USED VEHICLE INVOCING SYSTEM

TECHNOLOGY PARK MALAYSIA

CT077-3-2-DSTR

DATA STRUCTURES

APD2F2206CS(IS)

Group Member: **Yeong Chee Chiew (TP068860)**

Chai Cheng Ti (TP060723)

Siew Yung Hong (TP060743)

Wong Yen Wei (TP063782)

Contents

1. Introduction

Our group is tasked to help develop a used vehicle invoicing system for sale transactions. This system will help the staff, salesperson and manager in dealing with the vehicle data. With using C++ program, we have created a system that has three data structures and that is car, report and bill struct. Each struct will have an insert function for adding data and every new data inserted will add at the last of the struct link list too. The only difference is that the car struct will have reinsert function, this means that when a data of a car that has been removed from the system previously can be reinserted back into the car struct link list.

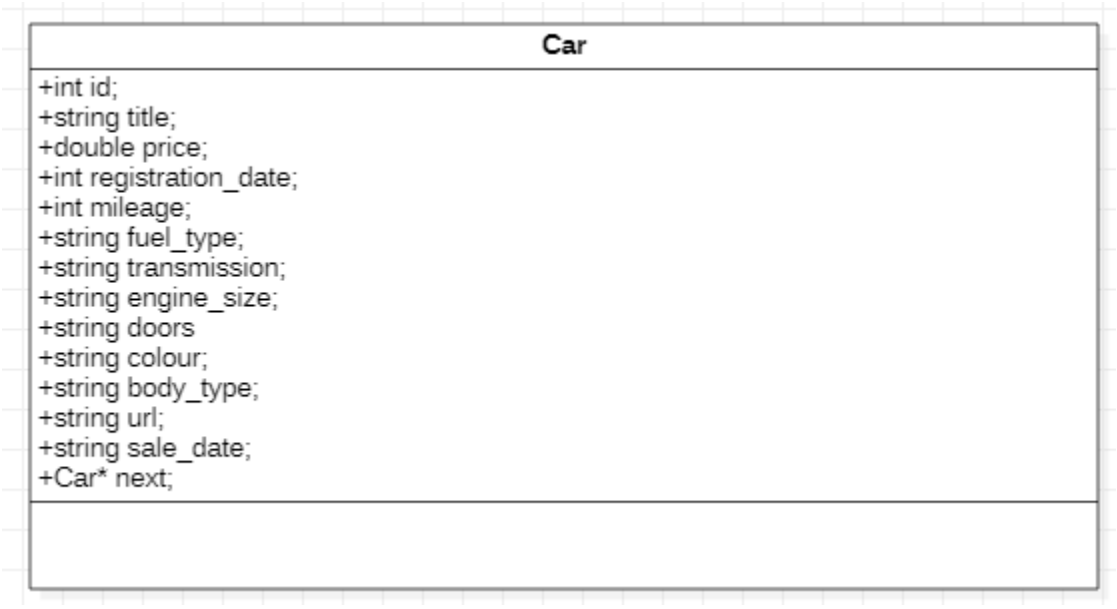

Figure 1.1 Car Struct Class Diagram

These are the attributes used in the Car Struct. The integers are for the items that will have value in them and will also show up in the system while each string naming after each part of a car found in the data.

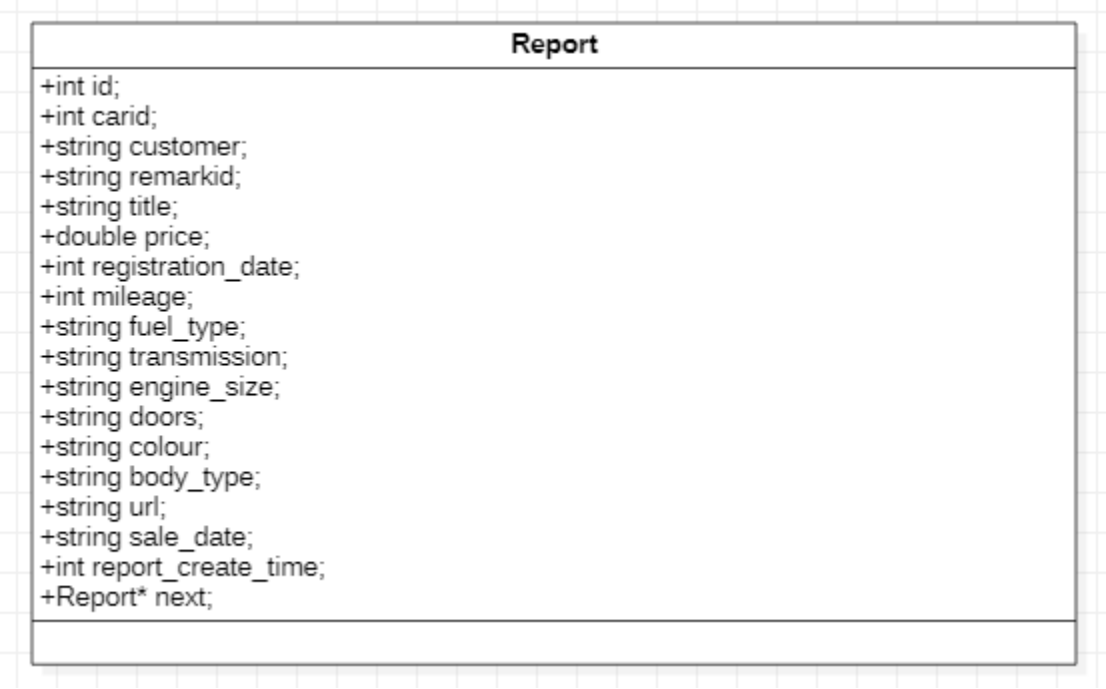

Figure 1.2 Report Struct Class Diagram

These are the attributes used in the Report Struct. It shares the same attributes as Car Struct but it has some extra attributes like for example now it has a string for customer and also integers for their id and also integers for the car id. There's also an extra integer at the end where it will create the time when the report is published.

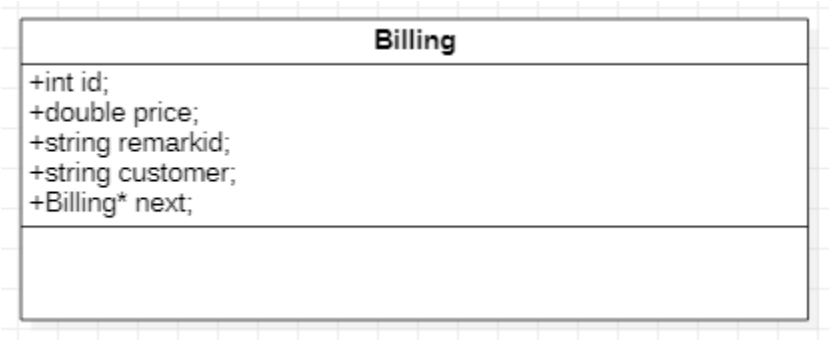

Figure 1.3 Billing Struct Class Diagram

These are the attributes used in the Billing Struct. It has lest attributes compared to both Report Struct and Car Struct because it's at the end of the process when the person is buying a car.

All of the structs are designed to be used as an element in a linked list of car, report and billing. For example, in a car linked list each element is a node that contains some data (in this case, a Car object) and a pointer to the next node in the list. The next member variable in the Car struct is used to store this pointer to the next node in the list.

When creating a linked list of cars, we can also create Car objects and connect them together by setting the next pointer of each node to the next node in the list.

So, using a linked list to represent a list of cars can be useful for various tasks, such as searching for specific cars, inserting new cars into the list, and deleting cars from the list. By using a linked list, we can easily add or remove elements from the list without having to move large amounts of data around.

