GROUP ASSIGNMENT

Technology Park Malaysia

CT077-3-2-DSTR

DATA STRUCTURE GROUP R

APU2F2211CS(DA)

HANDOUT DATE: 08 MAY 2023

HAND-IN DATE: 13 AUGUST 2023

WEIGHTAGE: 50%

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the administrative counter.**
2. **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing)**
3. **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**
4. **Cases of plagiarism will be penalized.**
5. **The assignment should be bound appropriately (comb bound or stapled).**
6. **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**
7. **You must obtain 50% overall to pass this module.**

Group R	
Student ID	**Student Name**
TP065116	Neaw Aik Ka
TP067345	Edmund Chen Siang Zuan
TP058190	Muhammad Aqieff bin Mohd Noh
TP067349	Nadila Binti Ahmad Shahrul Nizam

Table of Contents

1 Introduction

Asia Pacific Home (APH) is a project proposed to help promote the renting property within Malaysia's Klang Valley area. As a 4-person developers' team from APU tech-solution, we designed this APH system to allow the administrator to manage the information and advertise the property from an interfacing system called Advert web (mudah). The main features include searching and renting property, updating property information, managing the admin and manager, and, last but not least, providing clients with a comprehensive report of the system and property popularity.

The techniques used in this proposed system include multiple types of data structures, sorting, and searching algorithms. The data structures used encompasses circular linked list, doubly circular linked list, queue, stack, and dynamic array. The proposed sorting algorithms are merge and quick sort, whereas the searching algorithms use binary and linear search. In the sorting and searching section, we will also compare these algorithms to choose the best algorithm based on their runtime.

The dataset used in this project is called 'mudah-apartment-kl-selangor.csv.' It contains detailed information on the Kuala Lumpur and Selangor area property, which is going to be posted on mudah. The following are the metadata of the attributes provided in the dataset.

Attribute name	**Data type**	**Description**
Ads_id	Unique identifier	The unique identifier of each advertisement
Prop_name	string	Name of the property
Completion_year	numeric	The completion year of the property
Monthly_rent	Numeric	The monthly rental fees of the property in Ringgit Malaysia (RM)
Location	String	The location of the property
Property_type	String	The type of the property (condominium, studio, flat, duplex, apartment, etc)
Rooms	Numeric	The number of rooms in the property unit
Parking	Numeric	The area of the property parking unit
bathroom	Numeric	The number of bathrooms in the property unit
Size	Numeric	The size of the property in square feet (ft^2)
Furnished	String	The furnishing status of the property (fully furnished, partial-furnished, non-furnished)
Facilities	String	The facilities included for the property
Additional_facilities	string	The additional facilities included for the property

2 Assumption

A few notable circumstances in the rent system designed by APH are not mentioned in the question prompt, which will be further explained in this section.

Firstly, the system employs a robust authentication process that mandates secure login protocols customized for various user categories. This meticulous approach ensures robust data security and the uncompromised integrity of the system.

In addition, the system's functionality is founded on a hierarchical access structure. Specifically, exclusive authority to oversee and manage both Tenant and Manager accounts resides with the Admin. This system design bolsters managerial accountability and reinforces the security framework.

Moreover, the system is thoughtfully tailored to align with Tenant preferences and needs. Tenants enjoy the autonomy to personalize search outcomes using sorting options rooted in diverse variables, including property name, completion year, and monthly rent. Furthermore, Tenants can choose Quick Sort and Merge Sort as their preferred sorting algorithms. This tailored approach guarantees that the sorting methodology directly aligns with user expectations, thus enhancing their overall experience.

Last but not least, a noteworthy feature allows Tenants to curate lists of preferred properties. This functionality has been meticulously designed, enabling the addition of properties solely through unique property IDs. This method ensures transparent data accuracy and clear organization, streamlining the property selection process.

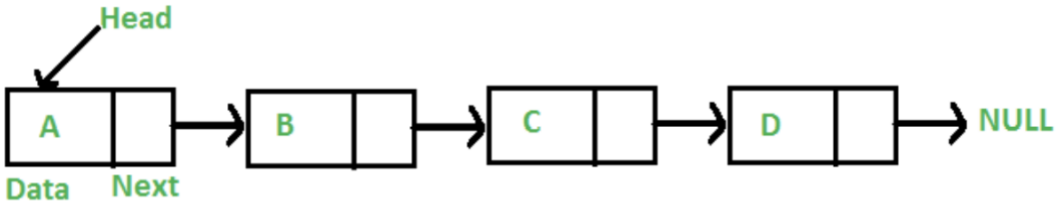
In summary, the APH’s rent system encompasses a cluster of crucial attributes that form the bedrock for a secure, user-centric, and efficient property rental experience. These aspects are in hopes to be able to contribute to the system’s seamless and purpose-driven nature.

3 Data Structure/Class

3.1 Singly Linked List

A singly linked list is a type of data structure. It consists of a sequence of nodes, where each node contains two parts: data and a reference (or link) to the next node in the sequence. The last node in the list points to NULL, indicating the end of the list.

3.1.1 Data Visualization



Taken from (Pkthapa, 2023)

3.1.2 Code Structure

```
template <typename T>
struct ListNode {
    T val;
    ListNode<T>* next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(T x) : val(x), next(nullptr) {}
    ListNode(T x, ListNode<T>* next) : val(x), next(next) {}
};
```

```
template <typename T>
class vector {
private:
    int length;
    ListNode<T>* head;
    ListNode<T>* tail;

public:
    vector() : length(0), head(nullptr), tail(nullptr) {}
```

Dynamic Size

The size of a linked list does not need to be declared hence it can be increased or decreased according to the implementation. The dynamic sizing of a linked allows users to add or delete elements without the need to care for the size or limit.

Memory Efficiency

Due to the dynamic size of the linked list, no memory is wasted for storing the data. The memory is only allocated according to the specified element inserted into the linked list during runtime.

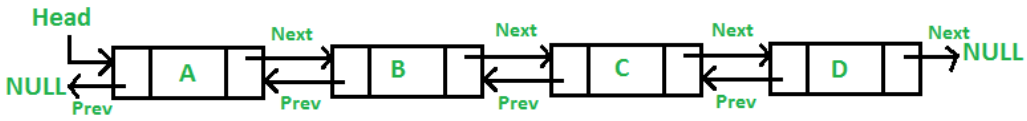
Insertion and Deletion

Insertion and deletion operations are easy to implement for linked list as the only the address for the pointers is updated when performing the operations (Neeraj, 2023).

3.2 Doubly Linked List

A doubly linked list is a type of data structure. It consists of nodes where each node contains three parts, data, a next pointer and a prev pointer. The next pointer points towards the next node present in the linked list while the prev pointer points toward the previous node in the linked list. A NULL value for the next pointer indicates the end of the list while a NULL value for the prev pointer indicates the start of the list.

3.2.1 Data Visualization



Taken from (Geeksforgeeks, 2023a)

3.2.2 Code Structure

```
template <typename T>
struct Doubly_ListNode {
    T val;
    Doubly_ListNode<T>* prev;
    Doubly_ListNode<T>* next;

    Doubly_ListNode() : val(T()), prev(nullptr), next(nullptr) {}
    explicit Doubly_ListNode(T value) : val(value), prev(nullptr), next(nullptr) {}
};
```

Easy Traversal

Doubly linked lists allow for an easier traversal as it can go both forward and backward.

Efficient deletion

Doubly linked lists offer an easier and efficient deletion process.

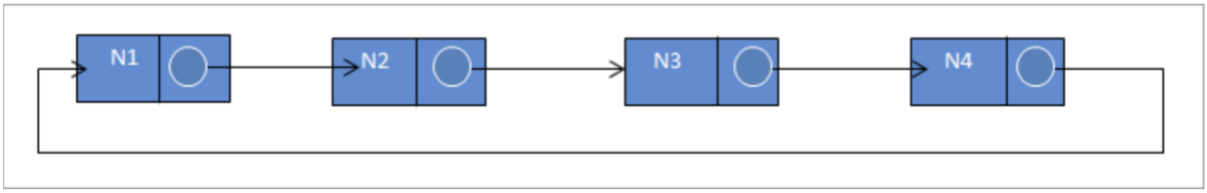
Quicker insertion

A new node can easily be inserted into the doubly linked list before a given node (Geeksforgeeks, 2023a).

3.3 Circular Linked List

A singly circular linked list is a variation of the singly linked list where the last node in the list points back to the first node, forming a circular structure. This means that the "next" pointer of the last node is not NULL but points to the head of the list.

3.3.1 Data Visualization



Taken from (SoftwareTestingHelp, 2023)

3.3.2 Code Structure

```
1  template <typename T>
2  class circular_linked_list {
3  private:
4      int length;
5      ListNode<T>* head;
6      ListNode<T>* tail;
7
8  public:
9      circular_linked_list() : length(0), head(nullptr), tail(nullptr) {}
```

Circular Behavior

The circular structure of the linked lists makes it easier to implement when compared to other linked lists as it is simple and easy to understand.

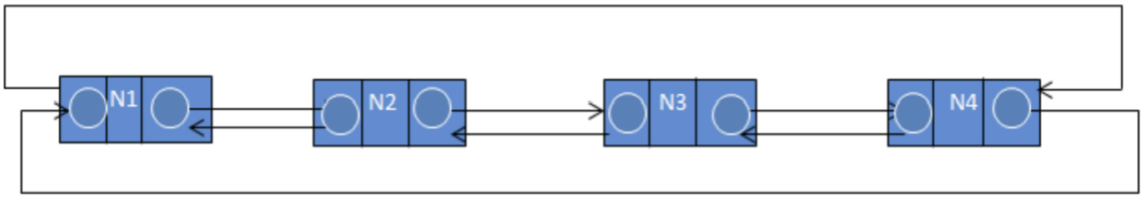
Memory Efficiency

Circular Linked Lists are more efficient when allocating memory as it does not need an extra pointer to store the end of the list (Shreyasnaphad, 2023a).

3.4 Doubly Circular Linked List

A doubly circular linked list is a variation of the doubly linked list where both the head and tail nodes are connected, forming a circular structure. Each node in the list contains two pointers: one points to the next node, and the other points to the previous node.

3.4.1 Data Visualization



Taken from (SoftwareTestingHelp, 2023)

3.4.2 Code Structure

```
1  template <typename T>
2  class doubly_circular_linked_list {
3  private:
4      int length;
5      Doubly_ListNode<T>* head;
6      Doubly_ListNode<T>* tail;
7
8  public:
9      doubly_circular_linked_list() : length(0), head(nullptr), tail(nullptr) {}
```

Efficient Traversal

Circular Doubly Linked List offers efficient traversal for both backward and forward directions when looking through the data stored.

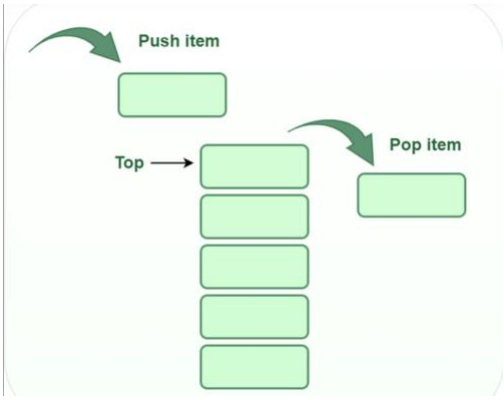
Implementing other data structures

Circular doubly linked lists can easily be used as a foundation to implement other data structures such as queues and hash tables showing its diverse applications (Shreyasnaphad, 2023b).

3.5 Stack

A stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It can be visualized as a stack of objects, where the last object added is the first one to be removed. The stack has two primary operations: "push" to add an element to the top of the stack, and "pop" to remove the topmost element. This was implemented using a linked list.

3.5.1 Data Visualization



Taken from (Geeksforgeeks, 2023b)

3.5.2 Code Structure

```
1  template <typename T>
2  class stack {
3  private:
4      int length;
5      ListNode<T>* top;
6  public:
7      stack() : length(0), top(nullptr) {}
```

Easy implementation

A stack data structure can be easily implemented into a program by using an array or a linked list. It also utilizes easy to understand operations such as pop and push.

