A logo of a globe with rings around it

AI-generated content may be incorrect.code

GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT077-3-2-DSTR

DATA STRUCTURES

APU2F2409CS(CYB)

HAND OUT DATE: 31 – MARCH - 2025

HAND IN DATE: 05 – MAY - 2025

WEIGHTAGE: 30%

**GROUP 21**

|  |  |
| --- | --- |
| **Name** | **TP Number** |
| **FOO JING SZE** | **TP070845** |
| **CHLOE TAN JIA XIN** | **TP070759** |
| **GOH YUAN KEE** | **TP070126** |
| **NG ZHE SHEN** | **TP071625** |

**LECTURER: TS. NICHOLAS TEH SEK KIT**

Table of Contents

[**1.0** **Workload Matrix Table with Signature** 5](#_Toc197356161)

[**2.0** **Theoretical Explanation** 6](#_Toc197356162)

[2.1 Structures used 6](#_Toc197356163)

[2.2 Data Manipulation 8](#_Toc197356164)

[2.2.1 Insert Data from CSV files 8](#_Toc197356165)

[2.2.2 Convert format 10](#_Toc197356166)

[2.2.3 Sort Data 11](#_Toc197356167)

[2.2.4 Display Results 13](#_Toc197356168)

[2.2.5 Filter Data 14](#_Toc197356169)

[2.2.6 Analyze Text 16](#_Toc197356170)

[2.2.7 Free Memory 19](#_Toc197356171)

[2.2.8 Calculate Time and Space Efficiency 20](#_Toc197356172)

[**3.0** **Input-Output Screenshots** 22](#_Toc197356173)

[3.1 Menu 22](#_Toc197356174)

[3.2 Choice 1 23](#_Toc197356175)

[3.3 Choice 2 24](#_Toc197356176)

[3.4 Choice 3 25](#_Toc197356177)

[3.5 Choice 0 26](#_Toc197356178)

[**4.0** **Summary Discussions** 27](#_Toc197356179)

[4.1 Analyze system efficiency 27](#_Toc197356180)

[4.2 Summary and observations made during development 29](#_Toc197356181)

[4.3 Strengths and weaknesses of the code 29](#_Toc197356182)

[**5.0** **Summarize and Personal thoughts on the assignment** 31](#_Toc197356183)

[5.1 Summarize 31](#_Toc197356184)

[5.2 Potential improvements and system weaknesses 31](#_Toc197356185)

[5.3 Personal thoughts on the assignment 32](#_Toc197356186)

[**6.0** **References** 33](#_Toc197356187)

z

Table of Figures

[Figure 2.1 Linked List structures 6](#_Toc197295850)

[Figure 2.2 Linked List additional structure 6](#_Toc197295851)

[Figure 2.3 Array structure 7](#_Toc197295852)

[Figure 2.4 Linked List insertReview 8](#_Toc197295853)

[Figure 2.5 Linked List insertTransaction 8](#_Toc197295854)

[Figure 2.6 Array insert CSV files 9](#_Toc197295855)

[Figure 2.7 Linked List sortTransactionsByDate 10](#_Toc197295856)

[Figure 2.8 Array sortTransactionsByDate 10](#_Toc197295857)

[Figure 2.9 Linked List displayElectronicsCreditCardTransaction 12](#_Toc197295858)

[Figure 2.10 Array displayElectronicsCreditCardTransactions 13](#_Toc197295859)

[Figure 2.11 Linked List analyzeNegativeReviews 14](#_Toc197295860)

[Figure 2.12 Array analyzeNegativeReviews 15](#_Toc197295861)

[Figure 2.13 Array analyzeNegativeReviews 16](#_Toc197295862)

[Figure 2.13 Linked List convertDataToInt 17](#_Toc197295863)

[Figure 2.14 Linked List cleanWord 17](#_Toc197295864)

[Figure 2.15 Array convertDateToInt 17](#_Toc197295865)

[Figure 2.16 Linked List displayTopTransactions 18](#_Toc197295866)

[Figure 2.17 Array displayTopTransactions 18](#_Toc197295867)

[Figure 2.18 Linked List free memory code 19](#_Toc197295868)

[Figure 2.19 Array free memory code 19](#_Toc197295869)

[Figure 2.20 Linked List calculateAndDisplayEfficiency 20](#_Toc197295870)