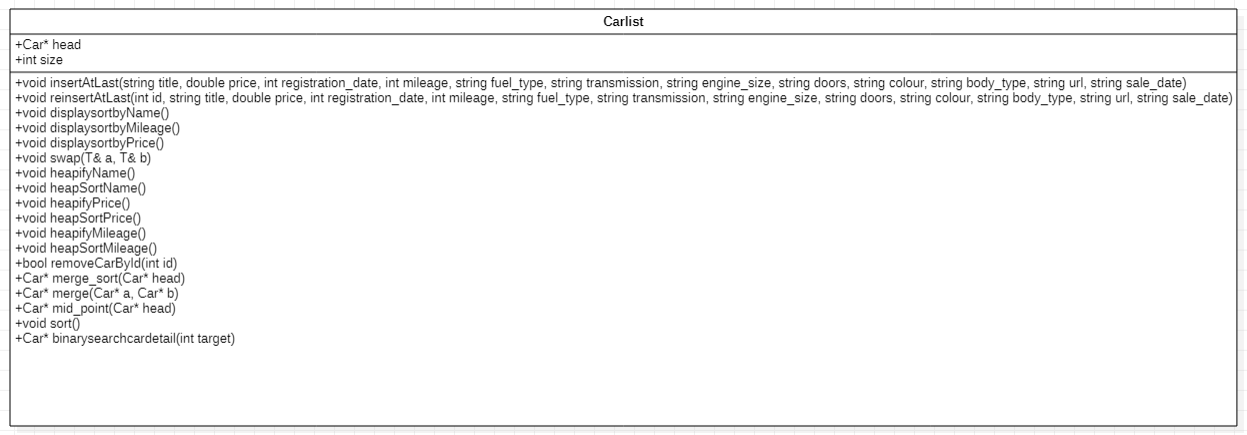

Figure 1.4 CarList Struct Class Diagram

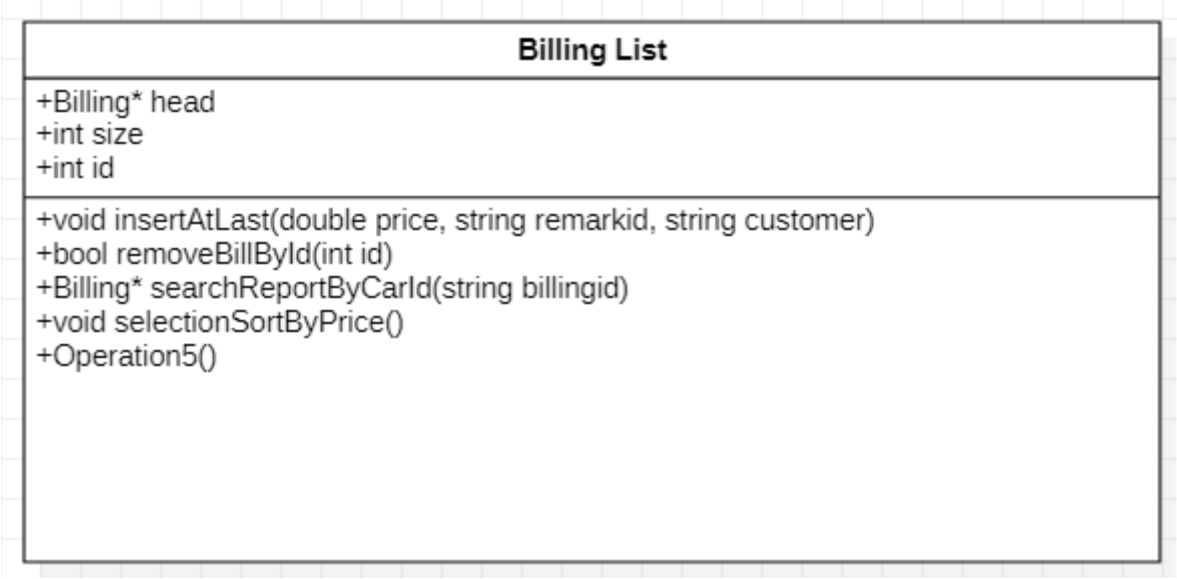

Figure 1.5 Billing List Class Diagram

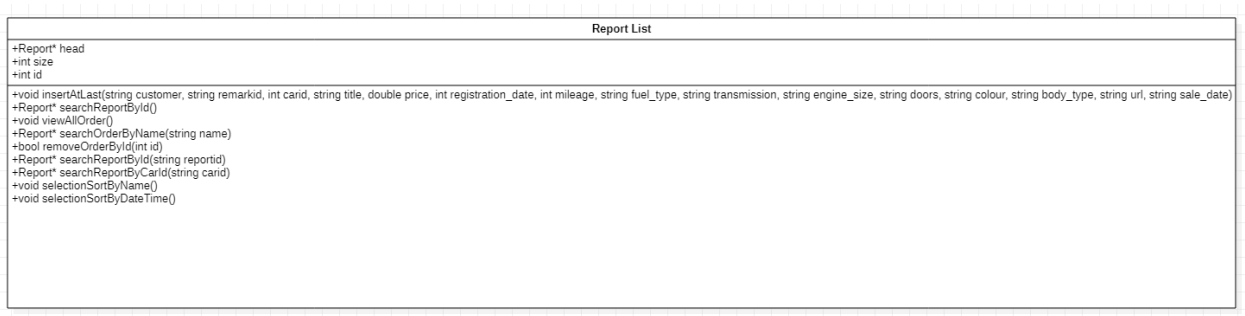

Figure 1.6 Report List Class Diagram

These are the attributes and functions for the Car List, Billing List and Report List Struct. For the functions part, we used void for most of the functions in this system. We use void because it's a function return type, this means that it specifies that the function doesn't return a value. Two void functions that is important is the +void insertAtLast and +void reinsertAtLast. The function +void insertAtLast is present in all three figures. This function works when a data is taken out from the main storage, and when the data is being reentered back into the main storage it will be inserted to the back of the main storage and not in the place where it used to be. The other void function +void reinsertAtLast is different compared to the other as this function is only present in the Car List Struct. The function is similar to the previous one, but the only difference is that instead of data being stored at the back of the main data storage, the data will go back to where it used to be. This

will be elaborated more in detail during the implementation part. The rest of the functions belong to the algorithm, we will explain at next session.

2. Implementation

2.1 Heap Sort

```
//heap sort (algorithm)
void heapifyName(Car* arr, int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l].title > arr[largest].title)
        largest = l;
    if (r < n && arr[r].title > arr[largest].title)
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapifyName(arr, n, largest);
    }
}

void heapSortName(Car* arr, int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapifyName(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapifyName(arr, i, 0);
    }
}
```

Figure 2.1: Heap Sort Algorithm for Name

```
//heap sort
void heapifyPrice(Car* arr, int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l].price > arr[largest].price)
        largest = l;
    if (r < n && arr[r].price > arr[largest].price)
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapifyPrice(arr, n, largest);
    }
}

void heapSortPrice(Car* arr, int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapifyPrice(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapifyPrice(arr, i, 0);
    }
}
```

Figure 2.2: Heap Sort Algorithm for Price

```
//heap sort
void heapifyMileage(Car* arr, int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l].mileage > arr[largest].mileage)
        largest = l;
    if (r < n && arr[r].mileage > arr[largest].mileage)
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapifyMileage(arr, n, largest);
    }
}

void heapSortMileage(Car* arr, int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapifyMileage(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapifyMileage(arr, i, 0);
    }
}
```

Figure 2.3: Heap Sort Algorithm for Mileage

Based on Figure 2.1, Figure 2.2 and Figure 2.3 we have chosen Heap Sort as the most suitable algorithm for it compared to others. Heap Sorts considered one of the widely known technique to perform sorting of data in an array. The advantage of implementing it in this section of code is because of its efficiency and the very low memory usage. (Sharma, 2022) As part of the Heap Sort Algorithm, it also involves with the Heapify Method. The heapify method is the process where it uses an array to form a heap data structure from a binary tree. With the combination of both Heap Sort and Heapify, the code can run smoothly and efficiently.

2.2 Merge Sort

```
//merge sort
Car* merge_sort(Car* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    Car* mid = mid_point(head);
    Car* a = head;
    Car* b = mid->next;
    mid->next = NULL;
    a = merge_sort(a);
    b = merge_sort(b);
    Car* c = merge(a, b);
    return c;
}


    Car* merge(Car* a, Car* b)
    {
        if (a == NULL) {
            return b;
        }
        if (b == NULL) {
            return a;
        }

        Car* c;
        if (a->id < b->id)
        {
            c = a;
            c->next = merge(a->next, b);
        }
        else
        {
            c = b;
            c->next = merge(a, b->next);
        }
        return c;
    }

    Car* mid_point(Car* head)
    {
        if (head == NULL || head->next == NULL) {
            return head;
        }
        Car* fast = head;
        Car* slow = head;
        while (fast != NULL && fast->next != NULL)
        {
            fast = fast->next;
            if (fast->next == NULL)
                break;
            fast = fast->next;
            slow = slow->next;
        }
        return slow;
```

Figure 2.4: Merge Sort Algorithm for Car

Based on Figure 2.4, we have implemented Merge Sort algorithm into it. Merge Sort is another efficient algorithm that continuously cuts down a list into multiple sub lists until each has only one item, then merges those sub lists together into one sorted list. (Khandelwal, 2023) Since we have a lot of car data, it is suitable for this algorithm to function as its able to rearrange any new data into the array quickly and accordingly. Merge sort helps with making sure all data in in proper sequence. One example of how Merge Sort works is when an array of numbers are given like (1,2,3) and the number (2) is taken out. After the number is taken out the number array (1,3) is left and when we insert back the number (2) into it leaving us with (1,3,2). This is when Merge Sort comes in and it helps sort out the array of number back into an orderly sequence.

2.3 Binary Search

```
//binary search
Car* binarysearchcardetail(int target)
{
    sort();
    int n = size;
    int left = 0, right = n - 1;
    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        Car* midNode = head;
        for (int i = 0; i < mid; i++)
            midNode = midNode->next;

        if (midNode->id == target)
            return midNode;

        if (midNode->id < target)
            left = mid + 1;

        else
            right = mid - 1;
    }

    // Target not found
    return NULL;
}
```

Figure 2.5: Binary Search Algorithm for Car Detail

Based on Figure 2.5, we have used Binary Search algorithm into it. Binary Search is used in a sorted array by repeatedly dividing the search interval in half. The advantage of using this algorithm and implementing it into the car detail is because its quick and its able to locate information in our big dataset in a short amount of time.

2.4 Linear Search

```
//linear search
Billing* searchReportByCarId(string billingid) {
    Billing* current = head;
    while (current != nullptr) {
        if (current->id == stoi(billingid)) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}
```

Figure 2.6: Linear Search Algorithm in searching report by car ID

Based on Figure 2.6, we have implemented Linear Search Algorithm into it. Linear Search is an algorithm where it will start at one end and go through each element of a list until the desired element is found; otherwise, the search will continue until the end of the data set. This means that it's suited for small datasets, and that is why we implemented it into this struct as the Billing List Struct has a smaller dataset compared to the dataset in Car List Struct.