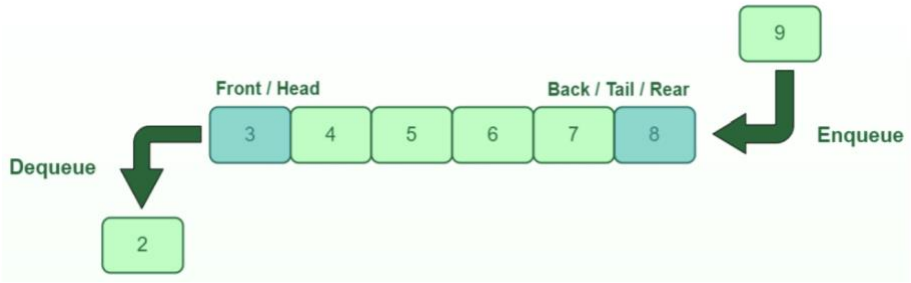
Enables undo/redo operations.

A stack helps in enabling undo / redo functions such as in text editors or graphic design tools. The nature of stack allows users to traverse the changes made easily (Aayushi2402, 2023).

3.6 Queue

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle. It can be visualized as a line of objects, where the first object added is the first one to be removed. The queue has two primary operations: "enqueue" to add an element to the back of the queue, and "dequeue" to remove an element from the front of the queue. Implemented using linked list.

3.6.1 Data Visualization



Taken from (Geeksforgeeks, 2023c)

3.6.2 Code Structure

```
1  template <typename T>
2  class queue {
3  private:
4      int length;
5      ListNode<T>* head;
6      ListNode<T>* tail;
7
8  public:
9      queue() : length(0), head(nullptr), tail(nullptr) {}
```

Efficient data management

A queue allows users to manage large amounts of data easily.

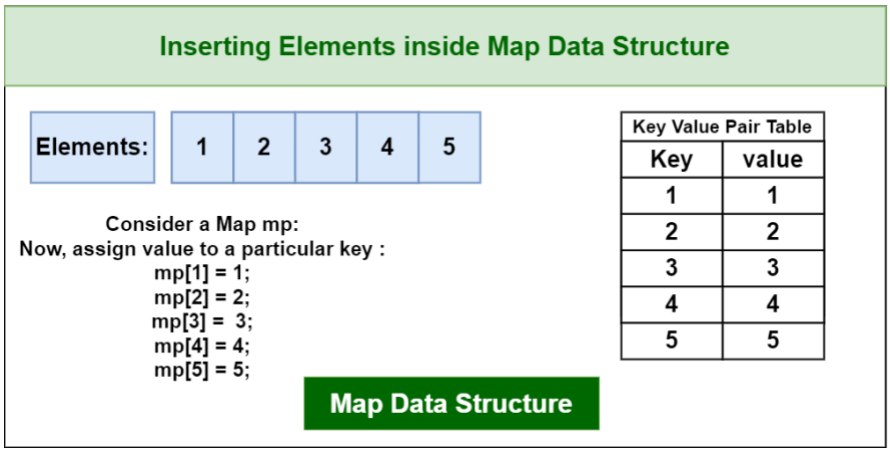
Easy insertion and Deletion methods

Insertion and deletion operations are easy to implement as it follows the First In/First Out (FIFO) rule (Shreyasnaphad, 2023c).

3.7 Map

Map is a type of data structure that can be utilised in C++. A map stores elements like a map and is an associative container. Each element stored in the map uses a key value and a mapped value. The key value is a unique value that is used to reference the mapped value while the mapped value is accessed by referencing or using the key value.

3.7.1 Data Visualization



Taken from (Itsadityash, 2023)

3.7.2 Code Structure

```
template<typename KeyType, typename ValueType>
class map {
private:
    vector<KeyType> keys;
    vector<ValueType> values;
```

Faster Data Access

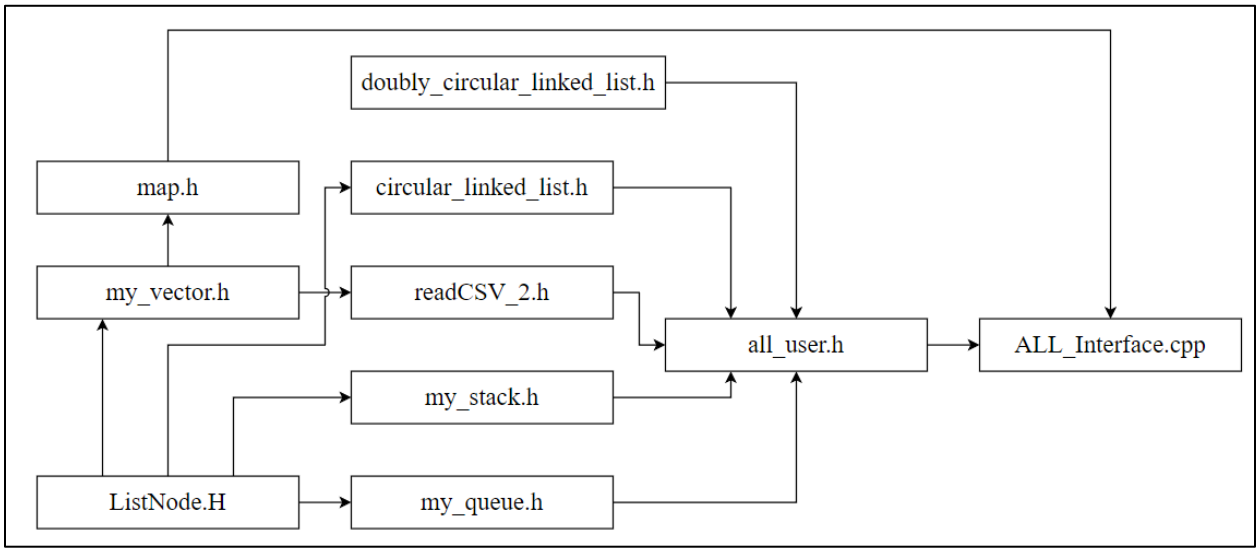
Using the key element in the map allows users to directly access the specified element which makes the process of accessing the data faster.

No Duplication

The key element in maps only stores unique values hence eliminating the possibility of redundancy for the data (Great Learning, 2021).

3.8 Data Structure Chart

This chart shows the implementation of the different data structures present within this program and how they are connected and utilized in each file.



3.9 Classes

3.9.1 Manager

The manager class consists of 5 variables which are all strings. The managerID is a unique identifier for each manager. The username is used for when managers log in to the system. The password is also used to access the log in. The email variable stores the email address of the manager. The status variable stores the status of the manager either active or inactive.

3.9.1.1 Class Visualization

3.9.1.2 Code Structure

```
class Manager{
    private:
        string managerID;
        string username;
        string password;
        string email;
        string status;
```

3.9.2 Admin

The admin class has 4 different variables. The first is the adminID which is the unique identifier for each admin. The username variable stores the username of the admin. The password variable stores the password used to log into the system for each admin. The email variable stores the email address of each admin.

3.9.2.1 Class Visualization

3.9.2.2 Code Structure

```
1  class Admin{
2      private:
3          string adminID;
4          string username;
5          string password;
6          string email;
```

3.9.3 Tenant

The tenant class has 9 variables. The userID variable is the unique identifier for each tenant. The name variable stores the name of the tenant. The password variable stores the password for the user account. The email variable stores the email address of the tenant. The LastActive variable stores the number of days the user has been active. The FavoriteProperty variable stores a list of the favorite property by the tenant using a queue. The RentingRequest variable stores a list of the rent requests made by the tenant using a circular doubly linked list. The RentingConfirm variables stores the confirmed renting requests for the tenant using a circular linked list. The RentingHistory variable stores the past renting history of the tenant using a stack.

3.9.3.1 Class visualization

3.9.3.2 Code Structure

```
1  class Tenant{
2      private:
3          string userID;
4          string name;
5          string password;
6          string email;
7          int LastActive;
8          queue<string> FavoriteProperty;
9          doubly_circular_linked_list<string> RentingRequest;
10         circular_linked_list<string> RentingConfirm;
11         stack<string> RentingHistory;
```

4 Search Algorithm

4.1 Binary Search

The binary search approach is an algorithm that demonstrates high efficiency by employing the "divide and conquer" principle to enhance the search process. Through recursive partitioning of the array into halves, the algorithm progressively refines the search space until the target element is located, or the list is reduced to a segment that does not contain the desired item.

Binary searches exploit the sorted nature of the array to achieve a time complexity of O(log n), significantly reducing search time. The fundamental steps of the binary search involve initializing an interval encompassing the entire array and iteratively narrowing down the interval based on a comparison between the search key value and the middle element of the current interval. This iterative process continues until the target value is found or the interval becomes empty (Terra, 2023).

4.1.1 Code

```
int Binary_Search(int n,int target)
        { //ID
            int left = 1;
            int right = numRows - 1;
            while (left <= right)
            {
                int mid = left + (right - left) / 2;
                if (extractInteger(warehouse.get(mid).get(n)) == target)
                {
                    return mid;
                }
                // If the target is greater, ignore the left half
                if (extractInteger(warehouse.get(mid).get(n)) < target)
                {
                    left = mid + 1;
                }
                // If the target is smaller, ignore the right half
                else
                {
                    right = mid - 1;
                }
            }
            return -1;
        }
```

Figure 1: Binary Search Code Snippet

Figure # is a snippet of the code for a binary search algorithm implementation. This algorithm efficiently locates a specific value, "target," within a sorted data structure, represented as a

"warehouse". The code starts by initializing two pointers, “left” to 1 and “right” to numRows-1, which represent the beginning and the end of the search space, respectively.

The algorithm then enters a while loop until the search space is exhausted. During each iteration, the algorithm calculates the middle index of the search range using the formula "mid = left + (right - left) / 2." This calculated "mid" index essentially represents the midpoint of the current search range. Afterward, the algorithm checks if the target value matches the value at the "mid" index in the "n" column. If there's a match, the algorithm returns the "mid" index as the result.

When the target value is greater than the value at the "mid" index, the algorithm infers that the desired value lies in the right half of the current search range. To focus on this right half, the "left" pointer is adjusted to "mid + 1."

On the flip side, if the target value is smaller than the value at the "mid" index, the algorithm deduces that the target is situated in the left half of the search range. This leads to the "right" pointer being updated to "mid - 1," ignoring the range's right half.

If the iteration completes without finding the target value, the algorithm concludes by returning "-1," indicating that the sought-after value is not present in the provided data structure.

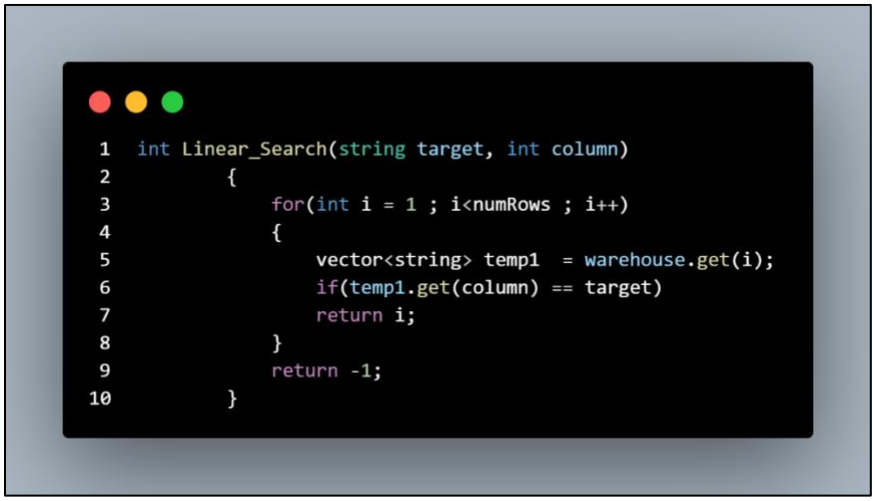
4.1.2 Implementation

In order to effectively harness the full potential of this dataset, the binary search algorithm is implemented for property searching. One of the most compelling reasons for utilizing this specific algorithm is that it has a binary time complexity of O log(n), where 'n' represents the size of the dataset. This ensures that the search time increases logarithmically with the dataset size, making binary search a highly efficient choice, especially since the dataset contains over 10,000 property entries. Moreover, utilizing binary search within the system requires sorting the dataset before being utilized.