[Figure 2.21 Array calculate time and space efficiency 20](#_Toc197295871)

[Figure 3.1 Linked List Menu 22](#_Toc197295872)

[Figure 3.2 Array Menu 22](#_Toc197295873)

[Figure 3.3 Input Choice 1 23](#_Toc197295874)

[Figure 3.4 Linked List Choice 1 23](#_Toc197295875)

[Figure 3.5 Array Choice 1 23](#_Toc197295876)

[Figure 3.6 Input Choice 2 24](#_Toc197295877)

[Figure 3.7 Linked List Choice 2 24](#_Toc197295878)

[Figure 3.8 Array Choice 2 24](#_Toc197295879)

[Figure 3.9 Linked List Choice 2 Percentage 24](#_Toc197295880)

[Figure 3.10 Array Choice 2 Percentage 24](#_Toc197295881)

[Figure 3.11 Input Choice 3 25](#_Toc197295882)

[Figure 3.12 Linked List Choice 3 25](#_Toc197295883)

[Figure 3.13 Array Choice 3 25](#_Toc197295884)

[Figure 3.14 Input Choice 0 26](#_Toc197295885)

[Figure 3.15 Linked List Choice 0 26](#_Toc197295886)

[Figure 3.16 Array Choice 0 26](#_Toc197295887)

[Figure 4.1 Linked List System Efficiency Question 1 27](#_Toc197295888)

[Figure 4.2 Array System Efficiency Question 1 27](#_Toc197295889)

[Figure 4.3 Linked List System Efficiency Question 2 27](#_Toc197295890)

[Figure 4.4 Array System Efficiency Question 2 27](#_Toc197295891)

[Figure 4.5 Linked List System Efficiency Question 3 28](#_Toc197295892)

[Figure 4.6 Array System Efficiency Question 3 28](#_Toc197295893)

[Figure 4.7 Linked List Used Space 28](#_Toc197295894)

[Figure 4.8 Array Used Space 28](#_Toc197295895)

# **Workload Matrix Table with Signature**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tasks** | | **Name** | | | | **Total** |
| **Part** | **Component** | **Ng Zhe Shen**  **TP071625** | **Foo Jing Sze**  **TP070845** | **Goh Yuan Kee**  **TP070126** | **Chloe Tan Jia Xin**  **TP070759** |
| **Lab Work** | 1. Array Implementation |  |  | 50% | 50% | 100% |
| 1. Linked List Implementation | 50% | 50% |  |  | 100% |
| **Solution Work** | 1. Theoretical Explanation | 25% | 25% | 25% | 25% | 100% |
| 1. Input-Output Screenshots | 25% | 25% | 25% | 25% | 100% |
| 1. Summary Discussions |  | 50% | 50% |  | 100% |
| 1. Conclusion and Reflection | 50% |  |  | 50% | 100% |
| 1. References | 25% | 25% | 25% | 25% | 100% |
| **Signature** | |  | A black text on a white background  Description automatically generated |  |  |  |

# **Theoretical Explanation**

## Structures used

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer program  AI-generated content may be incorrect.  Figure 2.1 Linked List structures | The main structures used are **Review** and **Transaction**, which define linked lists for storing data from **reviews.csv** and **transactions.csv**. The structures are used to store review data and transaction data as nodes in a linked list, where each node represents a single review or transaction record. Both structures support dynamic data handling by linking nodes together for efficient insertion and traversal of data. |
| Additional Temporary Structure  A white background with blue text  AI-generated content may be incorrect.  Figure 2.2 Linked List additional structure | The wordcount structure is an additional temporary structure used within the **analyzeNegativeReviews** function to store the words extracted from 1-star reviews along with their frequency counts. Once the analysis is completed, the dynamically allocated memory will be deleted. |
| **Array** | Figure 2.3 Array structure | The main structure used is Review and Transaction. Each of the structure store fields that are imported from CSV, respectively, and not including the internal pointer, therefore real case can be stored continuously on a single dynamic buffer such as **std::vector**. Review and Transaction include **productID, customerID, reviewText,** and **rating** used to store original sequences. These structures implement stable time complexity for appends, rapid direct access, and superior memory locality by using continuous storage and precise control over size and capacity. |