2.5 Selection Sort

```
//selection sort
void selectionSortByPrice() {
    Billing* current = head;
    Billing* minNode = NULL;
    Billing* prev = NULL;

    while (current != NULL) {
        // Find the node with the minimum price
        minNode = current;
        for (Billing* temp = current->next; temp != NULL; temp = temp->next) {
            if (temp->price < minNode->price) {
                minNode = temp;
            }
        }
        // Swap the current node with the node with the minimum price
        if (current != minNode) {
            double tempPrice = current->price;
            current->price = minNode->price;
            minNode->price = tempPrice;
        }

        prev = current;
        current = current->next;
    }

    Billing* curr = head;
    cout << "------------------------------------------------------------------------" << endl;
    cout << "            Billing Report     " << endl;
    cout << "------------------------------------------------------------------------" << endl;
    if (size == 0) {
        cout << "------------------------------------------------------------------------" << endl;
        cout << "            No Billing Report      " << endl;
        cout << "------------------------------------------------------------------------" << endl;
    }
else {
    double totalamount = 0;
    while (curr != NULL) {
        cout << "Price Amount: £" << curr->price << endl;
        cout << "--------------------------------" << endl;
        cout << "|| Id: " << curr->id << " || Customer Name: " << curr->customer << " || Price Amount: £" << curr->price << " ||" << endl;
        cout << "------------------------------------------------------------------------" << endl;
        totalamount += curr->price;
        curr = curr->next;
    }
    cout << "Total Amount: £" << totalamount << endl;
    cout << "------------------------------------------------------------------------" << endl;
}
}
```

Figure 2.7: Selection Sort Algorithm in Biling List Struct

Based on Figure 2.7, we have implemented Selection Sort Algorithm into it. Selection Sort is an algorithm where it will sort data according to the condition the user selected in. Just like Linear Search, it is most suitable in a smaller dataset as Report List Struct has a small data set like Billing List Struct.

2.6 Linear Search

```
//linear seearch
Report* searchOrderByName(string name) {
    Report* current = head;
    while (current != nullptr) {
        if (current->customer == name) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}
```

Figure 2.8: Linear Search Algorithm in Report List Struct

```
//linear seearch
Report* searchReportById(int id) {
    Report* current = head;
    while (current != nullptr) {
        if (current->id == id)
            return current;
        current = current->next;
    }
    return nullptr;
}
```

Figure 2.9: Linear Search Algorithm in Report List Struct

```
//linear seearch
Report* searchReportById(string reportid) {
    Report* current = head;
    while (current != nullptr) {
        if (current->id == stoi(reportid)) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}

//linear seearch
Report* searchReportByCarId(string carid) {
    Report* current = head;
    while (current != nullptr) {
        if (current->carid == stoi(carid)) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}
```

Figure 2.10: Linear Search Algorithm in Report List Struct

Based on Figure 2.6 to Figure 2.10. We have implemented Linear Search Algorithm into the Report Struct. Linear Search is an algorithm where it will start at one end and go through each element of a list until the desired element is found; otherwise, the search will continue until the end of the data set. This means that it's suited for small datasets, and that is why we implemented it into this struct as the Report List has a smaller dataset compared to the dataset in Car List Struct, just like the data set in Billing List.

2.7 Selection Sort

```
//selection sort
void selectionSortByName() {
    Report* current = head;
    Report* minNode = NULL;
    Report* prev = NULL;

    while (current != NULL) {
        // Find the node with the minimum title
        minNode = current;
        for (Report* temp = current->next; temp != NULL; temp = temp->next) {
            if (temp->customer < minNode->customer) {
                minNode = temp;
            }
        }

        // Swap the current node with the node with the minimum title
        if (current != minNode) {
            string tempTitle = current->customer;
            current->customer = minNode->customer;
            minNode->customer = tempTitle;
        }

        prev = current;
        current = current->next;
    }
```

```
    Report* curr = head;
    cout << "------------------------------------------------------------------------" << endl;
    cout << "            Report     " << endl;
    cout << "------------------------------------------------------------------------" << endl;
    if (size == 0) {
        cout << "------------------------------------------------------------------------" << endl;
        cout << "            No Report      " << endl;
        cout << "------------------------------------------------------------------------" << endl;
    }
    else {
        while (curr != NULL) {
            time_t t = curr->report_create_time; // convert to time_t
            cout << "Customer Name: " << curr->customer << endl;
            cout << "-------------------------------" << endl;
            cout << "|| Id: " << curr->id << " || Customer Name: " << curr->customer << " || Created Time: " << asctime(localtime(&t)) <<
            cout << "------------------------------------------------------------------------" << endl;
            curr = curr->next;
        }
    }
}
```

Figure 2.11: Selection Sort Algorithm in Report List Struct

```
//selection sort
void selectionSortByDateTime() {
    Report* current = head;
    Report* minNode = NULL;
    Report* prev = NULL;

    while (current != NULL) {
        // Find the node with the minimum price
        minNode = current;
        for (Report* temp = current->next; temp != NULL; temp = temp->next) {
            if (temp->report_create_time < minNode->report_create_time) {
                minNode = temp;
            }
        }
        // Swap the current node with the node with the minimum price
        if (current != minNode) {
            double tempPrice = current->report_create_time;
            current->report_create_time = minNode->report_create_time;
            minNode->report_create_time = tempPrice;
        }

        prev = current;
        current = current->next;
    }
```

```
    Report* curr = head;
    cout << "------------------------------------------------------------------------" << endl;
    cout << "            Report     " << endl;
    cout << "------------------------------------------------------------------------" << endl;
    if (size == 0) {
        cout << "------------------------------------------------------------------------" << endl;
        cout << "            No Report      " << endl;
        cout << "------------------------------------------------------------------------" << endl;
    }
    else {

        while (curr != NULL) {
            time_t t = curr->report_create_time;
            cout << "Created Time: " << asctime(localtime(&t)) << endl;
            cout << "-------------------------------" << endl;
            cout << "|| Id: " << curr->id << " || Customer Name: " << curr->customer << " || Created Time: " << asctime(localtime(&t)) << " || Car Id: " << curr->cari
            cout << "------------------------------------------------------------------------" << endl;
            curr = curr->next;
        }
    }
}
```

Figure 2.12: Selection Sort Algorithm in Report List Struct

Based on Figure 2.11 and Figure 2.12, we have implemented Selection Sort Algorithm into it. Selection Sort is an algorithm where it will sort data according to the condition the user selected in. Just like Linear Search, it is most suitable in a smaller dataset as Report List Struct has a small data set like Billing List Struct.

3. Result

3.1 Intro Page

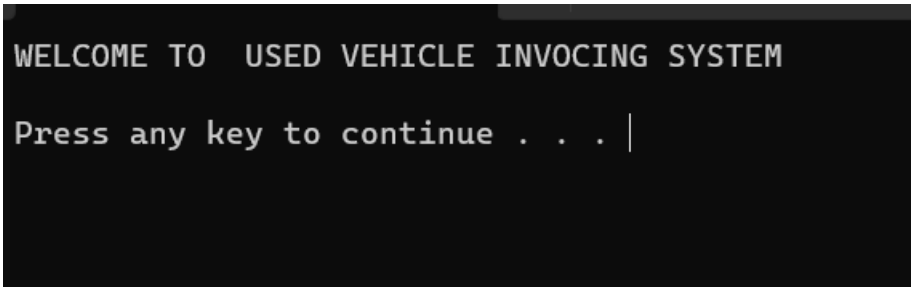

Figure 3.1 Intro page

When the system starts running, an intro page is prompt to the user welcoming the user to use the vehicle invoicing system.

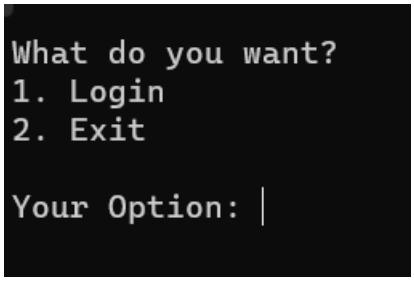

Figure 3.2 Homepage menu

After user pressed a key, the Homepage is prompt to the user. User can select to login or exit the system.

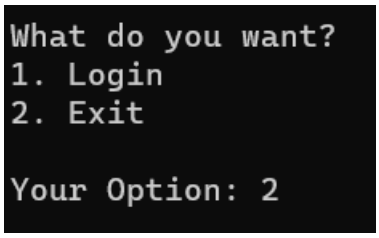

Figure 3.3 Input to exit system

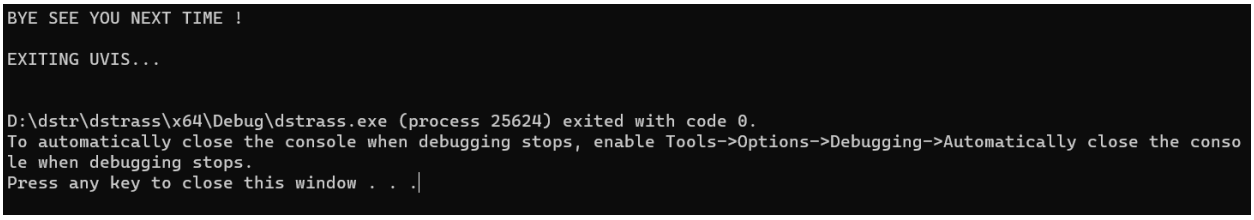

Figure 3.4 Successfully exit system

If user selected to exit the system, system will greet the user and end the system.