4.2 Linear Search

The linear search algorithm is widely regarded as the most straightforward method for searching an element within a dataset. It examines each dataset component sequentially, starting from the beginning and continuing until the end, to find a match. The search is deemed successful upon locating the target element, and the algorithm terminates. However, without a match, the algorithm gracefully concludes its execution and provides an appropriate result. The inherent simplicity and efficiency of the linear search algorithm make it particularly valuable in two distinct scenarios. The linear search algorithm navigates the unordered array through its sequential and systematic approach, ensuring comprehensive coverage until the target element is discovered (S, 2023).

4.2.1 Code



```
1  int Linear_Search(string target, int column)
2          {
3              for(int i = 1 ; i<numRows ; i++)
4              {
5                  vector<string> temp1  = warehouse.get(i);
6                  if(temp1.get(column) == target)
7                  return i;
8              }
9              return -1;
10         }
```

Figure 2: Linear Search Code Snippet

Figure # is a snippet of the code for a linear search algorithm implementation. This algorithm finds a specific value, "target," within a 2D vector called "warehouse." The code iterates through each row of the vector to search for the target value in the specified column.

The function takes two parameters: a string "target" representing the element to be searched for and an integer "column" indicating the specific column in the 2D vector where the search will be performed.

The algorithm starts by initializing a loop that iterates over each row in the 2D vector "warehouse." The loop variable "i" represents the row index and starts from 1.

Within the loop, a temporary vector of strings, "temp1," is created and assigned the contents of the row at index "i" from the 2D vector "warehouse" using the "get" function.

Using an if statement, the function checks if the value at the specified "column" in the current row matches the "target" string. If the match is found, the function immediately returns the current row index "i," indicating the target's position in the 2D vector.

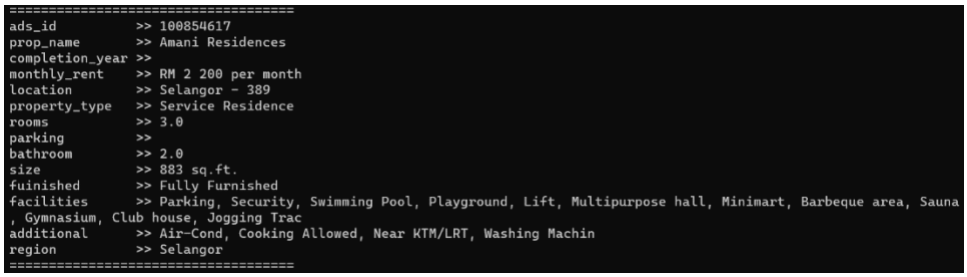
If the loop completes without finding the target element, the function returns -1, indicating that the target is not present in the 2D vector.

4.2.2 Implementation

To effectively harness the full potential of this dataset, the linear search algorithm is implemented for property searching. One of the most compelling reasons for utilizing this specific algorithm is that it has a linear time complexity of O(n), where 'n' represents the size of the dataset. This ensures that the search time increases proportionally with the dataset size. Moreover, utilizing linear search within the system makes it analogously simpler and more straightforward to implement than another search algorithm. Furthermore, the algorithm does not impose any prerequisite requirements to be implemented, further contributing to its suitability for property searching.

4.3 Algorithm comparison

In order to compare these two search algorithms, the same 'ads_id' is used for easier comparison; the property information is listed below in Figure #.



Search Algorithm	Binary Search	Linear Search
Execution time	0.00 s	0.03 s
Difference	0.03 s	

The table above presents a comparison between two implemented search algorithms' execution times. Which it is apparent that Binary Search has a faster execution time in comparison to Linear Search.

Search Algorithm	Binary Search	Linear Search
Space Complexity	O(1)	O(1)
Time Complexity (Best Case)	O(1)	O(1)
Time Complexity (Worst Case)	O(log n)	O(n)
Time Complexity (Average Case)	O(log n)	O(n/2)

According to it, Binary Search exhibits faster performance due to its logarithmic nature, requiring less time to locate the target. This contrasts with Linear Search, which sequentially checks each element, resulting in comparatively slower execution.

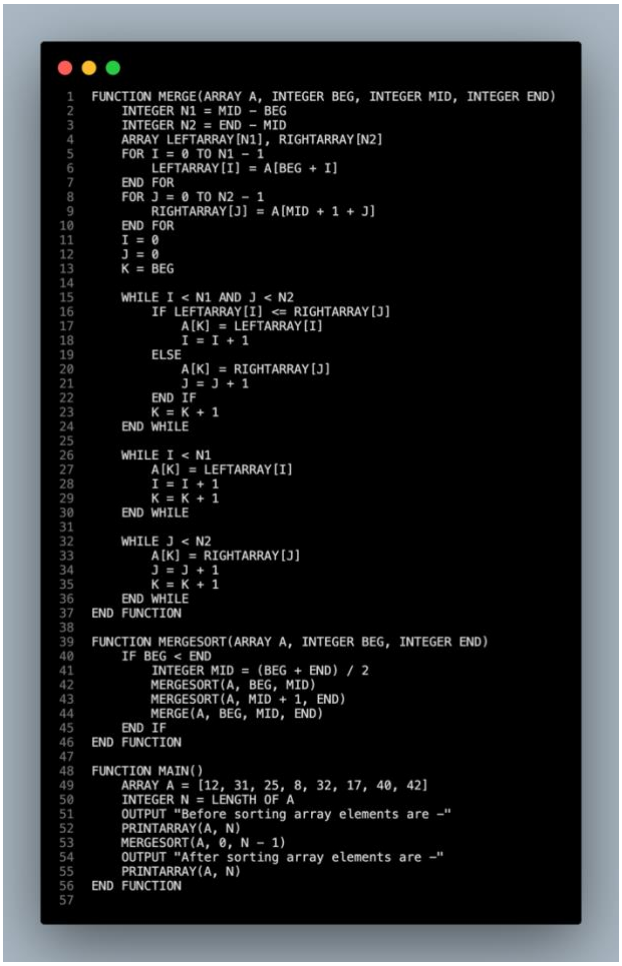
5 Sorting Algorithm

5.1 Merge Sort

Merge sort is a comparison-based algorithm that splits the targeted list into a few sublists and recursively compares each element's value in the sublists. Then, merge the result sublists into one. One of the advantages of using merge sort is that it makes sure the output and the input are the same and will not be changed during the sorting. The average, best-case performance and worst-case performance for merge sort are O(n log n). The steps of merge sort are:

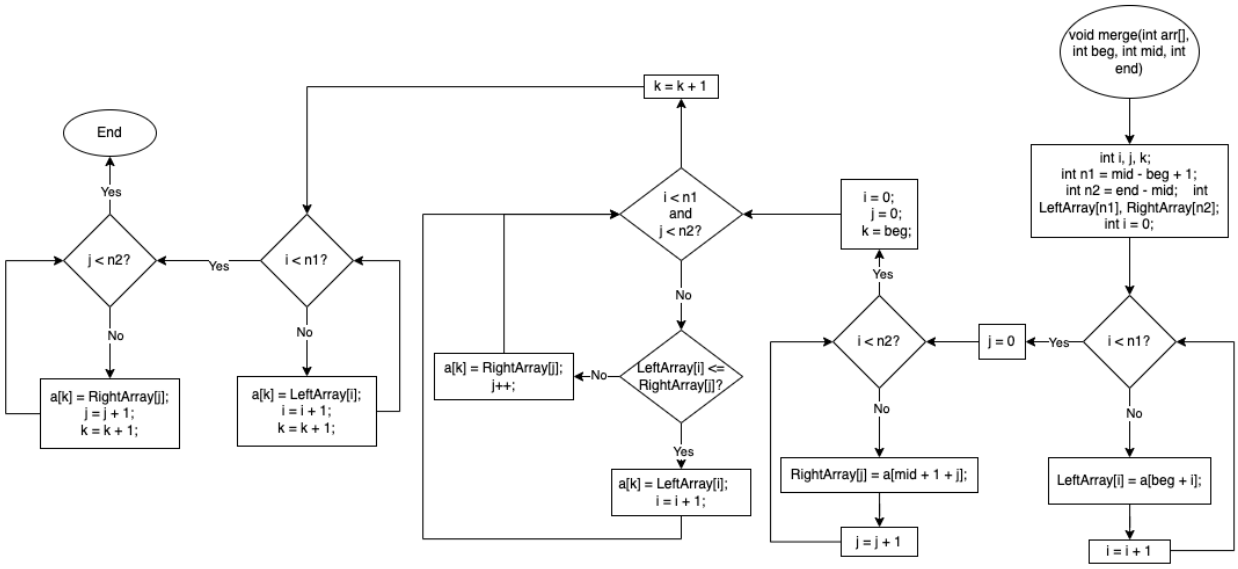
1. Divide the array into two equal-length arrays.
2. Keep dividing the subarrays until the smallest unit.
3. Compare and sort all of the final subarrays.
4. Merge based on their values.

5.1.1 Pseudocode

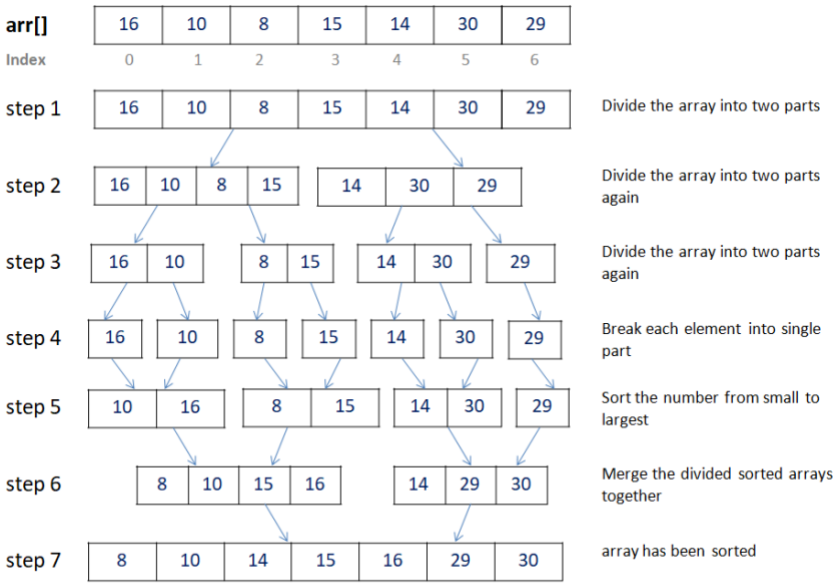


```
FUNCTION MERGE(ARRAY A, INTEGER BEG, INTEGER MID, INTEGER END)
    INTEGER N1 = MID - BEG
    INTEGER N2 = END - MID
    ARRAY LEFTARRAY[N1], RIGHTARRAY[N2]
    FOR I = 0 TO N1 - 1
        LEFTARRAY[I] = A[BEG + I]
    END FOR
    FOR J = 0 TO N2 - 1
        RIGHTARRAY[J] = A[MID + 1 + J]
    END FOR
    I = 0
    J = 0
    K = BEG

    WHILE I < N1 AND J < N2
        IF LEFTARRAY[I] <= RIGHTARRAY[J]
            A[K] = LEFTARRAY[I]
            I = I + 1
        ELSE
            A[K] = RIGHTARRAY[J]
            J = J + 1
        END IF
        K = K + 1
    END WHILE

    WHILE I < N1
        A[K] = LEFTARRAY[I]
        I = I + 1
        K = K + 1
    END WHILE

    WHILE J < N2
        A[K] = RIGHTARRAY[J]
        J = J + 1
        K = K + 1
    END WHILE
END FUNCTION

FUNCTION MERGESORT(ARRAY A, INTEGER BEG, INTEGER END)
    IF BEG < END
        INTEGER MID = (BEG + END) / 2
        MERGESORT(A, BEG, MID)
        MERGESORT(A, MID + 1, END)
        MERGE(A, BEG, MID, END)
    END IF
END FUNCTION

FUNCTION MAIN()
    ARRAY A = [12, 31, 25, 8, 32, 17, 40, 42]
    INTEGER N = LENGTH OF A
    OUTPUT "Before sorting array elements are -"
    PRINTARRAY(A, N)
    MERGESORT(A, 0, N - 1)
    OUTPUT "After sorting array elements are -"
    PRINTARRAY(A, N)
END FUNCTION
```