## Data Manipulation

### Insert Data from CSV files

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer program  AI-generated content may be incorrect.  Figure 2.4 Linked List insertReview | The code shows the **insertReview** and **insertTransaction** functions, which handle the insertion of new nodes at the end of the linked list for reviews and transactions. Unlike arrays, which have a fixed or known in advance size of data structure, linked list’s data structure size can be frequently changing as the functions dynamically allocate the memory with a new node by using the data passed through the given CSV files and set the **next** pointer to **nullptr**, indicating the end of the linked list. If the linked list is empty (head pointer = nullptr), the new node will become the head of the linked list. Moreover, the functions will traverse to the end of the list to find the last node and update the next pointer to link with a new node, preserving the insertion order and ensuring the list is connected properly. The purpose of the functions is to support an efficient and organized storage sequence of data from the CSV files. |
| A screenshot of a computer code  AI-generated content may be incorrect.  Figure 2.5 Linked List insertTransaction |
| **Array** | Figure 2.6 Array insert CSV files | CSV interpretation opened the input file and directly discarded its title row. Subsequently, every line is read by a character string, and a comma is used to separate various fields. For example, customer ID, product, price, date, category, and payment method. Afterward, a strict verification system is implemented. A numeric field, such as price parsed by a helper program to ensure the validity of the floating-point format. For example, the date string complies with the expected "YYYY-MM-DD" format, and the text field that is necessary to fill up is empty or not. Any line that fails to pass through these checks, such as an invalid date, will be skipped to ensure data integrity. A successful verification item will be converted to the custom "transaction" or "review" structure and appended to the dynamic array. This approach gets the balance between simplicity and robustness, ensures only the clean and correct format data is processed on the sequence, filtering, and analysis programs during downstream processing. |

### Convert format

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A white background with black and red text  AI-generated content may be incorrect.  Figure 2.13 Linked List convertDataToInt | The code shows the **convertDateToInt** and **cleanWord** functions, which both functions are mainly used to convert to different formats for easier manipulation. The **convertDateToInt** functions convert the date format from **YYYY-MM-DD** format, extract the day, month, and year to **YYYYMMDD** format to allow numerical comparison, making it easier to sort the dates. Besides, the **cleanWord** function processes words by removing the non-alphabetic characters and converting the remaining letters to lowercase, ensuring that the text data is consistent for frequency analysis or sentiment analysis. |
| A screen shot of a computer code  AI-generated content may be incorrect.  Figure 2.14 Linked List cleanWord |
| **Array** | A computer code with colorful text  AI-generated content may be incorrect.  Figure 2.15 Array convertDateToInt | The **convertDateToInt** function accepts one format of string, which is "**DD/MM/YYYY**". By using **substr(0,2),** **substr(3,2),** and **substr(6,4)** to retrieve the day, month, and year, respectively. These operations will generate a new string object and retrieve the **std::stoi** to implement each string. Lastly, the integer **YYYYMMDD** has been generated. |

### Sort Data

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer code  AI-generated content may be incorrect.  Figure 2.7 Linked List sortTransactionsByDate | The code shows the **sortTransactionsByDate** function, which sorts a linked list of Transaction nodes by date in ascending order using an insertion sort algorithm. It first verifies whether the list is null or has one element, where there is no need to sort. Otherwise, it iterates through each of the nodes of the original list sequentially, removing the first node and attempting to insert it into the appropriate position in the sorted list. If the sorted list is empty or the date of the current node is before, it will be inserted at the front of the list. Otherwise, the operation traverses the sorted list to find where the current node fits based on its date and inserts it in place by adjusting pointers. This is repeated until every node is included in the sorted list and is updated from the head to point to the sorted list. |
| **Array** | Figure 2.8 Array sortTransactionsByDate | The sorting function is implemented, and the insertion sort algorithm is applied. The transaction records sequence is arranged based on the date field in ascending order. This algorithm iterates over every transaction starting from the second element of the dynamic array. Regarding each of the transactions, the program will temporarily store the current record as key and compare it with the previous date and record. If the previous date is more than the date of the key, then shift the previous record one space to free up spaces. This process continues until the correct location is found, then the key is inserted into the correct position. Due to the format of the date field used, "YYYY-MM-DD" format, thus compared lexicographically to ensure the correct time sequence. This algorithm is executed in place, and is considered effective for small or partially sorted datasets |

