

Conclusion/recommendations:

Overall the array based spreadsheet was the best in both simplicity during implementation and overall performance and even though the CSR spreadsheet does perform better with less dense data the difference is very small so unless you are very certain that the bulk of the data is going to be none values i would still recommend using the array list. The linked list was out of the question from what was observed as it performs worse and worse as the dataset increases and even with smaller datasets its performance was poor.