5.1.2 Flowchart



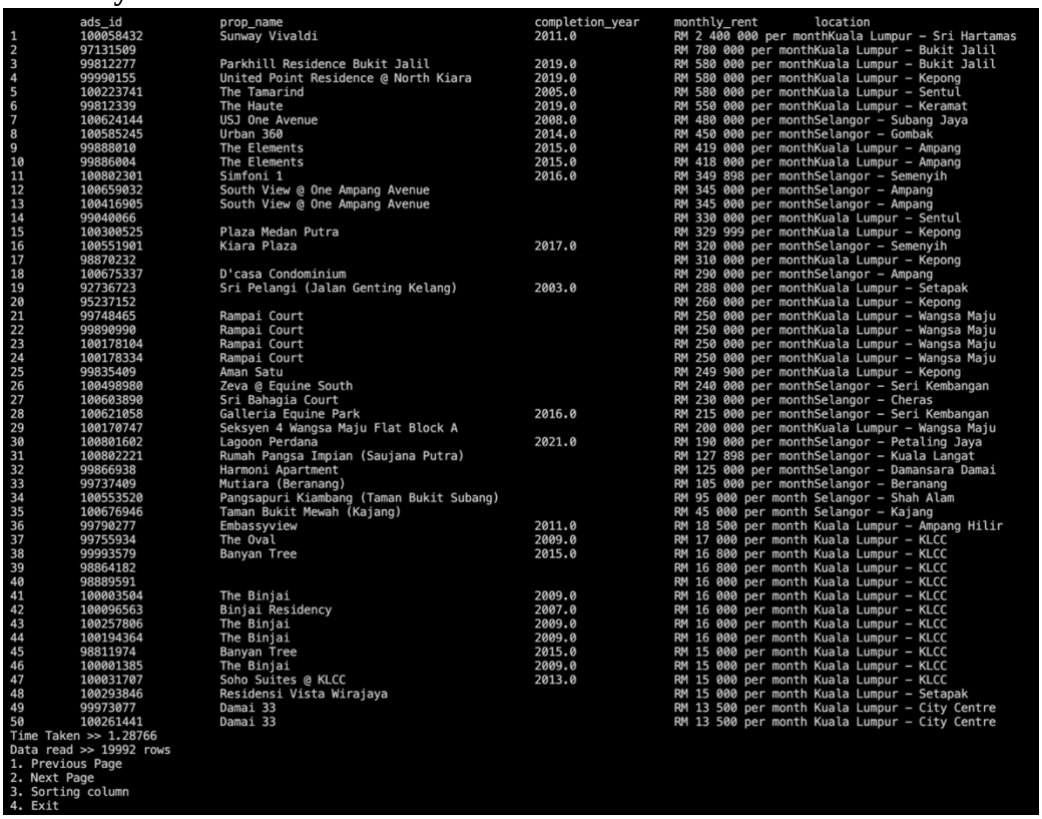
5.1.3 Visualization



5.1.4 Implementation

```
void s_merge(int low, int mid, int high, int column) {
            int n1 = mid - low + 1;
            int n2 = high - mid;
            vector<vector<string>> left;
            vector<vector<string>> right;
            for (int i = 0; i < n1; i++)
                left.push_back(warehouse[low + i]);
            for (int j = 0; j < n2; j++)
                right.push_back(warehouse[mid + 1 + j ]);
            int i = 0;
            int j = 0;
            int k = low;
            while (i < n1 && j < n2) {
                vector<string> temp1 = left.get(i);
                vector<string> temp2 = right.get(j);
                if (temp1.get(column) > temp2.get(column)) { //version1 for string
                    set(k,temp1);
                    i++;
                } else {
                    set(k,temp2);
                    j++;
                }
                k++;
            }
            while (i < n1) {
                set(k,left.get(i));
                i++;
                k++;
            }
            while (j < n2) {
                set(k,right.get(j));
                j++;
                k++;
            }
        }
void s_mergeSort(int low, int high, int column) {
            if (low < high) {
                int mid = low + (high - low) / 2;
                s_mergeSort(low, mid, column);
                s_mergeSort(mid + 1, high, column);
                s_merge(low, mid, high, column);
            }
        }
```

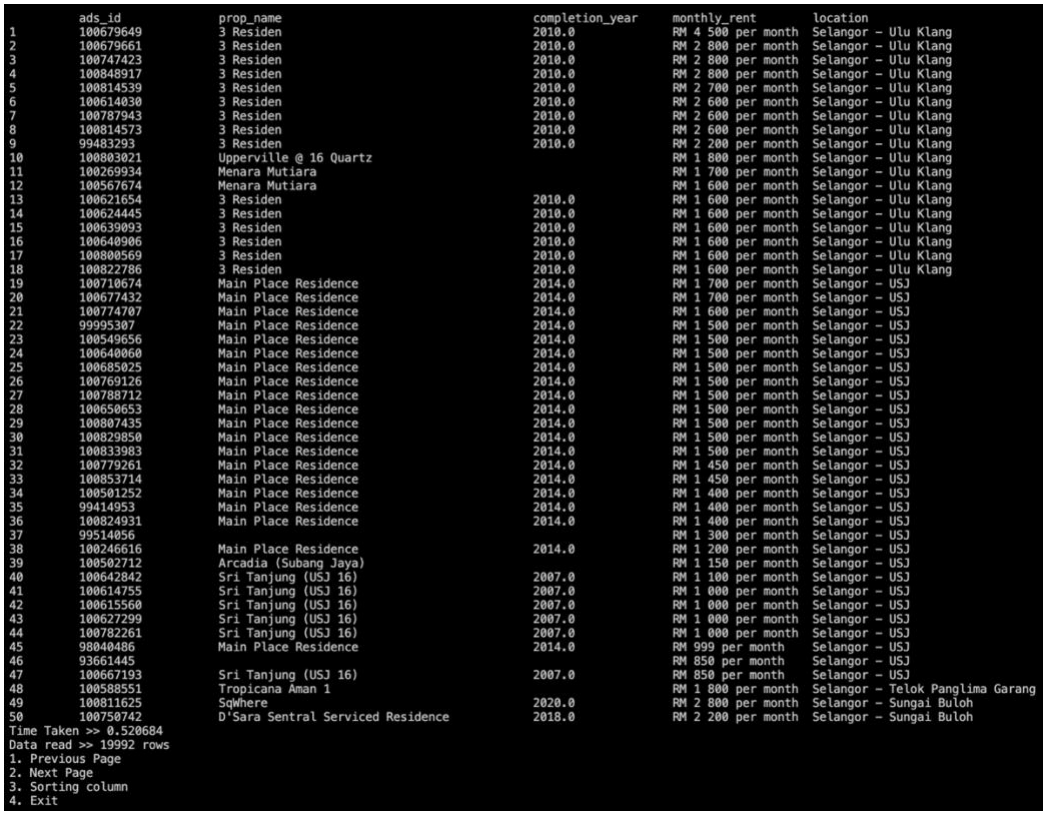
5.1.4.1 Monthly rent



	ads_id	prop_name	completion_year	monthly_rent	location
1	100058432	Sunway Vivaldi	2011.0	RM 2 400 000	per month Kuala Lumpur – Sri Hartamas
2	97131509			RM 780 000	per month Kuala Lumpur – Bukit Jalil
3	99812277	Parkhill Residence Bukit Jalil	2019.0	RM 580 000	per month Kuala Lumpur – Bukit Jalil
4	99990155	United Point Residence @ North Kiara	2019.0	RM 580 000	per month Kuala Lumpur – Kepong
5	100223741	The Tamarind	2005.0	RM 580 000	per month Kuala Lumpur – Sentul
6	99812339	The Haute	2019.0	RM 550 000	per month Kuala Lumpur – Keramat
7	100624144	USJ One Avenue	2008.0	RM 480 000	per month Selangor – Subang Jaya
8	100585245	Urban 360	2014.0	RM 450 000	per month Selangor – Gombak
9	99888010	The Elements	2015.0	RM 419 000	per month Kuala Lumpur – Ampang
10	99886004	The Elements	2015.0	RM 418 000	per month Kuala Lumpur – Ampang
11	100802301	Simfoni 1	2016.0	RM 349 898	per month Selangor – Semenyih
12	100659032	South View @ One Ampang Avenue		RM 345 000	per month Selangor – Ampang
13	100416905	South View @ One Ampang Avenue		RM 345 000	per month Selangor – Ampang
14	99040066			RM 330 000	per month Kuala Lumpur – Sentul
15	100300525	Plaza Medan Putra		RM 329 999	per month Kuala Lumpur – Kepong
16	100551901	Kiara Plaza	2017.0	RM 320 000	per month Selangor – Semenyih
17	98870232			RM 310 000	per month Kuala Lumpur – Kepong
18	100675337	D'casa Condominium		RM 290 000	per month Selangor – Ampang
19	92736723	Sri Pelangi (Jalan Genting Kelang)	2003.0	RM 288 000	per month Kuala Lumpur – Setapak
20	95237152			RM 260 000	per month Kuala Lumpur – Kepong
21	99748465	Rampai Court		RM 250 000	per month Kuala Lumpur – Wangsa Maju
22	99890990	Rampai Court		RM 250 000	per month Kuala Lumpur – Wangsa Maju
23	100178104	Rampai Court		RM 250 000	per month Kuala Lumpur – Wangsa Maju
24	100178334	Rampai Court		RM 250 000	per month Kuala Lumpur – Wangsa Maju
25	99835409	Aman Satu		RM 249 900	per month Kuala Lumpur – Kepong
26	100498980	Zeva @ Equine South		RM 240 000	per month Selangor – Seri Kembangan
27	100603890	Sri Bahagia Court		RM 230 000	per month Selangor – Cheras
28	100621058	Galleria Equine Park	2016.0	RM 215 000	per month Selangor – Seri Kembangan
29	100170747	Seksyen 4 Wangsa Maju Flat Block A		RM 200 000	per month Kuala Lumpur – Wangsa Maju
30	100801602	Lagoon Perdana	2021.0	RM 190 000	per month Selangor – Petaling Jaya
31	100802221	Rumah Pangsa Impian (Saujana Putra)		RM 127 898	per month Selangor – Kuala Langat
32	99866938	Harmoni Apartment		RM 125 000	per month Selangor – Damansara Damai
33	99737409	Mutiara (Beranang)		RM 105 000	per month Selangor – Beranang
34	100553520	Pangsapuri Kiambang (Taman Bukit Subang)		RM 95 000	per month Selangor – Shah Alam
35	100676946	Taman Bukit Mewah (Kajang)		RM 45 000	per month Selangor – Kajang
36	99790277	Embassyview	2011.0	RM 18 500	per month Kuala Lumpur – Ampang Hilir
37	99755934	The Oval	2009.0	RM 17 000	per month Kuala Lumpur – KLCC
38	99993579	Banyan Tree	2015.0	RM 16 800	per month Kuala Lumpur – KLCC
39	98864182			RM 16 800	per month Kuala Lumpur – KLCC
40	98889591			RM 16 000	per month Kuala Lumpur – KLCC
41	100003504	The Binjai	2009.0	RM 16 000	per month Kuala Lumpur – KLCC
42	100096563	Binjai Residency	2007.0	RM 16 000	per month Kuala Lumpur – KLCC
43	100257806	The Binjai	2009.0	RM 16 000	per month Kuala Lumpur – KLCC
44	100194364	The Binjai	2009.0	RM 16 000	per month Kuala Lumpur – KLCC
45	98811974	Banyan Tree	2015.0	RM 15 000	per month Kuala Lumpur – KLCC
46	100001385	The Binjai	2009.0	RM 15 000	per month Kuala Lumpur – KLCC
47	100031707	Soho Suites @ KLCC	2013.0	RM 15 000	per month Kuala Lumpur – KLCC
48	100293846	Residensi Vista Wirajaya		RM 15 000	per month Kuala Lumpur – Setapak
49	99973077	Damai 33		RM 13 500	per month Kuala Lumpur – City Centre
50	100261441	Damai 33		RM 13 500	per month Kuala Lumpur – City Centre