### Display Results

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer code  AI-generated content may be incorrect.  Figure 2.16 Linked List displayTopTransactions | The code shows the **displayTopTransactions** function, which displays the details of the top N transactions from the transaction linked list, sorted by date. It will display the header of the table first, which includes the transaction details column such as CustomerID, Product, Category, Price, Date, and Payment Method, which are formatted neatly in columns. The function then iterates through the linked list and displays the **topN** transactions, ensuring that the details are properly aligned using **setw** for column width and **setprecision** for formatting the price. The while loop will stop when the specified number of transactions is reached or when the list ends. This allows users to view a specified number of the most recent transactions clearly. |
| **Array** | A screen shot of a computer  AI-generated content may be incorrect.  Figure 2.17 Array displayTopTransactions | The code tidy prints the top N entries of a DynArray<Transaction>. First, it used std::setw to emit the title and structured header row. After that, a dashed line was drawn to separate. In the main loops, it incremental iteration from index0 to the topN-1. Each of the transactions shows fields in a fixed-width column. The price field shows two decimal places accurately and uses std::fixed and std::setprecision. |

### Filter Data

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer code  AI-generated content may be incorrect.  Figure 2.9 Linked List displayElectronicsCreditCardTransaction | The code shows the **displayElectronicsCreditCardTransaction** functions, which demonstrate the use of linked list traversal and filtering to perform category-based data analysis. The function first iterates through each node in the linked list, checking for the type of transaction under the “**Electronics**” category, and further filters the payment method to identify those that use “**Credit Card**” in the “**Electronics**” category. The counters will then keep track of the total number of users using “**Credit Card**” as the payment method in the “**Electronic**” category. The percentage will then be computed by the function to calculate the specific category over all users in two decimal places. |
| **Array** | Figure 2.10 Array displayElectronicsCreditCardTransactions | The **displayElectronicsCreditCardTransactions** function filters and displays transactions where the category is "**Electronics**" and the payment method is "**Credit Card**". It first prints a table header to show the output format. Then, it goes through each transaction in the array one by one. If a transaction belongs to the "**Electronics**" category, it increases a counter. If the same transaction was paid using a credit card, it also increases another counter and displays that transaction’s details. After checking all the transactions, the function calculates the percentage of electronics purchases that were made with a credit card and prints the result. If no electronic transactions are found at all, it displays a message to inform the user. |

### Analyze Text

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer program  AI-generated content may be incorrect.A screenshot of a computer program  AI-generated content may be incorrect.  Figure 2.11 Linked List analyzeNegativeReviews | The code shows the **analyzeNegativeReviews** function, which processes all 1-star reviews from the **Review** linked list of Review nodes to identify the most frequently mentioned words. The function will start to iterate through each review first, extract and clean the review text to remove the punctuation and convert words to lowercase using the **cleanWord** function. After that, the function will then check if the word is iterated through the review if it already exists in the word count list (wordlist); if it does, the count is incremented, otherwise a new entry will be created. Once all reviews have finished processing, the function will sort out the word list in descending order by frequency using an insertion sort algorithm. It then displays the top 10 most frequent words from 1-star reviews, along with their frequencies, and deletes the allocated memory for the word count list to prevent memory leaks. |
| **Array** | Figure 2.12 Array analyzeNegativeReviews    Figure 2.13 Array analyzeNegativeReviews | This code is used to analyze negative reviews, such as rating = 1, thus analyzing the frequency of the usage of words from reviews. It iterates collection of reviews, using a string stream to segment of review text. Each of the words needs to be cleaned, such as converting the characters to small letters to ensure consistency. The code used a dynamic array of WordCount structures to calculate the frequency of each word. If the word exists, then update the count, and if it is a new word, add count. After handling all the reviews, sequence the words based on frequency in descending order, and show the 10 highest frequency words. This approach can identify repeated terms of negative reviews and improve sentiment analysis or find the common questions.  The linear scan approach is to implement search and filter operations, which includes traversing the dataset or special analysis requirements for every element following the sequence. For example, the code will traverse every object of Review on the dynamic array in the **analyzeNegativeReviews** function, to check if the rating is equal to 1 or not. This function will process reviewText to extract and count words if it fulfills the requirement. This approach can ensure it checks every element, making it effective for unsorted datasets or datasets that follow specific criteria for analysis. The time complexity is 0(n), although this approach seems easy to implement and simple, where n is elements in the dataset, because each element must be evaluated individually. The complexity of the search operation is still 0(1), but might need extra spaces to save the result, such as word counts. |