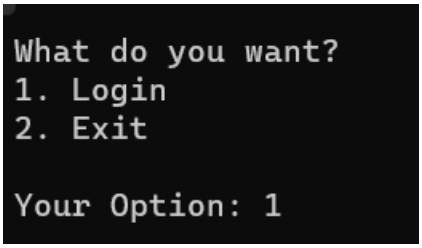

Figure 3.5 Input to login system

If user wants to login into the system, user have to type 1 in the option and hit enter.

3.2 Manager Functions

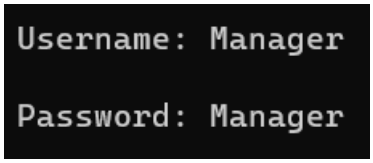

Figure 3.6 Input manager username and password

After user selected to login into the system, user has to type in their username and password to let system check which type of the user he or she is.

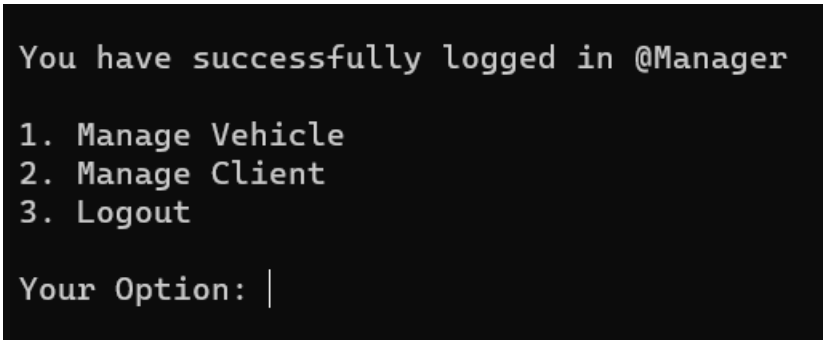

Figure 3.7 Manager menu

If the user is a manager type of user, the manager menu is prompt to the user. Manager's function is to manage vehicle, manage client and logout the system.

3.3 Salesperson Functions

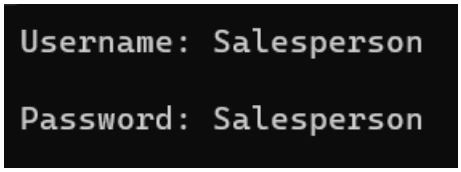

Figure 3.8 Input salesperson username and password

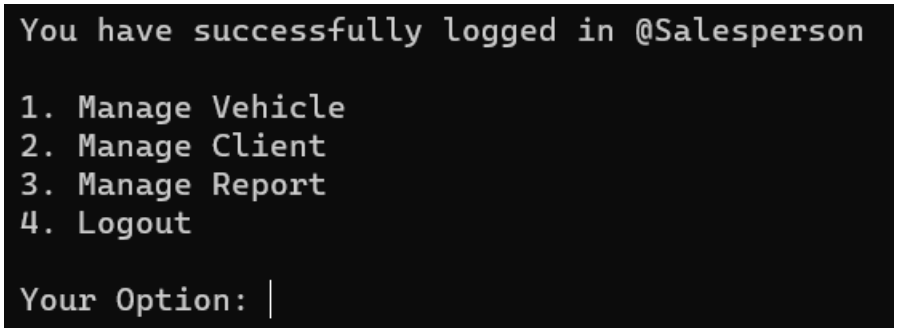

Figure 3.9 Salesperson menu

Salesperson menu is prompt to the user after the user logged in as a salesperson. Salesperson's functions is similar to manager's but has one additional function which is 'Manage Report'

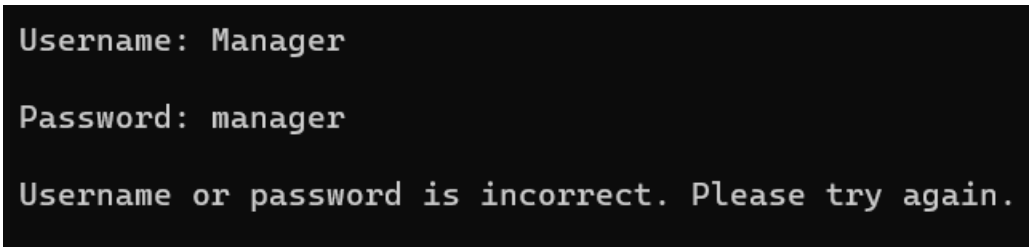

Figure 3.10 Validation of login input

Validations are made while user try to log in into the system, either one of the inputs are incorrect will lead to unable to log in into the system.

3.4 Manage Vehicle

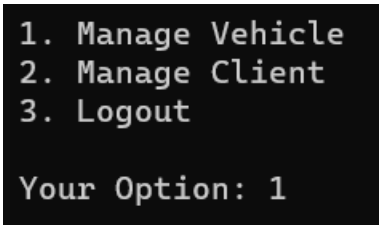

Figure 3.11 Input manage vehicle

When user wants to manage vehicle, he can input ‘1’ as the option.

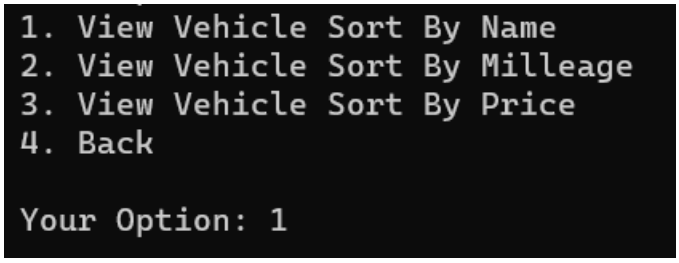

Figure 3.12 Manage vehicle menu

The menu to manage vehicle is prompt to the user for selecting the function they want. ‘1’ refers to sorting vehicles by name, ‘2’ refers to sorting the vehicles by mileage, 3’3 refers to sorting vehicles by price and ‘4’ refers to go back to the last page.

3.5 View Vehicle Sort by Name

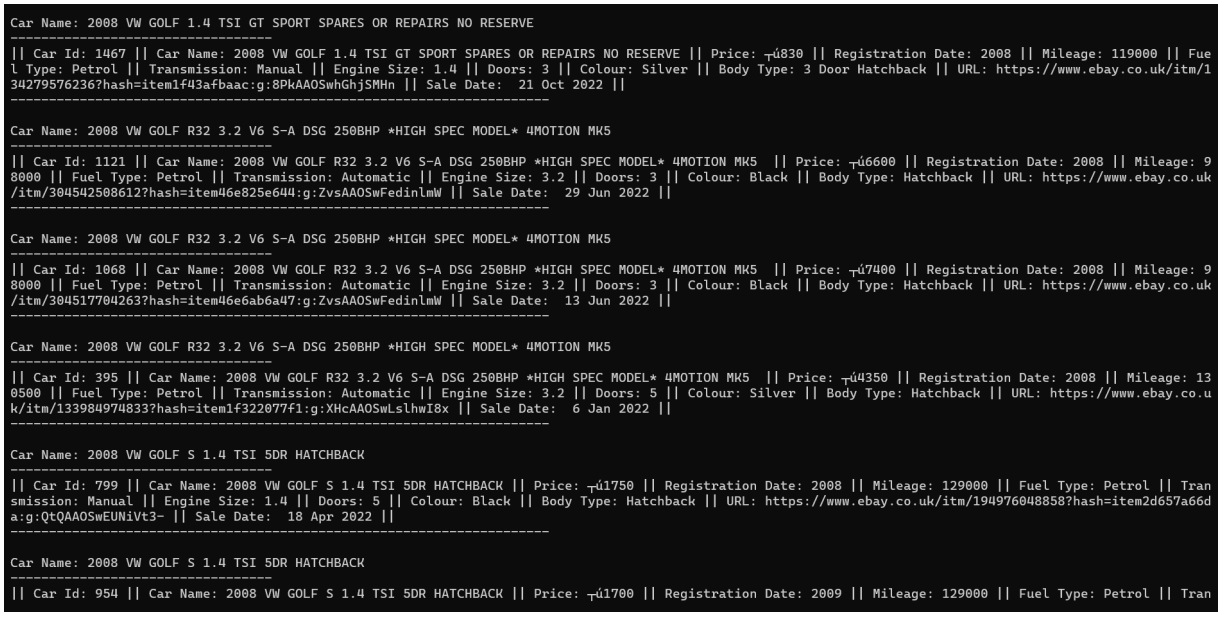

Figure 3.13 Top part of vehicle sort by name

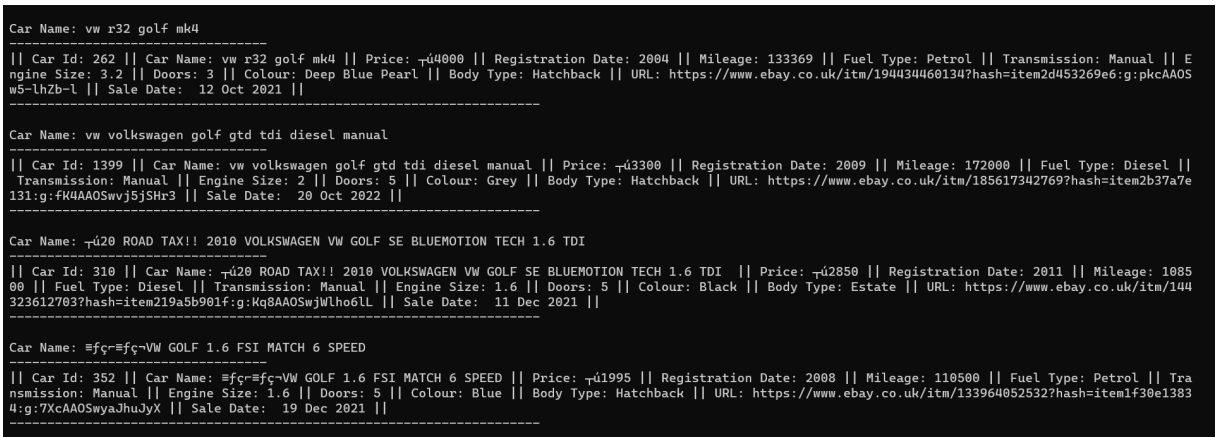

Figure 3.14 Bottom part of vehicle sort by name

From above output shows the sorting of the vehicles by name, the algorithm sorts from numbers, then alphabet, uppercase to lowercase. The system also shows the information about the vehicle along with the sorted data.