Time Taken >> 1.28766
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit

5.1.4.2 Location



	ads_id	prop_name	completion_year	monthly_rent	location
1	100679649	3 Residen	2010.0	RM 4 500 per month	Selangor – Ulu Klang
2	100679661	3 Residen	2010.0	RM 2 800 per month	Selangor – Ulu Klang
3	100747423	3 Residen	2010.0	RM 2 800 per month	Selangor – Ulu Klang
4	100848917	3 Residen	2010.0	RM 2 800 per month	Selangor – Ulu Klang
5	100814539	3 Residen	2010.0	RM 2 700 per month	Selangor – Ulu Klang
6	100614030	3 Residen	2010.0	RM 2 600 per month	Selangor – Ulu Klang
7	100787943	3 Residen	2010.0	RM 2 600 per month	Selangor – Ulu Klang
8	100814573	3 Residen	2010.0	RM 2 600 per month	Selangor – Ulu Klang
9	99483293	3 Residen	2010.0	RM 2 200 per month	Selangor – Ulu Klang
10	100803021	Upperville @ 16 Quartz		RM 1 800 per month	Selangor – Ulu Klang
11	100269934	Menara Mutiara		RM 1 700 per month	Selangor – Ulu Klang
12	100567674	Menara Mutiara		RM 1 600 per month	Selangor – Ulu Klang
13	100621654	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
14	100624445	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
15	100639093	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
16	100640906	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
17	100800569	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
18	100822786	3 Residen	2010.0	RM 1 600 per month	Selangor – Ulu Klang
19	100710674	Main Place Residence	2014.0	RM 1 700 per month	Selangor – USJ
20	100677432	Main Place Residence	2014.0	RM 1 700 per month	Selangor – USJ
21	100774707	Main Place Residence	2014.0	RM 1 600 per month	Selangor – USJ
22	99995307	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
23	100549656	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
24	100640060	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
25	100685025	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
26	100769126	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
27	100788712	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
28	100650653	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
29	100807435	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
30	100829850	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
31	100833983	Main Place Residence	2014.0	RM 1 500 per month	Selangor – USJ
32	100779261	Main Place Residence	2014.0	RM 1 450 per month	Selangor – USJ
33	100853714	Main Place Residence	2014.0	RM 1 450 per month	Selangor – USJ
34	100501252	Main Place Residence	2014.0	RM 1 400 per month	Selangor – USJ
35	99414953	Main Place Residence	2014.0	RM 1 400 per month	Selangor – USJ
36	100824931	Main Place Residence	2014.0	RM 1 400 per month	Selangor – USJ
37	99514056			RM 1 300 per month	Selangor – USJ
38	100246616	Main Place Residence	2014.0	RM 1 200 per month	Selangor – USJ
39	100502712	Arcadia (Subang Jaya)		RM 1 150 per month	Selangor – USJ
40	100642842	Sri Tanjung (USJ 16)	2007.0	RM 1 100 per month	Selangor – USJ
41	100614755	Sri Tanjung (USJ 16)	2007.0	RM 1 000 per month	Selangor – USJ
42	100615560	Sri Tanjung (USJ 16)	2007.0	RM 1 000 per month	Selangor – USJ
43	100627299	Sri Tanjung (USJ 16)	2007.0	RM 1 000 per month	Selangor – USJ
44	100782261	Sri Tanjung (USJ 16)	2007.0	RM 1 000 per month	Selangor – USJ
45	98040486	Main Place Residence	2014.0	RM 999 per month	Selangor – USJ
46	93661445			RM 850 per month	Selangor – USJ
47	100667193	Sri Tanjung (USJ 16)	2007.0	RM 850 per month	Selangor – USJ
48	100588551	Tropicana Aman 1		RM 1 800 per month	Selangor – Telok Panglima Garang
49	100811625	SqWhere	2020.0	RM 2 800 per month	Selangor – Sungai Buloh
50	100750742	D'Sara Sentral Serviced Residence	2018.0	RM 2 200 per month	Selangor – Sungai Buloh

Time Taken >> 0.520684
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit

5.1.4.3 Size as per square feet

	ads_id	prop_name	size	monthly_rent	location
1	100255897	Angkasa Condominiums	99999999 sq.ft.	RM 110 per month	Kuala Lumpur - Cheras
2	100213068	Zamrud Apartment	48010 sq.ft.	RM 1 400 per month	Kuala Lumpur - Old Klang Road
3	99994083	Sri Wangsaria	15000 sq.ft.	RM 3 300 per month	Kuala Lumpur - Bangsar
4	99659804	Mizumi Residences	9001 sq.ft.	RM 1 800 per month	Kuala Lumpur - Kepong
5	99706946	The Lumayan Apartment	8300 sq.ft.	RM 1 400 per month	Kuala Lumpur - Cheras
6	99755934	The Oval	7800 sq.ft.	RM 17 000 per month	Kuala Lumpur - KLCC
7	99790277	Embassyview	7506 sq.ft.	RM 18 500 per month	Kuala Lumpur - Ampang Hilir
8	100564173	Eclipse Residence @ Pan'gaea	6400 sq.ft.	RM 1 780 per month	Selangor - Cyberjaya
9	98976840		6067 sq.ft.	RM 12 000 per month	Kuala Lumpur - Ampang Hilir
10	98976763		5000 sq.ft.	RM 8 000 per month	Kuala Lumpur - Ampang Hilir
11	100261441	Damai 33	5000 sq.ft.	RM 13 500 per month	Kuala Lumpur - City Centre
12	99973077	Damai 33	5000 sq.ft.	RM 13 500 per month	Kuala Lumpur - City Centre
13	100612908	Subang Olives Residence	4915 sq.ft.	RM 4 848 per month	Selangor - Subang Jaya
14	100612892	Subang Olives Residence	4915 sq.ft.	RM 6 888 per month	Selangor - Subang Jaya
15	95897047		4356 sq.ft.	RM 9 000 per month	Kuala Lumpur - Ampang Hilir
16	99794408	325 Persiaran Ritchie	4300 sq.ft.	RM 12 000 per month	Kuala Lumpur - Ampang Hilir
17	100078345	325 Persiaran Ritchie	4300 sq.ft.	RM 12 000 per month	Kuala Lumpur - Ampang Hilir
18	99740698	Brunsfield EmbassyView	4098 sq.ft.	RM 12 500 per month	Kuala Lumpur - Cheras
19	100541053	Imperial Hatamas	4012 sq.ft.	RM 3 980 per month	Selangor - Cheras
20	100648736	Imperial Residence Cheras	4000 sq.ft.	RM 4 000 per month	Selangor - Cheras
21	88274040	Anjali @ North Kiara	3713 sq.ft.	RM 5 500 per month	Kuala Lumpur - Segambut
22	100611344	Villa Lagenda	3700 sq.ft.	RM 2 999 per month	Selangor - Selayang
23	99732058	Impiana on The Waterfront	3650 sq.ft.	RM 8 500 per month	Kuala Lumpur - Ampang
24	100225460	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
25	100226511	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
26	100206265	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
27	100159124	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
28	100118818	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
29	99793908	Hartamas Regency II	3488 sq.ft.	RM 5 800 per month	Kuala Lumpur - Mont Kiara
30	99914450	Sky Vista Residency	3400 sq.ft.	RM 5 000 per month	Kuala Lumpur - Cheras
31	98873901		3400 sq.ft.	RM 10 000 per month	Kuala Lumpur - Damansara Heights
32	100576263	Tropicana Grande	3340 sq.ft.	RM 9 800 per month	Selangor - Petaling Jaya
33	77073260		3337 sq.ft.	RM 7 900 per month	Kuala Lumpur - KLCC
34	100576251	Tropicana Grande	3320 sq.ft.	RM 9 500 per month	Selangor - Petaling Jaya
35	100169369	Suria Stonor	3283 sq.ft.	RM 7 000 per month	Kuala Lumpur - KLCC
36	100176173	Vila Vista	3223 sq.ft.	RM 5 800 per month	Kuala Lumpur - Cheras
37	99914870	Vila Vista	3223 sq.ft.	RM 5 800 per month	Kuala Lumpur - Cheras
38	100001385	The Binjai	3218 sq.ft.	RM 15 000 per month	Kuala Lumpur - KLCC
39	100194364	The Binjai	3218 sq.ft.	RM 16 000 per month	Kuala Lumpur - KLCC
40	100257806	The Binjai	3218 sq.ft.	RM 16 000 per month	Kuala Lumpur - KLCC
41	100096563	Binjai Residency	3218 sq.ft.	RM 16 000 per month	Kuala Lumpur - KLCC
42	100003504	The Binjai	3218 sq.ft.	RM 16 000 per month	Kuala Lumpur - KLCC
43	98889591		3218 sq.ft.	RM 16 000 per month	Kuala Lumpur - KLCC
44	100052711	Ceriaan Kiara	3174 sq.ft.	RM 3 300 per month	Kuala Lumpur - Mont Kiara
45	100214760	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara
46	100138802	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara
47	100060069	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara
48	100029073	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara
49	99967981	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara
50	99845277	Damansara Foresta	3150 sq.ft.	RM 4 000 per month	Kuala Lumpur - Sri Damansara

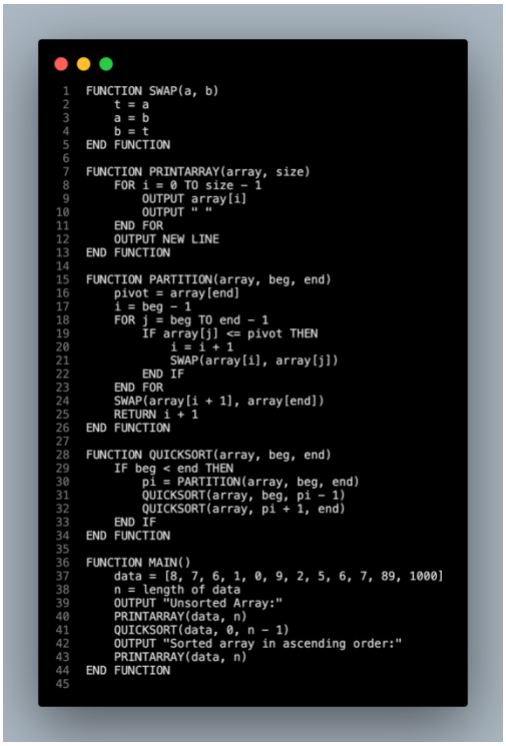
```
Time Taken >> 1.2296
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit
```

5.2 Quick Sort

Quick sort is a divide-and-conquer algorithm. It means quickly dividing the list into a few sublists, sorting each one by one, and merging it together. Quick sort's worst-case performance is $O(n^2)$ and the average, best-case performance is O(n log n). Here are the steps for a quick sort:

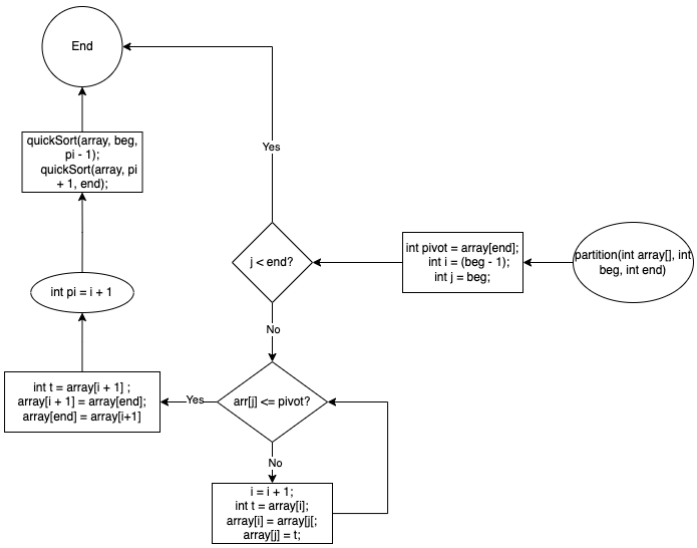
1. Select one of the elements as a pivot (can be a left-most or right-most element)
2. Compare each of the elements with the pivot:
    a. If the element > pivot: produce a new pointer that points to the following value
    b. If the element < pivot: swap it with the previous pointer (if any)
3. Swap the pivot with the last next pointer.
4. Divide the list with all the next pointers and recursively compare like above.