### Free Memory

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screenshot of a computer code  AI-generated content may be incorrect.  Figure 2.18 Linked List free memory code | The code is the memory management section from the main() function, with the responsibility to free all memory allocated for both the Review and Transaction lists. At the end of the program, these two while loops are used to delete each node by storing the current node in a temporary pointer first, then moving the head pointer to the next node and deleting the current node in the temporary pointer. This process will continue until all nodes are deleted to ensure that degradation in performance due to unreleased memory is prevented. |
| **Array** | Figure 2.19 Array free memory code | Unlike the linked list that needs a manual to delete each node, the array focuses memory management on the DynArray<T> class. The DynArray destructor will invoke the delete[] on its internal pointer to release the whole heap-allocated block at once. Due to review and trxs is the local variable on main(), thus its destructors will run automatically when it exceeds the scope. This RAII pattern binds data cleaning to the lifetime of the object to ensure all the memory will be released during append and resize. |

### Calculate Time and Space Efficiency

|  |  |  |
| --- | --- | --- |
|  | **Code** | **Explanation** |
| **Linked List** | A screen shot of a computer code  AI-generated content may be incorrect.  Figure 2.20 Linked List calculateAndDisplayEfficiency | The code shows the **calculateAndDisplayEffieciency**, which evaluates and displays the performance of the task by calculating both time and space efficiency. The function will take the start and end timestamps of the task and compute the total duration in microseconds, and derive the average time taken per item processed to calculate the time efficiency. It also estimates the memory space used by Review and Transaction linked lists by multiplying the node counts by the size of their data structures. The function will be called after the user selects the options from the menu to provide insight into how long the task took and how much memory space was used, to give the user a better understanding, and to compare the differences in time and space used between the linked list and the array. |
| **Array** | A computer code with text  AI-generated content may be incorrect.  Figure 2.21 Array calculate time and space efficiency | Time & Space Trade-Offs are finding the balance between the calculation and the complexity of the system and space. Generally, increasing performance requires more memory and vice versa. Giving an example of time & Space Trade-Offs, a dynamic array is used to store reviews and transactions. For instance, when a dynamic array reaches its maximum capacity, it will arrange its size; therefore, can access and insert elements more quickly, but needs more memory to store elements after arranging its size. Moreover, the sorting operation might be time-consuming, especially on processing larger datasets, but it provides simple, executable, and occupies minimal memory. In this section of code, the condition of usage storage is based on the size of the review and transaction dataset, thus performance measurement tool is used to track the time used for sequence operation. This helps to understand or process the data structure and analyze the relationship between time-consuming. |

# **Input-Output Screenshots**

## Menu

|  |  |
| --- | --- |
| **Linked List** | A screen shot of a computer  AI-generated content may be incorrect.  Figure 3.1 Linked List Menu |
| **Array** | Figure 3.2 Array Menu |
| **Explanation** | When the user runs the system, the total valid reviews and transactions value after cleaning will be displayed along with a menu consisting of options for users to choose from. The menu interfaces between the Linked List and Array implementations show identical functional operation and result outputs. Both display the same options: sorting transactions by date, calculating the percentage of Electronics purchases using Credit Card, identifying the most frequent words in 1-star product reviews, and exiting the program. |

## Choice 1

|  |  |
| --- | --- |
| **INPUT** | Figure 3.3 Input Choice 1 |
| **Linked List** | A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.4 Linked List Choice 1 |
| **Array** | Figure 3.5 Array Choice 1 |
| Explanation | When the user executes the sorting operation (Choice 1), a table showing the top 20 transactions sorted by date is displayed. The table consists of information such as CustomerID, Product, Category, Price, Date, and Payment Method. After the list, the time taken for sorting and displaying transactions, the time efficiency per item, and the space used by transactions are displayed. |

## Choice 2

|  |  |  |
| --- | --- | --- |
| **INPUT** | Figure 3.6 Input Choice 2 | |
| **Linked List** | | **Array** |
| A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 3.7 Linked List Choice 2 | | Figure 3.8 Array Choice 2 |
| A screen shot of a computer  AI-generated content may be incorrect.  Figure 3.9 Linked List Choice 2 Percentage | | Figure 3.10 Array Choice 2 Percentage |
| **Explanation** | When the user choose option 2, it displays a list of Electronics transactions paid with credit card, which consists of information such as CustomerID, Product, Category, and Payment Method. After the list is displayed, the percentage of Electronics transactions paid with credit card is shown, followed by the time taken for calculating Electronics purchased using credit card, time efficiency per item, and the space used by reviews and transactions. | |