3.6 View Vehicle Sort by Mileage

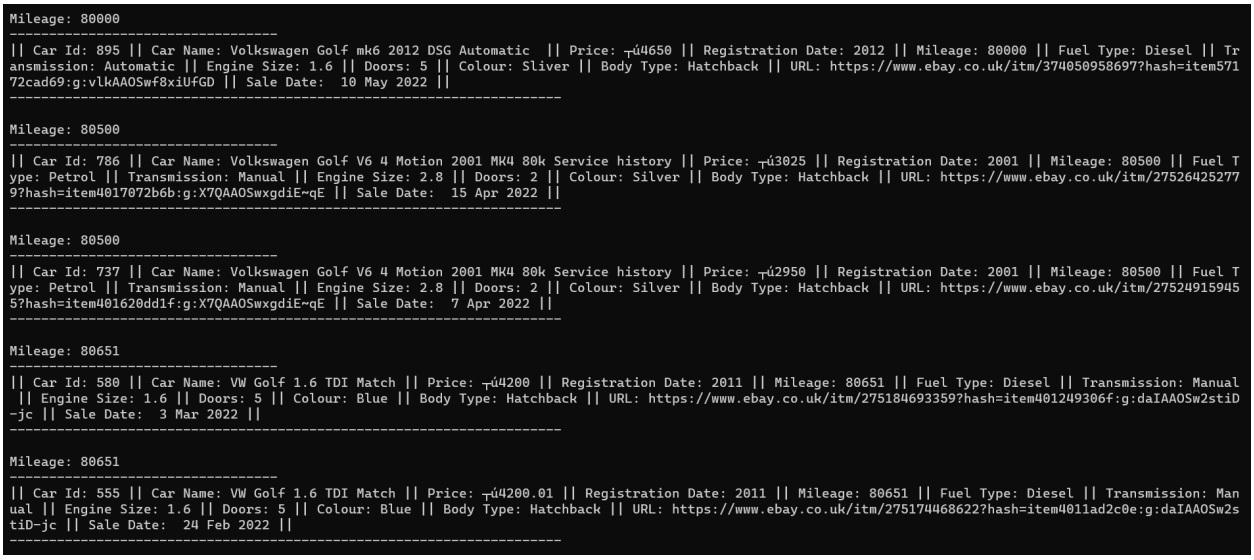

Figure 3.15 Top part of vehicle sort by mileage

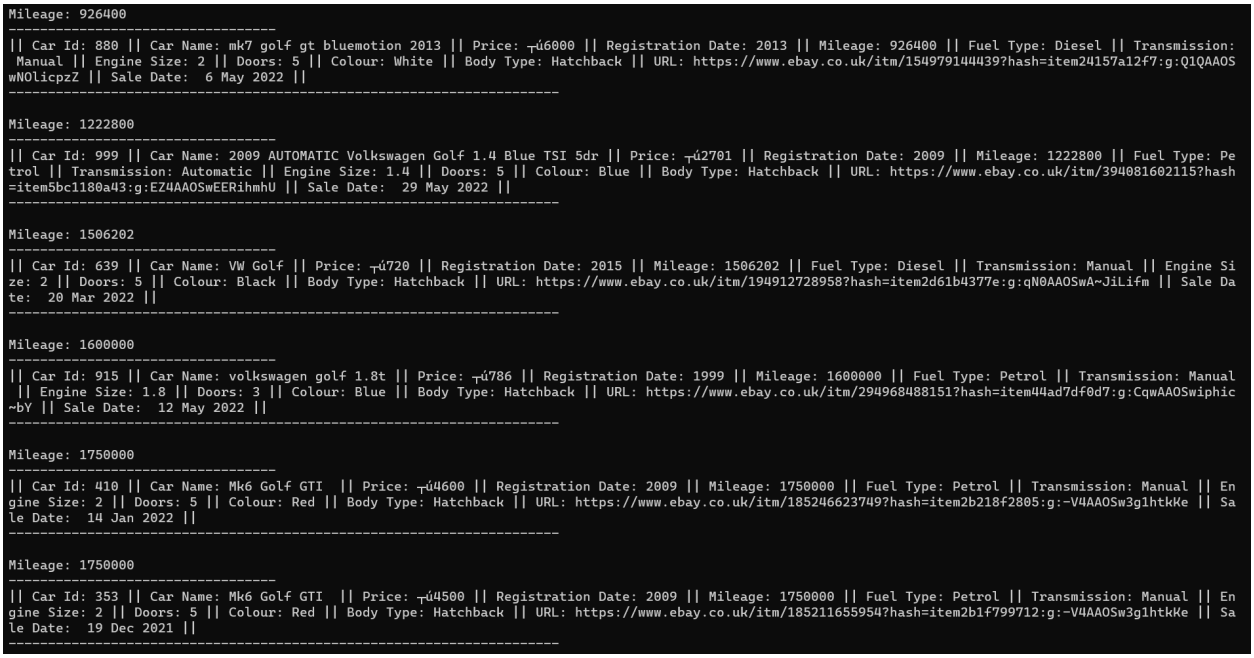

Figure 3.16 Bottom part of vehicle sort by mileage

From above output shows the sorting of the vehicles by mileage, from lower value to higher value. The system also shows the information about the vehicle along with the sorted data.