5.2.1 Pseudocode

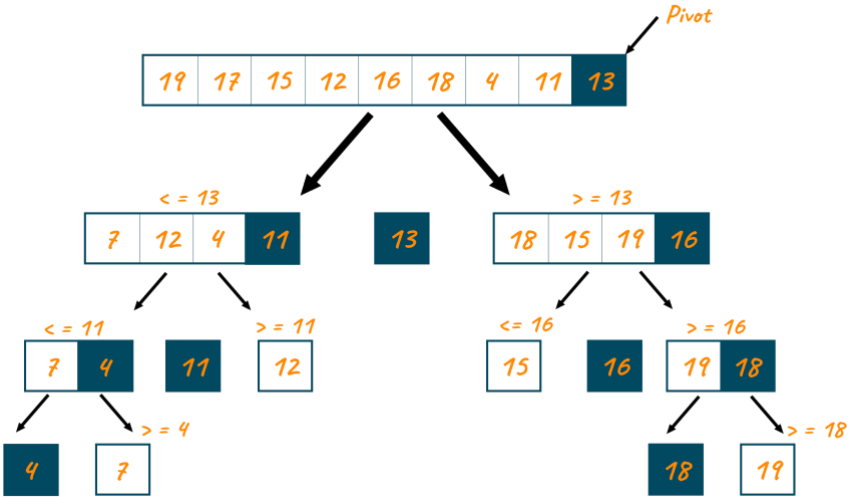


```
FUNCTION SWAP(a, b)
    t = a
    a = b
    b = t
END FUNCTION

FUNCTION PRINTARRAY(array, size)
    FOR i = 0 TO size - 1
        OUTPUT array[i]
        OUTPUT " "
    END FOR
    OUTPUT NEW LINE
END FUNCTION

FUNCTION PARTITION(array, beg, end)
    pivot = array[end]
    i = beg - 1
    FOR j = beg TO end - 1
        IF array[j] <= pivot THEN
            i = i + 1
            SWAP(array[i], array[j])
        END IF
    END FOR
    SWAP(array[i + 1], array[end])
    RETURN i + 1
END FUNCTION

FUNCTION QUICKSORT(array, beg, end)
    IF beg < end THEN
        pi = PARTITION(array, beg, end)
        QUICKSORT(array, beg, pi - 1)
        QUICKSORT(array, pi + 1, end)
    END IF
END FUNCTION

FUNCTION MAIN()
    data = [8, 7, 6, 1, 0, 9, 2, 5, 6, 7, 89, 1000]
    n = length of data
    OUTPUT "Unsorted Array:"
    PRINTARRAY(data, n)
    QUICKSORT(data, 0, n - 1)
    OUTPUT "Sorted array in ascending order:"
    PRINTARRAY(data, n)
END FUNCTION

```

5.2.2 Flowchart



5.2.3 Visualization



5.2.4 Implementation

```
int s_partition( int low, int high, int column){
            vector<string> temp1 = warehouse[high];
            string pivot = temp1.get(column);
            int i = low-1;
            for (int j = low; j <= high - 1; j++) {
                vector<string> temp2 = warehouse[j];
                if (temp2.get(column) >= pivot) {
                    i++;
                    swap(i,j);
                }
            }
            swap(i+1,high);
            return i+1;
        }
void s_quickSort(int low, int high, int column ){
            if(low < high){
                int pivotIndex = s_partition(low, high,column);
                s_quickSort(low, pivotIndex-1, column);
                s_quickSort(pivotIndex+1, high, column);
            }
        }
```

5.2.4.1 Monthly rent

	ads_id	prop_name	completion_year	monthly_rent	location
1	100058432	Sunway Vivaldi	2011.0	RM 2 400 000 per month	Kuala Lumpur - Sri Hartamas
2	97131509			RM 780 000 per month	Kuala Lumpur - Bukit Jalil
3	99812277	Parkhill Residence Bukit Jalil	2019.0	RM 580 000 per month	Kuala Lumpur - Bukit Jalil
4	100223741	The Tamarind	2005.0	RM 580 000 per month	Kuala Lumpur - Sentul
5	99990155	United Point Residence @ North Kiara	2019.0	RM 580 000 per month	Kuala Lumpur - Kepong
6	99812339	The Haute	2019.0	RM 550 000 per month	Kuala Lumpur - Keramat
7	100624144	USJ One Avenue	2008.0	RM 480 000 per month	Selangor - Subang Jaya
8	100585245	Urban 360	2014.0	RM 450 000 per month	Selangor - Gombak
9	99888010	The Elements	2015.0	RM 419 000 per month	Kuala Lumpur - Ampang
10	99886004	The Elements	2015.0	RM 418 000 per month	Kuala Lumpur - Ampang
11	100802301	Simfoni 1	2016.0	RM 349 898 per month	Selangor - Semenyih
12	100416905	South View @ One Ampang Avenue		RM 345 000 per month	Selangor - Ampang
13	100659032	South View @ One Ampang Avenue		RM 345 000 per month	Selangor - Ampang
14	99040066			RM 330 000 per month	Kuala Lumpur - Sentul
15	100300525	Plaza Medan Putra		RM 329 999 per month	Kuala Lumpur - Kepong
16	100551901	Kiara Plaza	2017.0	RM 320 000 per month	Selangor - Semenyih
17	98870232			RM 310 000 per month	Kuala Lumpur - Kepong
18	100675337	D'casa Condominium		RM 290 000 per month	Selangor - Ampang
19	92736723	Sri Pelangi (Jalan Genting Kelang)	2003.0	RM 288 000 per month	Kuala Lumpur - Setapak
20	95237152			RM 260 000 per month	Kuala Lumpur - Kepong
21	100178334	Rampai Court		RM 250 000 per month	Kuala Lumpur - Wangsa Maju
22	100178104	Rampai Court		RM 250 000 per month	Kuala Lumpur - Wangsa Maju
23	99890990	Rampai Court		RM 250 000 per month	Kuala Lumpur - Wangsa Maju
24	99748465	Rampai Court		RM 250 000 per month	Kuala Lumpur - Wangsa Maju
25	99835409	Aman Satu		RM 249 900 per month	Kuala Lumpur - Kepong
26	100498980	Zeva @ Equine South		RM 240 000 per month	Selangor - Seri Kembangan
27	100603890	Sri Bahagia Court		RM 230 000 per month	Selangor - Cheras
28	100621058	Galleria Equine Park	2016.0	RM 215 000 per month	Selangor - Seri Kembangan
29	100170747	Seksyen 4 Wangsa Maju Flat Block A		RM 200 000 per month	Kuala Lumpur - Wangsa Maju
30	100801602	Lagoon Perdana	2021.0	RM 190 000 per month	Selangor - Petaling Jaya
31	100802221	Rumah Pangsa Impian (Saujana Putra)		RM 127 898 per month	Selangor - Kuala Langat
32	99866938	Harmoni Apartment		RM 125 000 per month	Selangor - Damansara Damai
33	99737409	Mutiara (Beranang)		RM 105 000 per month	Selangor - Beranang
34	100553520	Pangsapuri Kiambang (Taman Bukit Subang)		RM 95 000 per month	Selangor - Shah Alam
35	100676946	Taman Bukit Mewah (Kajang)		RM 45 000 per month	Selangor - Kajang
36	99790277	Embassyview	2011.0	RM 18 500 per month	Kuala Lumpur - Ampang Hilir
37	99755934	The Oval	2009.0	RM 17 000 per month	Kuala Lumpur - KLCC
38	99993579	Banyan Tree	2015.0	RM 16 800 per month	Kuala Lumpur - KLCC
39	98864182			RM 16 800 per month	Kuala Lumpur - KLCC
40	98889591			RM 16 000 per month	Kuala Lumpur - KLCC
41	100257806	The Binjai	2009.0	RM 16 000 per month	Kuala Lumpur - KLCC
42	100003504	The Binjai	2009.0	RM 16 000 per month	Kuala Lumpur - KLCC
43	100096563	Binjai Residency	2007.0	RM 16 000 per month	Kuala Lumpur - KLCC
44	100194364	The Binjai	2009.0	RM 16 000 per month	Kuala Lumpur - KLCC
45	98811974	Banyan Tree	2015.0	RM 15 000 per month	Kuala Lumpur - KLCC
46	100293846	Residensi Vista Wirajaya		RM 15 000 per month	Kuala Lumpur - Setapak
47	100001385	The Binjai	2009.0	RM 15 000 per month	Kuala Lumpur - KLCC
48	100031707	Soho Suites @ KLCC	2013.0	RM 15 000 per month	Kuala Lumpur - KLCC
49	100261441	Damai 33		RM 13 500 per month	Kuala Lumpur - City Centre
50	99973077	Damai 33		RM 13 500 per month	Kuala Lumpur - City Centre

```
Time Taken >> 5.21577
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit
```

5.2.4.2 Location

```
        ads_id          prop_name                       completion_year     monthly_rent        location
1       100679649       3 Residen                       2010.0              RM 4 500 per month  Selangor - Ulu Klang
2       100679661       3 Residen                       2010.0              RM 2 800 per month  Selangor - Ulu Klang
3       100848917       3 Residen                       2010.0              RM 2 800 per month  Selangor - Ulu Klang
4       100747423       3 Residen                       2010.0              RM 2 800 per month  Selangor - Ulu Klang
5       100814539       3 Residen                       2010.0              RM 2 700 per month  Selangor - Ulu Klang
6       100814573       3 Residen                       2010.0              RM 2 600 per month  Selangor - Ulu Klang
7       100787943       3 Residen                       2010.0              RM 2 600 per month  Selangor - Ulu Klang
8       100614030       3 Residen                       2010.0              RM 2 600 per month  Selangor - Ulu Klang
9       99483293        3 Residen                       2010.0              RM 2 200 per month  Selangor - Ulu Klang
10      100803021       Upperville @ 16 Quartz                              RM 1 800 per month  Selangor - Ulu Klang
11      100269934       Menara Mutiara                                      RM 1 700 per month  Selangor - Ulu Klang
12      100822786       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
13      100800569       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
14      100640906       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
15      100639093       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
16      100624445       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
17      100621654       3 Residen                       2010.0              RM 1 600 per month  Selangor - Ulu Klang
18      100567674       Menara Mutiara                                      RM 1 600 per month  Selangor - Ulu Klang
19      100677432       Main Place Residence            2014.0              RM 1 700 per month  Selangor - USJ
20      100710674       Main Place Residence            2014.0              RM 1 700 per month  Selangor - USJ
21      100774707       Main Place Residence            2014.0              RM 1 600 per month  Selangor - USJ
22      100829850       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
23      100788712       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
24      100685025       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
25      100833983       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
26      100807435       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
27      100650653       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
28      100769126       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
29      100640060       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
30      100549656       Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
31      99995307        Main Place Residence            2014.0              RM 1 500 per month  Selangor - USJ
32      100853714       Main Place Residence            2014.0              RM 1 450 per month  Selangor - USJ
33      100779261       Main Place Residence            2014.0              RM 1 450 per month  Selangor - USJ
34      100501252       Main Place Residence            2014.0              RM 1 400 per month  Selangor - USJ
35      99414953        Main Place Residence            2014.0              RM 1 400 per month  Selangor - USJ
36      100824931       Main Place Residence            2014.0              RM 1 400 per month  Selangor - USJ
37      99514056                                                            RM 1 300 per month  Selangor - USJ
38      100246616       Main Place Residence            2014.0              RM 1 200 per month  Selangor - USJ
39      100502712       Arcadia (Subang Jaya)                               RM 1 150 per month  Selangor - USJ
40      100642842       Sri Tanjung (USJ 16)            2007.0              RM 1 100 per month  Selangor - USJ
41      100627299       Sri Tanjung (USJ 16)            2007.0              RM 1 000 per month  Selangor - USJ
42      100614755       Sri Tanjung (USJ 16)            2007.0              RM 1 000 per month  Selangor - USJ
43      100782261       Sri Tanjung (USJ 16)            2007.0              RM 1 000 per month  Selangor - USJ
44      100615560       Sri Tanjung (USJ 16)            2007.0              RM 1 000 per month  Selangor - USJ
45      98040486        Main Place Residence            2014.0              RM 999 per month    Selangor - USJ
46      93661445                                                            RM 850 per month    Selangor - USJ
47      100667193       Sri Tanjung (USJ 16)            2007.0              RM 850 per month    Selangor - USJ
48      100588551       Tropicana Aman 1                                    RM 1 800 per month  Selangor - Telok Panglima Garang
49      100836583       Seri Tijanni                                        RM 1 500 per month  Selangor - Sungai Buloh
50      100823780       Suria Putra                     2018.0              RM 1 500 per month  Selangor - Sungai Buloh
Time Taken >> 0.690694
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit
```