## Choice 3

|  |  |
| --- | --- |
| **INPUT** | Figure 3.11 Input Choice 3 |
| **Linked List** | A screenshot of a computer  AI-generated content may be incorrect.  Figure 3.12 Linked List Choice 3 |
| **Array** | Figure 3.13 Array Choice 3 |
| **Explanation** | When the user chooses option 3, a list of the top frequent words in 1-star reviews is displayed. The list consists of words along with the frequency of each word shown next to it. After the list, the time taken for analyzing 1-star reviews is shown, followed by the time efficiency per item, space used by reviews, and transactions. |

## Choice 0

|  |  |
| --- | --- |
| **INPUT** | Figure 3.14 Input Choice 0 |
| **Linked List** | A screen shot of a computer  AI-generated content may be incorrect.  Figure 3.15 Linked List Choice 0 |
| **Array** | Figure 3.16 Array Choice 0 |
| **Explanation** | When the user chooses option 0, the user initiates the program's exit process. The system stops accepting further input and exits, followed by a performance summary displayed. The summary includes the space used by reviews and transactions, and the execution time. |

# **Summary Discussions**

## Analyze system efficiency

|  |  |
| --- | --- |
|  | **Output** |
| **Question 1** | Linked List  A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 4.1 Linked List System Efficiency Question 1 |
| Array  A screenshot of a computer screen  AI-generated content may be incorrect.  Figure 4.2 Array System Efficiency Question 1 |
| **Explanation** | * **Difference between times taken**   893,588 microseconds   * **Difference in time efficiency per item**   216.47 microseconds/item   * **Conclusion**   The Linked List took less time for displaying and sorting transactions by date compared to the Array. |
| **Question 2** | Linked List  A screen shot of a computer  AI-generated content may be incorrect.  Figure 4.3 Linked List System Efficiency Question 2 |
| Array  A screen shot of a computer screen  AI-generated content may be incorrect.  Figure 4.4 Array System Efficiency Question 2 |
| **Explanation** | * **Difference between times taken**   94,489 microseconds   * **Difference in time efficiency per item**   22.89 microseconds/item   * **Conclusion**   The Linked List took less time for displaying and searching the payment method “Credit Card” in the “Electronics” category than the Array. |
| **Question 3** | Linked List  A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.5 Linked List System Efficiency Question 3 |
| Array  A screenshot of a computer  AI-generated content may be incorrect.  Figure 4.6 Array System Efficiency Question 3 |
| **Explanation** | * **Difference between times taken**   5,165 microseconds   * **Difference in time efficiency per item**   1.3 microseconds/item   * **Conclusion**   The Linked List took less time for displaying and analyzing the most frequent word in 1-star reviews than the Array. |
| **Space Used** | Linked List  A screen shot of a computer  AI-generated content may be incorrect.  Figure 4.7 Linked List Used Space |
| Array  A screen shot of a computer  AI-generated content may be incorrect.  Figure 4.8 Array Used Space |
| **Explanation** | * **Difference in space used by Reviews**   15,860 bytes   * **Difference in space used in Transactions**   0 bytes   * **Conclusion**   The Array used less memory than the Linked List for storing Reviews, while space usage for Transactions remained equal. |

## Summary and observations made during development

Throughout the development process, it became clear that linked lists work well in situations requiring dynamic data insertion and deletion. Shifting elements can be a time-consuming procedure in arrays, especially when working with huge datasets. The linked list's structure reduces the need for this. Because each node in a linked list stores more pointers, the flexibility that linked lists provide is accompanied by a higher memory usage cost. On the other hand, because data is kept in memory continuously, arrays offer quicker access times and more effective memory use. However, arrays have restrictions when it comes to insertion and deletion operations, particularly when shifting items are required. To highlight the advantages and disadvantages of both data structures, the system also showed how linked lists and arrays perform differently in a variety of data manipulation tasks, including sorting, searching, and filtering. The project also required the use of data cleaning and verification procedures to make sure that only legitimate data was processed because managing dirty and unstructured data was a crucial component. These procedures contributed to the entire system's increased efficiency and dependability. Whether linked lists or arrays are used ultimately relies on the needs of the activity, such as whether quick access or dynamic data changes are important.