3.7 View Vehicle sort by Price

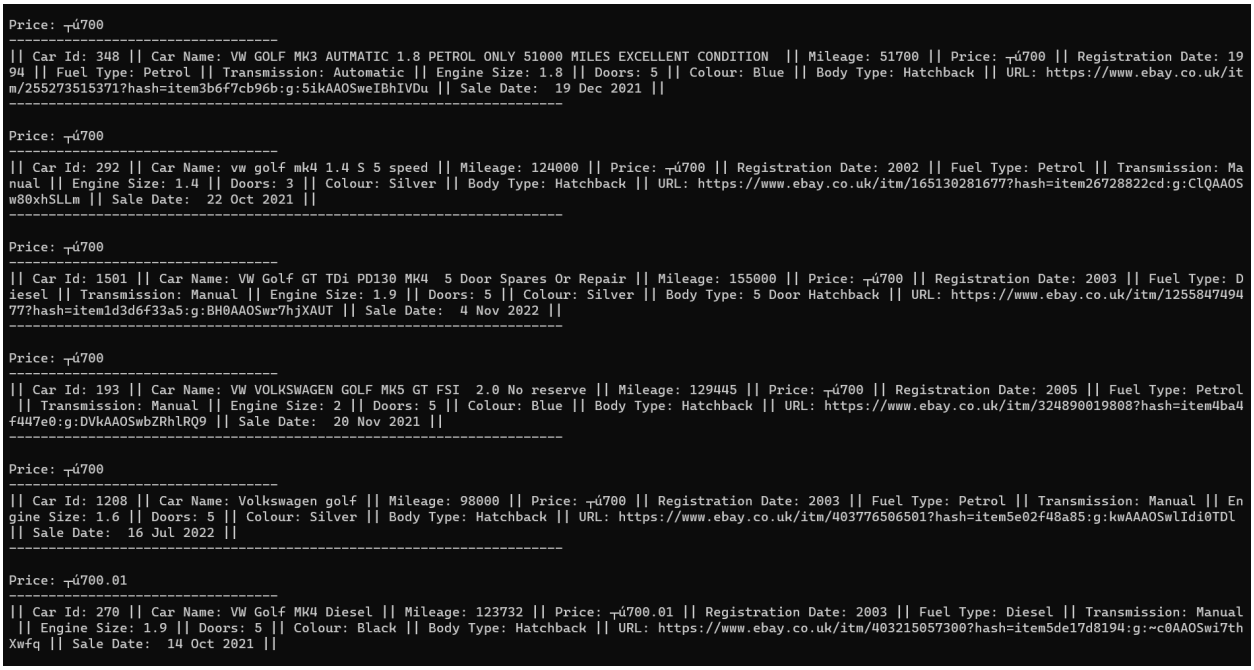

Figure 3.17 Top part of vehicle sort by price

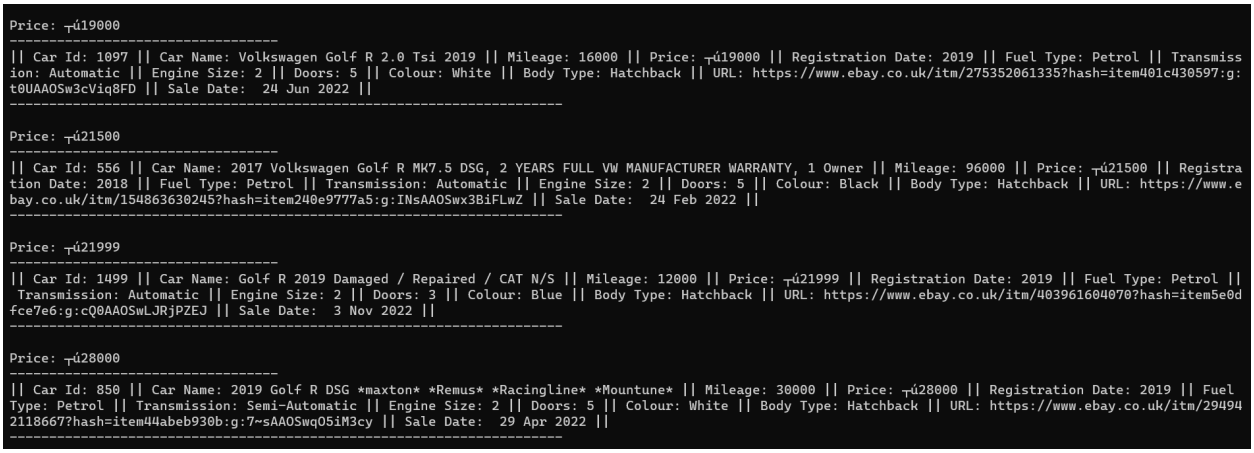

Figure 3.18 Bottom part of vehicle sort by price

From above output shows the sorting of the vehicles by price, from lower value to higher value. The system also shows the information about the vehicle along with the sorted data.

3.8 Book Vehicle after View Vehicle

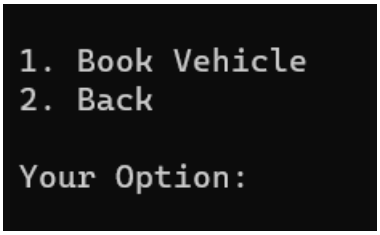

Figure 3.19 Menu after viewing the sorted vehicles

After viewing the vehicles, system prompt to ask user to book or go back to the previous page.

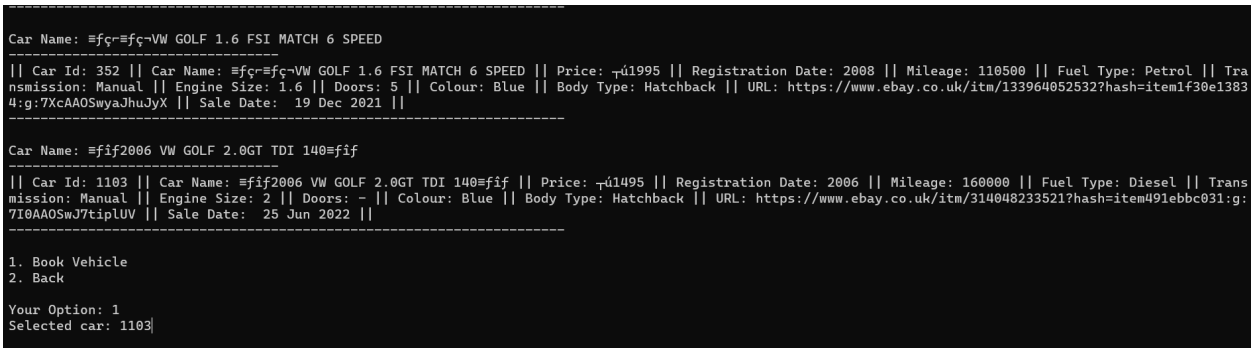

Figure 3.20 Book a vehicle

If user wants to book vehicle, user has to type the selected vehicle ID.

Figure 3.21 Booking successfully and input customer name

The system checks the vehicle ID and shows the information about the vehicle ID and request client's name for record.

Figure 3.22 Output receipt of the booking

After finish booking, a receipt prompts for the manager or salesperson to check. Client's name, selected vehicle ID, vehicle information and booking datetime will be recorded into the system for future use.

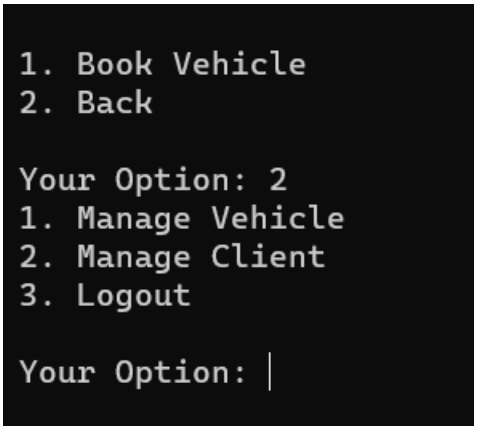

Figure 3.23 Input back to menu

3.9 Manage Client

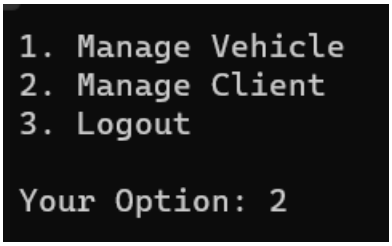

Figure 3.24 Input manage client

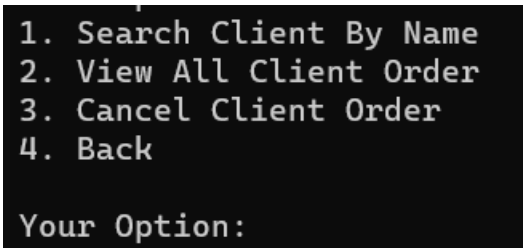

Figure 3.25 Manage client menu

The menu includes function to search client by name, view all client order and cancel client order.

3.10 Search Client by Name

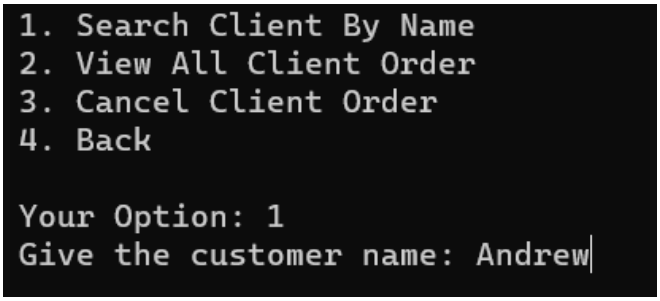

Figure 3.26 Search client and input client name

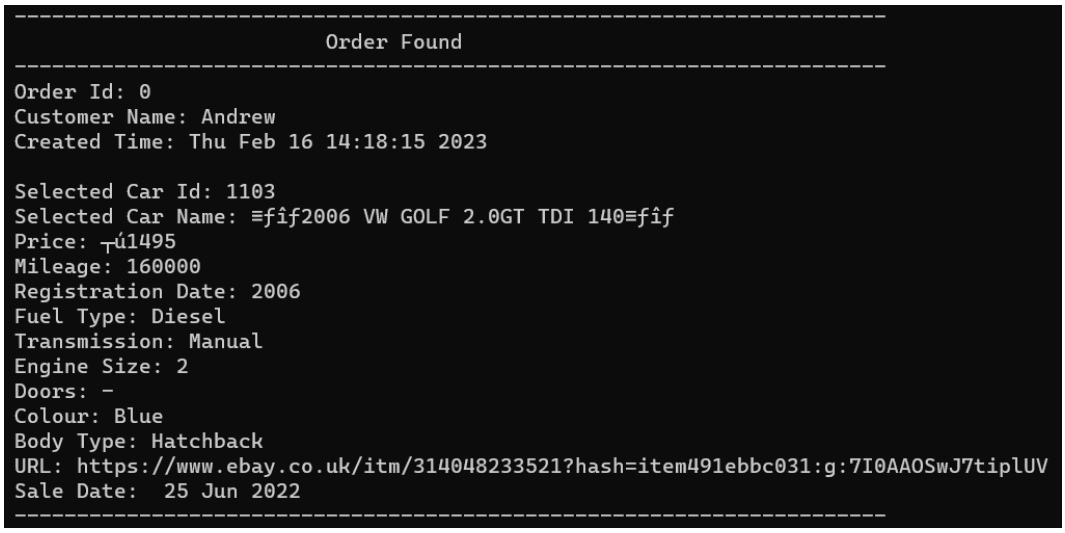

Figure 3.27 Output of the client's order

When checking the client's order, user has to input the client's name for the system to check. After getting the record, all order that the client made will be shown.