5.2.4.3 Size as per square feet

```
        ads_id          prop_name                       size                monthly_rent        location
1       100255897       Angkasa Condominiums            99999999 sq.ft.     RM 110 per month    Kuala Lumpur - Cheras
2       100213068       Zamrud Apartment                48010 sq.ft.        RM 1 400 per month  Kuala Lumpur - Old Klang Road
3       99994083        Sri Wangsaria                   15000 sq.ft.        RM 3 300 per month  Kuala Lumpur - Bangsar
4       99659804        Mizumi Residences               9001 sq.ft.         RM 1 800 per month  Kuala Lumpur - Kepong
5       99706946        The Lumayan Apartment           8300 sq.ft.         RM 1 400 per month  Kuala Lumpur - Cheras
6       99755934        The Oval                        7800 sq.ft.         RM 17 000 per month Kuala Lumpur - KLCC
7       99790277        Embassyview                     7506 sq.ft.         RM 18 500 per month Kuala Lumpur - Ampang Hilir
8       100564173       Eclipse Residence @ Pan'gaea    6400 sq.ft.         RM 1 780 per month  Selangor - Cyberjaya
9       98976840                                        6067 sq.ft.         RM 12 000 per month Kuala Lumpur - Ampang Hilir
10      99973077        Damai 33                        5000 sq.ft.         RM 13 500 per month Kuala Lumpur - City Centre
11      100261441       Damai 33                        5000 sq.ft.         RM 13 500 per month Kuala Lumpur - City Centre
12      98976763                                        5000 sq.ft.         RM 8 000 per month  Kuala Lumpur - Ampang Hilir
13      100612908       Subang Olives Residence         4915 sq.ft.         RM 4 848 per month  Selangor - Subang Jaya
14      100612892       Subang Olives Residence         4915 sq.ft.         RM 6 888 per month  Selangor - Subang Jaya
15      95897047                                        4356 sq.ft.         RM 9 000 per month  Kuala Lumpur - Ampang Hilir
16      100078345       325 Persiaran Ritchie           4300 sq.ft.         RM 12 000 per month Kuala Lumpur - Ampang Hilir
17      99794408        325 Persiaran Ritchie           4300 sq.ft.         RM 12 000 per month Kuala Lumpur - Ampang Hilir
18      99740698        Brunsfield EmbassyView          4098 sq.ft.         RM 12 500 per month Kuala Lumpur - Cheras
19      100541053       Imperial Hatamas                4012 sq.ft.         RM 3 980 per month  Selangor - Cheras
20      100648736       Imperial Residence Cheras       4000 sq.ft.         RM 4 000 per month  Selangor - Cheras
21      88274040        Anjali @ North Kiara            3713 sq.ft.         RM 5 500 per month  Kuala Lumpur - Segambut
22      100611344       Villa Lagenda                   3700 sq.ft.         RM 2 999 per month  Selangor - Selayang
23      99732058        Impiana on The Waterfront       3650 sq.ft.         RM 8 500 per month  Kuala Lumpur - Ampang
24      100225460       Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
25      100226511       Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
26      100206265       Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
27      100159124       Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
28      100118818       Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
29      99793908        Hartamas Regency II             3488 sq.ft.         RM 5 800 per month  Kuala Lumpur - Mont Kiara
30      98873901                                        3400 sq.ft.         RM 10 000 per month Kuala Lumpur - Damansara Heights
31      99914450        Sky Vista Residency             3400 sq.ft.         RM 5 000 per month  Kuala Lumpur - Cheras
32      100576263       Tropicana Grande                3340 sq.ft.         RM 9 800 per month  Selangor - Petaling Jaya
33      77073260                                        3337 sq.ft.         RM 7 900 per month  Kuala Lumpur - KLCC
34      100576251       Tropicana Grande                3320 sq.ft.         RM 9 500 per month  Selangor - Petaling Jaya
35      100169369       Suria Stonor                    3283 sq.ft.         RM 7 000 per month  Kuala Lumpur - KLCC
36      100176173       Vila Vista                      3223 sq.ft.         RM 5 800 per month  Kuala Lumpur - Cheras
37      99914870        Vila Vista                      3223 sq.ft.         RM 5 800 per month  Kuala Lumpur - Cheras
38      100096563       Binjai Residency                3218 sq.ft.         RM 16 000 per month Kuala Lumpur - KLCC
39      100194364       The Binjai                      3218 sq.ft.         RM 16 000 per month Kuala Lumpur - KLCC
40      100257806       The Binjai                      3218 sq.ft.         RM 16 000 per month Kuala Lumpur - KLCC
41      98889591                                        3218 sq.ft.         RM 16 000 per month Kuala Lumpur - KLCC
42      100001385       The Binjai                      3218 sq.ft.         RM 15 000 per month Kuala Lumpur - KLCC
43      100003504       The Binjai                      3218 sq.ft.         RM 16 000 per month Kuala Lumpur - KLCC
44      100052711       Ceriaan Kiara                   3174 sq.ft.         RM 3 300 per month  Kuala Lumpur - Mont Kiara
45      99758200        Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
46      100214760       Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
47      100138802       Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
48      100060069       Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
49      100029073       Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
50      99967981        Damansara Foresta               3150 sq.ft.         RM 4 000 per month  Kuala Lumpur - Sri Damansara
Time Taken >> 1.93934
Data read >> 19992 rows
1. Previous Page
2. Next Page
3. Sorting column
4. Exit
```

5.3 Algorithm comparison

The table below shows the runtime of the algorithms in sorting specific columns:

Sorting algorithms	Monthly rent	location	Size as per square feet
Quick sort	5.21577 s	0.690694 s	1.93934 s
Merge sort	1.28766 s	0.520684 s	1.2296 s
Difference	3.92811 s	0.17001 s	0.70974 s

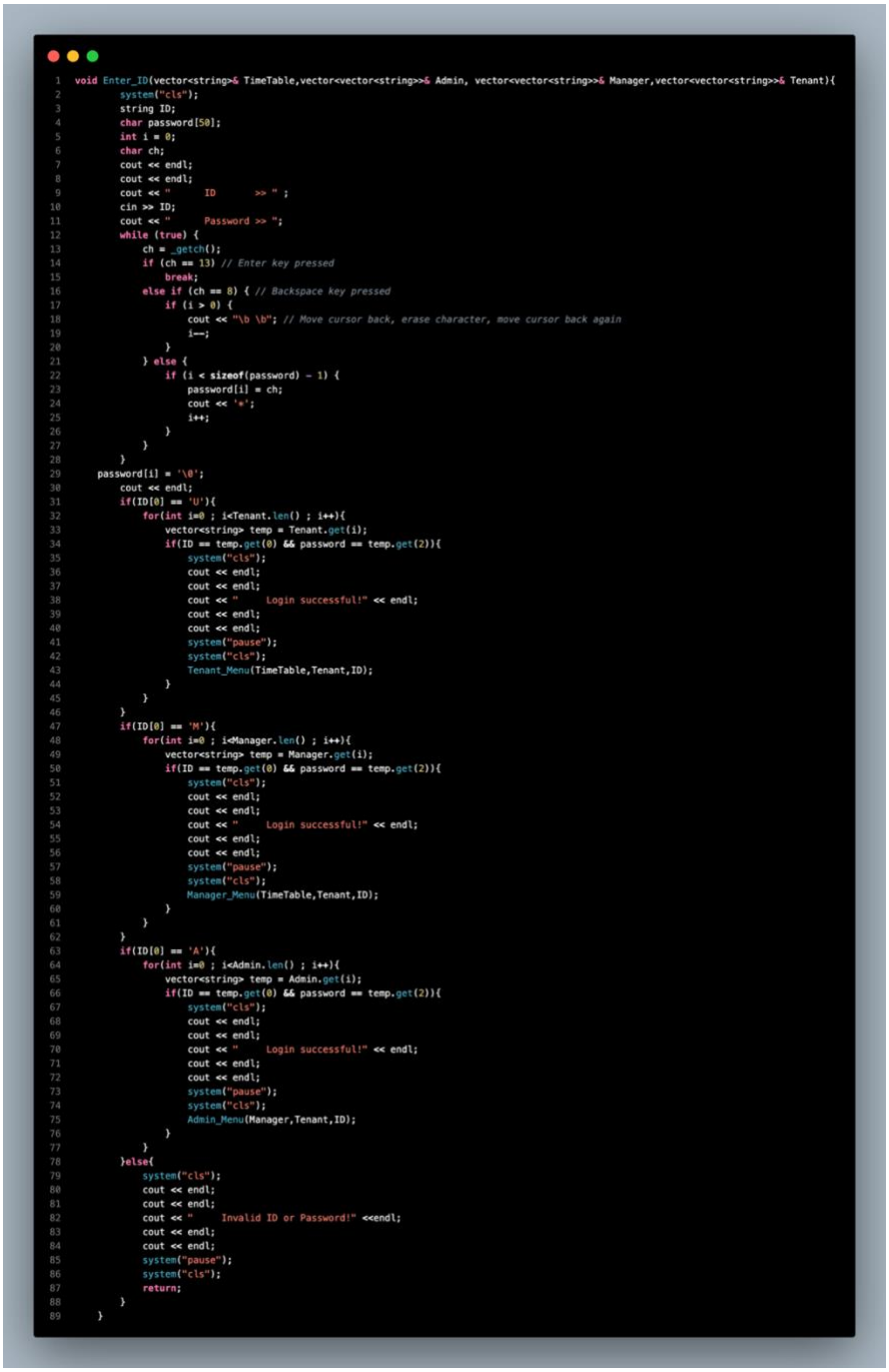
In all comparisons to the selected columns, the merge sort is faster than the quick sort.

	Merge Sort	Quick Sort
Space Complexity	O(n)	O(log n)
Time Complexity (Best Case)	O(n log n)	O(n log n)
Time Complexity (Worst Case)	O(n log n)	$O(n^2)$
Time Complexity (Average Case)	O(n log n)	O(n log n)
Advantages	Stable sorting, efficient in sorting massive, linked list	unstable sorting (the order of elements could be changed accidentally

According to their complexity in terms of time and space, quick sort is more efficient in sorting less data and merge sort is more superior in sorting a dataset with nearly 20 thousand records like ours.