## Strengths and weaknesses of the code

The code's main strength is its capacity to manage big datasets well by giving users an array or linked list option. Linked lists are perfect for applications that need frequent data alterations because they enable fast insertions and deletions without requiring the shifting of big blocks of data, which is especially useful in dynamic circumstances. However, because more memory is required to store pointers for every node, linked lists have greater memory costs. On the other hand, arrays provide more effective memory utilization by storing data in a continuous manner, which is advantageous for applications that need to retrieve data quickly. Nevertheless, arrays are less effective for insertion and deletion tasks, especially when moving or resizing big amounts of data. Relying on linear traversal for data analysis and searching is a major flaw in the code that might cause inefficiencies when dealing with bigger datasets. Enhancing performance would involve optimizing the search process using more sophisticated algorithms or indexing strategies. Further affecting system performance is the linked list implementation, which necessitates careful memory allocation and deallocation management because improper memory release could lead to memory leaks, even if array memory management is generally efficient. (Sharma, PrepBytes Blog, 2021)

# **Summarize and Personal thoughts on the assignment**

## Summarize

In conclusion, this project implements two data structures in C++, namely array and linked list to analyze the behavior of e-commerce customers. The program manages to arrange the sequence for transactions based on date, calculates the percentage of electronic products bought using credit card, and identifying the most frequent word used in one-star comments for the products.

Both the array and the linked list can be used to process a big collection of elements, but it is observable that linked list is typically much faster than array. This is because when it comes to operations like adding or deleting elements, linked list only modifies the pointer of the previous node, but array must shift the position of every element within it. On the other hand, in a linked list, a pointer is required by each node to point to the next node, adding extra overhead, while an array stores data in contiguous memory blocks that creates no overhead (Aramendia, 2024). This statement does indeed reflect to our actual results from running the program.

## Potential improvements and system weaknesses

**Weaknesses**

Linked lists will use more memory due to the pointers and lead to a slowdown in speed and sequence.

Arrays will be faster, but struggle with inserting or deleting data due to the need to shift.

The system lacks advanced error handling for corrupted CSV files.

**Improvements**

In order to process the error, adding better data validation is a good option.

Use hybrid data. For example, enhance the performance by combining the array and linked list.

Adding a cache to speed up the access speed of data.

## Personal thoughts on the assignment

In this project, we have understood the essentials of data and memory address management to handle actual files such as reviews and transactions. We realize that making choices between array and linked list will change the attributes of the program, especially in terms of speed and memory space usage. We also learned how to clean and verify messy data that is usually sourced from a real program. Coding a dynamic array and linked list helps us understand the method of allocation in memory and how small mistakes lead to bugs. This project motivates developers to build a structured program and fix the bug nicely, making developers act like problem solvers, not just programmers.

# **6.0 References**

Aramendia, A. I. (2024, May 1). *Arrays vs Linked Lists : A Complete Guide*. Medium. <https://medium.com/@alejandro.itoaramendia/arrays-vs-linked-lists-a-complete-guide-bc23c0ab0e7c>

Duggal, N. (2024, July 23). *Difference between array & linked list in data structure: Simplilearn*. Simplilearn.com. <https://www.simplilearn.com/tutorials/data-structure-tutorial/difference-between-array-and-linked-list-in-data-structure>

Goodrich, M. T., Mount, D. M., & Tamassia, R. (2011).*Data structures and algorithms in C++ (2nd ed).* John Wiley & Sons.

IEEE. (2022, November 23). *Experimental Comparison of Array-based and Linked-based List Implementations*. IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/10083495>

Sharma, A. (2023, April 25). *Advantage and disadvantage of linked list over array*. PrepBytes Blog. <https://www.prepbytes.com/blog/linked-list/advantage-and-disadvantage-of-linked-list-over-array/>

Singh, N. (2023, September 25). *Difference between array and linked list*. Medium. <https://blog.devgenius.io/linked-lists-vs-arrays-whats-the-difference-f2a71b9fbd8>

Sharma, A. (2021, September 28). *Advantage and Disadvantage of Linked List Over Array*. PrepBytes Blog. <https://www.prepbytes.com/blog/linked-list/advantage-and-disadvantage-of-linked-list-over-array/>

‌

‌