3.11 View All Client Order

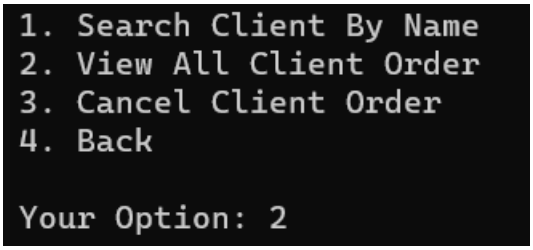

Figure 3.28 Input view all client order

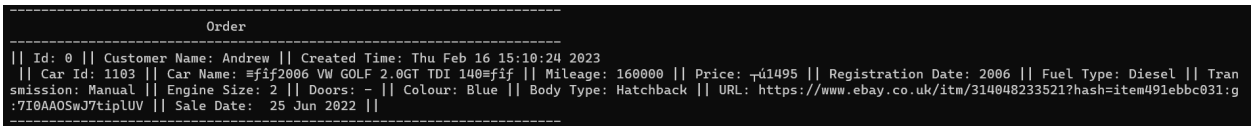

Figure 3.29 Output of all client order

To view all client's orders, system will show all the orders made and its records and information.

3.12 Cancel Client Order

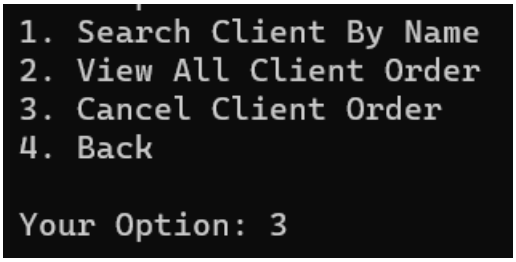

Figure 3.30 Input cancel client order

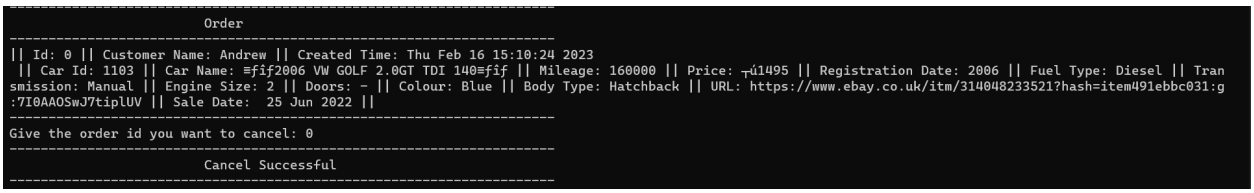

Figure 3.31 Output of cancel client order and cancellation

To cancel an order, the system shows all the previous order and user has to select the order ID to be cancelled, then the cancellation is done. The record will be deleted from the system.

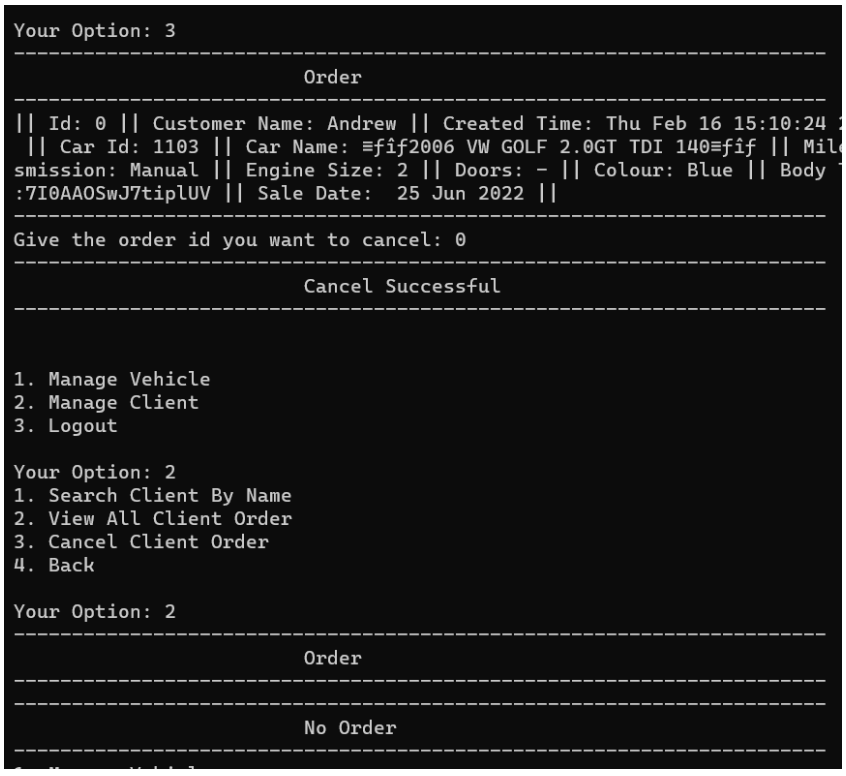

Figure 3.32 Check all client's order after cancelling

3.13 Salesperson Manage Report

```
You have successfully logged in @Salesperson

1. Manage Vehicle
2. Manage Client
3. Manage Report
4. Logout

Your Option: 3
```

Figure 3.33 Input salesperson manage report

```
1. View All Sale Report By Name
2. View All Sale Report By DateTime
3. View All Billing Report By Price
4. Search Sale Report By Id
5. Search Billing Report By Id
6. Search Sale Report By Car Id
7. Back

Your Option:
```

Figure 3.34 Output manager report menu

Salesperson has a additional function which is to manage report. When this function is called, a menu prompt to the salesperson, Salesperson can view all sale report sort by name or datetime, view billing report sort by price, and also search sale report by ID or vehicle ID, and search billing report by ID.

3.14 View All Sale Report by Name

```
1. View All Sale Report By Name
2. View All Sale Report By DateTime
3. View All Billing Report By Price
4. Search Sale Report By Id
5. Search Billing Report By Id
6. Search Sale Report By Car Id
7. Back

Your Option: 1
```

Figure 3.35 Input view all sale report by name

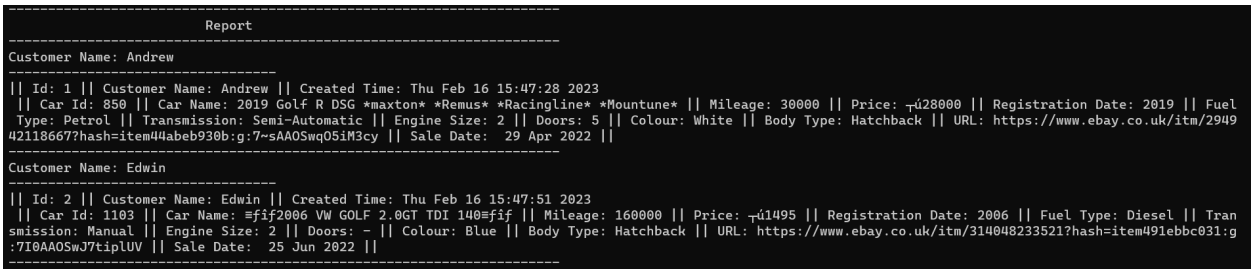

Figure 3.36 Output view all sale report by name

The above output shows all sale report sorted by name from A-Z. Information about the report is carried with the name.

3.15 View All Sale Report by DateTime

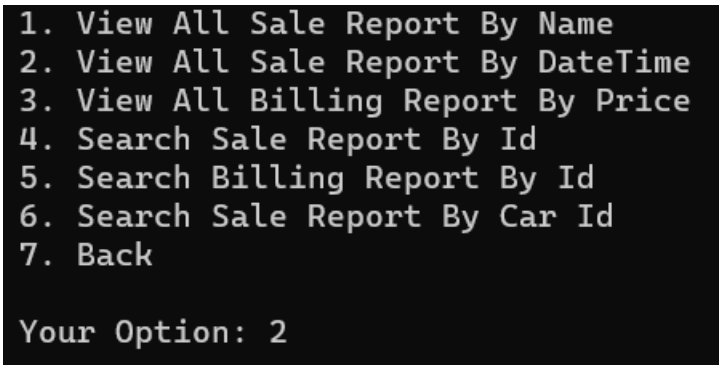

Figure 3.37 Input view all sale report by DateTime

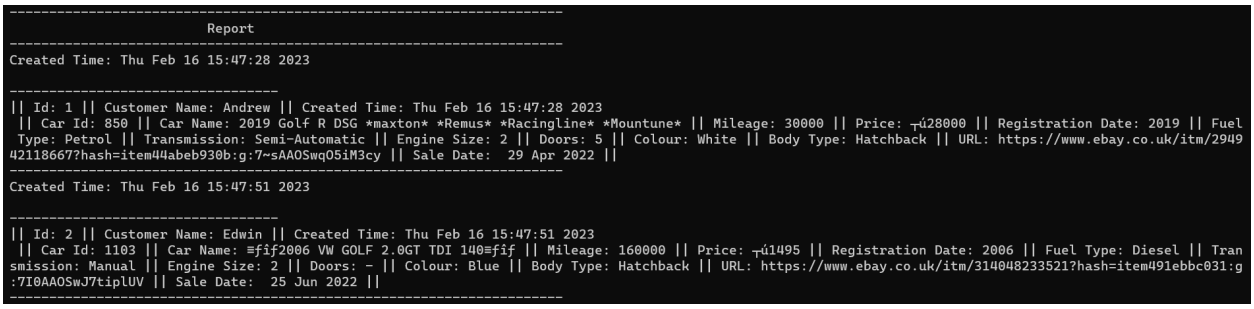

Figure 3.38 Output view all sale report by DateTime

The above output shows all sale report sorted by name from the earlier records to the latest record. Information about the report is carried out.

3.16 View All Billing Report by Price

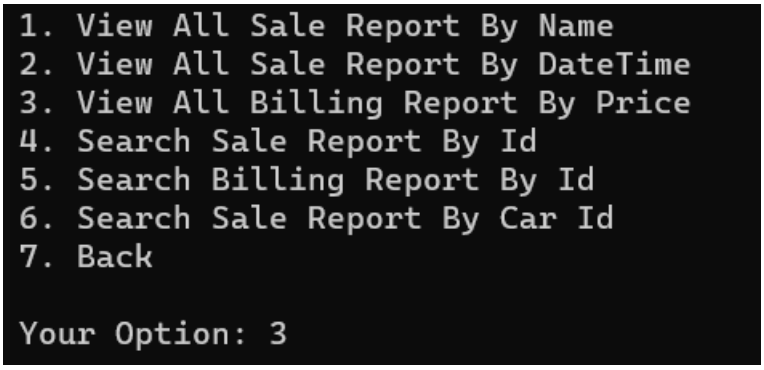

Figure 3.39 Input view all billing report by price

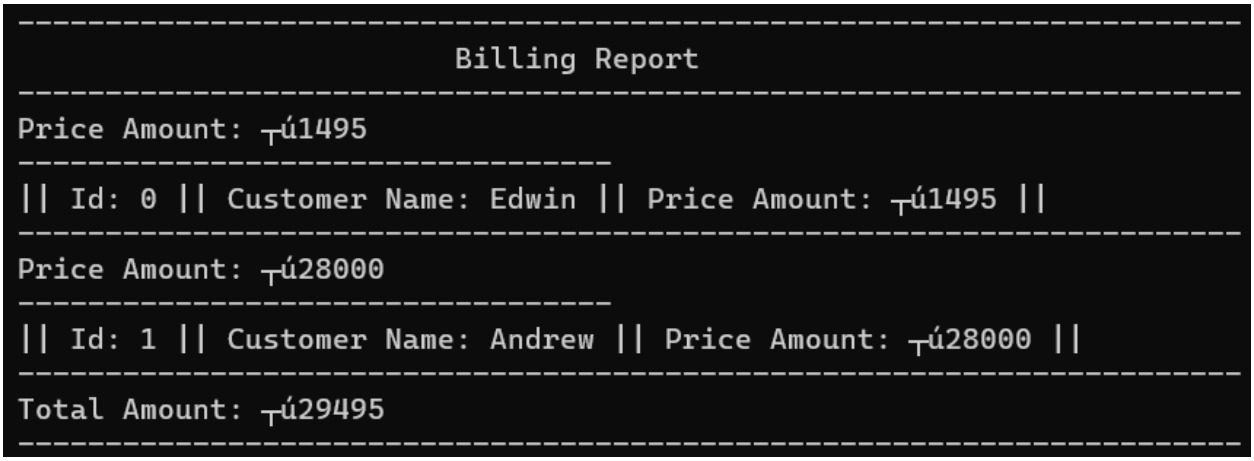

Figure 3.40 Output view all billing report by price

The above output shows all billing report sorted by price from lowest to highest record. Information about the report is carried out.

3.17 Search Sale Report by ID

```
1. View All Sale Report By Name
2. View All Sale Report By DateTime
3. View All Billing Report By Price
4. Search Sale Report By Id
5. Search Billing Report By Id
6. Search Sale Report By Car Id
7. Back

Your Option: 4
```

Figure 3.41 Input search sale report by ID

```
Give the report id: 1
----------------------------------------------------------------------
                        Report Found
----------------------------------------------------------------------
Report Id: 1
Customer Name: Andrew
Selected Car Id: 850
Selected Car Name: 2019 Golf R DSG *maxton* *Remus* *Racingline* *Mountune*
Price: ┬ú28000
Mileage: 30000
Registration Date: 2019
Fuel Type: Petrol
Transmission: Semi-Automatic
Engine Size: 2
Doors: 5
Colour: White
Body Type: Hatchback
URL: https://www.ebay.co.uk/itm/294942118667?hash=item44abeb930b:g:7~sAAOSwqO5iM3cy
Sale Date:  29 Apr 2022
----------------------------------------------------------------------
```

Figure 3.42 Output search sale report by ID no. 1

The above output shows the sale report searched by the report ID. Information about the report ID is prompt to the salesperson.

3.18 Search Billing Report by Id

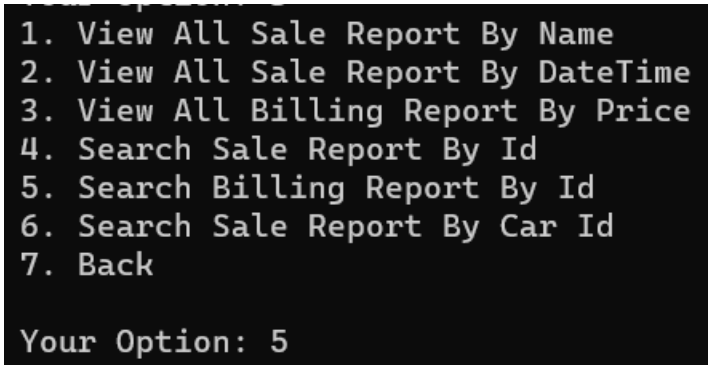

Figure 3.43 Input search billing report by ID

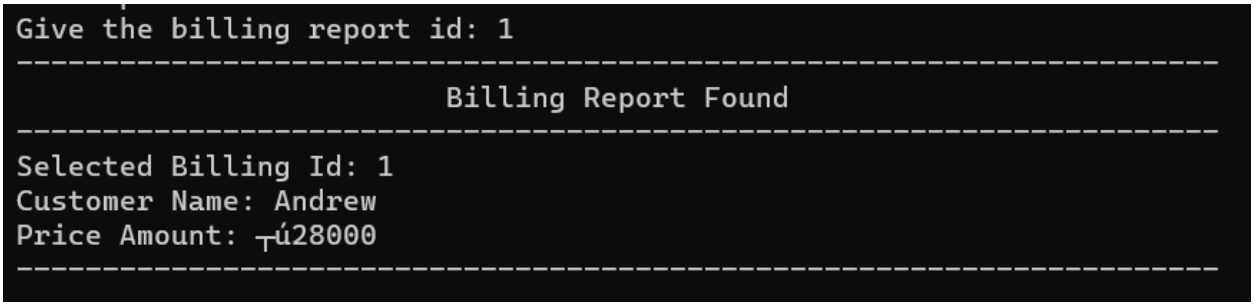

Figure 3.44 Output search billing report by ID no. 1

The above output shows billing report search by the report ID. Information about the billing report ID is prompt to the salesperson.

3.19 Search Sale Report by Car ID

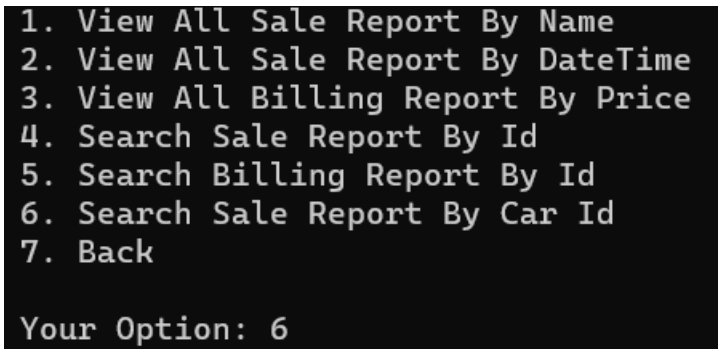

Figure 3.45 Input search sale report by car ID

Figure 3.46 Output search sale report by car ID no. 1103

The above output shows the sale report searched by the car ID. Information about sale report of the specific car ID is prompt to the salesperson.

4. Conclusion

The used car invoicing system developed using C++ is a commendable achievement. The system is designed to cater to two types of users, managers, and salespersons, both of whom can manage vehicles and clients and log in and out of the system. Moreover, salespersons have an additional function to manage reports. The system employs several searching and sorting algorithms, including binary search, linear search, and heap sort, to enable efficient data processing.

However, the lack of input validation is one of the system's limitations. Incorrect data entry by the user may result from this. Implementing input validation might be helpful in getting around this restriction. Try-catch blocks and the "isdigit ()" function can be used to validate the input data type and avert application crashes.

In conclusion, developing a system of this magnitude requires expertise and significant commitment. It is an inspiring experience during the implementation of the system, it is enjoyable and informative. Consistent improvement of coding abilities and implementation of best practices, such as input validation, can significantly enhance system robustness and efficiency.

5. References

Academy, K. (n.d.). Binary Search. Retrieved from Khan Academy: https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search

GeeksforGeeks. (2022, November 2). Binary Heap. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/binary-heap/

GeeksforGeeks. (2023, February 15). Heap sort. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/heap-sort/

Khandelwal, V. (2023, February 14). What is Merge Sort Algorithm: How does it work, and More. Retrieved from Simplilearn: https://www.simplilearn.com/tutorials/data-structure-tutorial/merge-sort-algorithm

Sharma, R. (2022, November 23). Heap sort in data structures: Usability and performance. Retrieved from upGrad blog: https://www.upgrad.com/blog/heap-sort-in-data-structures/#:~:text=in%20Data%20Structure-,Advantages%20and%20Disadvantages%20of%20using%20Heap%20Sort%20in%20Data%20Structure,with%20very%20low%20memory%20usage.

Toturialspoint. (n.d.). Data Structure and Algorithms Linear Search. Retrieved from Toturialspoint: https://www.tutorialspoint.com/data_structures_algorithms/linear_search_algorithm.htm