6 Code Implementation

6.1 Login Function



```
void Enter_ID(vector<string>& TimeTable,vector<vector<string>>& Admin, vector<vector<string>>& Manager,vector<vector<string>>& Tenant){
        system("cls");
        string ID;
        char password[50];
        int i = 0;
        char ch;
        cout << endl;
        cout << endl;
        cout << "      ID       >> " ;
        cin >> ID;
        cout << "      Password >> ";
        while (true) {
            ch = _getch();
            if (ch == 13) // Enter key pressed
                break;
            else if (ch == 8) { // Backspace key pressed
                if (i > 0) {
                    cout << "\b \b"; // Move cursor back, erase character, move cursor back again
                    i--;
                }
            } else {
                if (i < sizeof(password) - 1) {
                    password[i] = ch;
                    cout << '*';
                    i++;
                }
            }
        }
    password[i] = '\0';
        cout << endl;
        if(ID[0] == 'U'){
            for(int i=0 ; i<Tenant.len() ; i++){
                vector<string> temp = Tenant.get(i);
                if(ID == temp.get(0) && password == temp.get(2)){
                    system("cls");
                    cout << endl;
                    cout << endl;
                    cout << "     Login successful!" << endl;
                    cout << endl;
                    cout << endl;
                    system("pause");
                    system("cls");
                    Tenant_Menu(TimeTable,Tenant,ID);
                }
            }
        }
        if(ID[0] == 'M'){
            for(int i=0 ; i<Manager.len() ; i++){
                vector<string> temp = Manager.get(i);
                if(ID == temp.get(0) && password == temp.get(2)){
                    system("cls");
                    cout << endl;
                    cout << endl;
                    cout << "     Login successful!" << endl;
                    cout << endl;
                    cout << endl;
                    system("pause");
                    system("cls");
                    Manager_Menu(TimeTable,Tenant,ID);
                }
            }
        }
        if(ID[0] == 'A'){
            for(int i=0 ; i<Admin.len() ; i++){
                vector<string> temp = Admin.get(i);
                if(ID == temp.get(0) && password == temp.get(2)){
                    system("cls");
                    cout << endl;
                    cout << endl;
                    cout << "     Login successful!" << endl;
                    cout << endl;
                    cout << endl;
                    system("pause");
                    system("cls");
                    Admin_Menu(Manager,Tenant,ID);
                }
            }
        }else{
            system("cls");
            cout << endl;
            cout << endl;
            cout << "     Invalid ID or Password!" <<endl;
            cout << endl;
            cout << endl;
            system("pause");
            system("cls");
            return;
        }
    }
```

6.2 Tenant Function

6.2.1 Tenant menu

```
void Tenant_Menu(vector<string> TimeTable,vector<vector<string>> tenant,string ID){
    while(true){
        cout << "==================================" << endl;
        cout << "1. Apartment Information " << endl;
        cout << "2. Search Apartment Information " << endl;
        cout << "3. Add to Favorite Property" << endl;
        cout << "4. Place a Rent Request" << endl;
        cout << "5. Property Renting History" << endl;
        cout << "6. Confirm Payment " <<endl;
        cout << "7. Profile " <<endl;
        cout << "8. Exit " << endl;
        cout << "==================================" << endl;
        cout << "Enter >> ";
        string option;
        cin >> option ;
        if(option == "1"){
            system("cls");
            Apartment_Information();
        }
        else if(option == "2"){
            system("cls");
            Search_Apartment_Information();
        }
        else if(option == "3"){
            system("cls");
            Add_to_Favorite(tenant,ID);
        }
        else if(option == "4"){
            system("cls");
            Place_Rent_Request(TimeTable,tenant,ID);
        }
        else if(option == "5"){
            system("cls");
            Renting_History(tenant,ID);
        }
        else if(option == "6"){
            system("cls");
            Confirm_Payment(tenant,ID);
        }
        else if(option == "7"){
            system("cls");
            Self_Information(tenant,ID);
        }
        else if(option == "8"){
            system("cls");
            return;
        }
        else{
            system("cls");
            cout << "==================================" << endl;
            cout << endl;
            cout << endl;
            cout << endl;
            cout << "        Invalid Option!" << endl;
            cout << endl;
            cout << endl;
            cout << endl;
            cout << endl;
            cout << "==================================" << endl;
            system("pause");
            system("cls");
        }
    }
}
```

6.2.2 Tenant register

```
void Tenant_Register(vector<vector<string>>& Tenant){
    vector<string> temp1;
    string temp2;

    vector<string> temp4;
    for(int i=0 ; i<Tenant.len() ; i++){
        vector<string> temp3 = Tenant.get(i);
        temp4.push_back(temp3.get(0));
    }
    string largestNumber = temp4.get(0);
    for (int i = 1; i < temp4.len(); i++) {
        if (temp4.get(i) > largestNumber) {
            largestNumber = temp4.get(i);
        }
    }

    std::string nextLargest = largestNumber;
    int numericPart = std::stoi(nextLargest.substr(1)) + 1;
    int temp = numericPart;
    int numericPart_length = 0;
    while(temp){
        numericPart_length++;
        temp = temp/10;
    }
    nextLargest = "U";
    for(int i=0 ; i<4-numericPart_length ; i++){
        nextLargest+="0";
    }
    nextLargest += std::to_string(numericPart);
    temp2 = nextLargest;
    cout << "Your New Tenant ID >> " << temp2 << endl;
    temp1.push_back(temp2);
    cout << "Name >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    cout << "Password >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    cout << "Email >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    temp1.push_back("0");
    temp1.push_back("");
    temp1.push_back("");
    temp1.push_back("");
    temp1.push_back("");
    Tenant.push_back(temp1);
}
```

The registration of the users in the APH system requires a Name, password, and email.

6.2.3 Sort and display property information in descending order

```
void Apartment_Information(){
    clock_t start, end;
    double time_taken = 0;
    readCSV temp("mudah-apartment-kl-selangor.csv");
    string option = "0";
    int page=1;
    int sorting_column =0;
    string sorting_method;
    temp.tenant_display_apartment(page);
    while(option!="4"){
        cout << "1. Previous Page " << endl;
        cout << "2. Next Page " << endl;
        cout << "3. Sorting column " << endl;
        cout << "4. Exit" << endl;
        cout << "Select an option >> " ;
        cin >> option;
        if(option == "1"){
            if(page!=1)
                page--;
            system("cls");
            temp.tenant_display_apartment(page);
        }else if(option == "2"){
            if(page!=400)
                page++;
            system("cls");
            temp.tenant_display_apartment(page);
        }else if(option == "3"){
                cout << "Sorting Column >> " << endl;
                cout << "1.  ID " << endl;
                cout << "2.  Property Name " << endl;
                cout << "3.  Completion Year "<< endl;
                cout << "4.  Monthly Rent " << endl;
                cout << "5.  Location " << endl;
                cout << "6.  Property Type " << endl;
                cout << "7.  Number of Rooms " << endl;
                cout << "8.  Number of Parking " << endl;
                cout << "9.  Number of Bathroom " << endl;
                cout << "10. Square Feet " << endl;
                cout << "11. Furnishing Status " << endl;
                cout << "12. Facilities " << endl;
                cout << "13. Additional Facilities " << endl;
                cout << "Sorting which column? >> ";
                cin >> sorting_column ;
                cout << "Sorting Method >> " << endl;
                cout << "1.  Quick Sort " << endl;
                cout << "2.  Merge Sort " << endl;
                cout << "With which method? >> ";
                cin >> sorting_method;
                if(sorting_method == "1"){
                    if(sorting_column==1||sorting_column==4||sorting_column==10){
                        start = clock();
                        temp.s_quickSort(1,temp.get_numRows()-1,sorting_column-1);
                        end = clock();
                        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
                    } else {
                        start = clock();
                        temp.s_quickSort(1,temp.get_numRows()-1,sorting_column-1);
                        end = clock();
                        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
                    }
                }else if(sorting_method == "2"){
                    if(sorting_column==1||sorting_column==4||sorting_column==10){
                        start = clock();
                        temp.s_mergeSort(1,temp.get_numRows()-1,sorting_column-1);
                        end = clock();
                        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
                    } else {
                        start = clock();
                        temp.s_mergeSort(1,temp.get_numRows()-1,sorting_column-1);
                        end = clock();
                        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
                    }
                } else {
                    cout << "Invalid Option" << endl;
                }
                system("cls");
                temp.tenant_display_apartment(page);
                cout << "Time Taken >> " << time_taken << endl;
        }else if(option == "4"){
            system("cls");
            return;
        } else {
            system("cls");
            cout << endl;
            cout << endl;
            cout << "    Invalid Option!" << endl;
            cout << endl;
            cout << endl;
            system("pause");
            system("cls");
            temp.tenant_display_apartment(page);
        }
    }
}
```

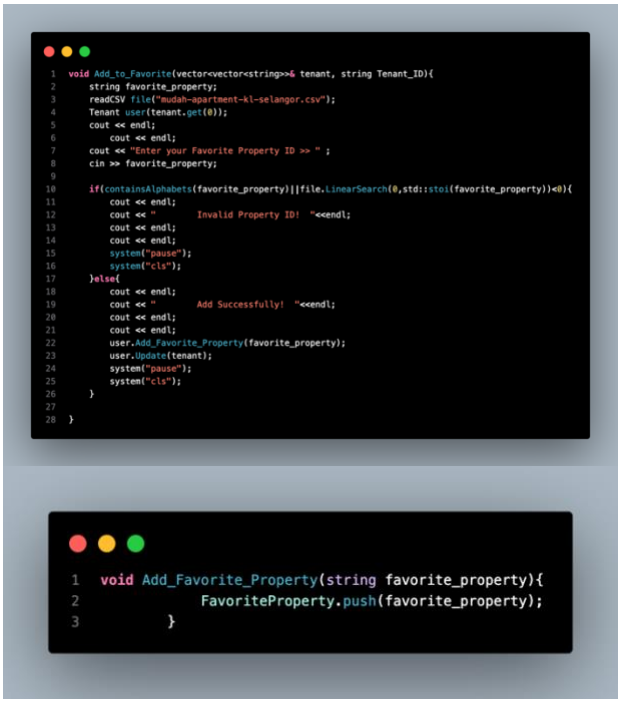
```
void tenant_display_apartment(int page){
            vector<string> temp1 = warehouse[0];
            for(int j=0 ; j<numCols-9 ; j++){
                if(j==0)
                    cout << setiosflags(ios::left)<< setw(10) << " ";
                if(j==1)
                    cout << setiosflags(ios::left)<< setw(45) << temp1.get(j);
                else
                    cout << setiosflags(ios::left)<< setw(20) << temp1.get(j);
            }
            cout << endl;
            int page_start = (page-1)*50+1;
            int page_end = (page-1)*50+51;
            if(page_end > numRows)
                page_end = numRows;
            for(int i=page_start ; i<page_end ; i++){
                vector<string> temp1 = warehouse[i];
                for(int j=0 ; j<numCols-9 ; j++){
                    if(j==0)
                        cout << setiosflags(ios::left)<< setw(10) << i;
                    if(j==1)
                        cout << setiosflags(ios::left)<< setw(45) << temp1.get(j);
                    else
                        cout << setiosflags(ios::left)<< setw(20) << temp1.get(j);
                }
                cout << endl;
            }
        }
```

The apartment_information() function is in all_interface.h and it will call the tenant_display_apartment(int page) function from readCSV_2.h during display information. The sorting algorithms choice is also integrated into this function.

6.2.4 Search and display property information

```
void Search_Apartment_Information()
{
    readCSV temp1("mudah-apartment-kl-selangor.csv");
    double time_taken = 0;
    clock_t start, end;
    int column;
    string target;
    string search_method;
    cout << endl;
    cout << endl;
    cout << "   Enter the ID >> " ;
    cin >> target;
    cout << "   Select Search Method " <<endl;
    cout << "   1. Linear Search " <<endl;
    cout << "   2. Binary Search " <<endl;
    cout << "   Select Search Method >> " ;
    cin >> search_method;
    int number;
    if(search_method =="1")
    {
        start = clock();
        number = temp1.LinearSearch(0,stoi(target));
        end = clock();
        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    }
    else if(search_method=="2")
    {
        temp1.i_mergeSort(1, temp1.get_numRows()-1,0);
        start = clock();
        number = temp1.Binary_Search(0,stoi(target));
        end = clock();
        time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    }
    else if(search_method !="1" && search_method !="2")
    {
        number = -2;
    }
    if(number == -1)
    {
        cout << endl;
        cout << endl;
        cout << "         Invalid ID               " << endl;
        cout << endl;
        cout << endl;
        cout << endl;
        cout << endl;
        system("pause");
        system("cls");
        return;
    }
    else if(number == -2)
    {
        cout << endl;
        cout << endl;
        cout << "         Invalid Search Method               " << endl;
        cout << endl;
        cout << endl;
        cout << endl;
        cout << endl;
        system("pause");
        system("cls");
        return;
    }
    vector<string> temp2 = temp1.get_row(temp1.search(target, 0));
    system("cls");
    cout << "================================" << endl;
    cout << "ads_id          >> " << temp2.get(0) << endl;
    cout << "prop_name       >> " << temp2.get(1) << endl;
    cout << "completion_year >> " << temp2.get(2) << endl;
    cout << "monthly_rent    >> " << temp2.get(3) << endl;
    cout << "location        >> " << temp2.get(4) << endl;
    cout << "property_type   >> " << temp2.get(5) << endl;
    cout << "rooms           >> " << temp2.get(6) << endl;
    cout << "parking         >> " << temp2.get(7) << endl;
    cout << "bathroom        >> " << temp2.get(8) << endl;
    cout << "size            >> " << temp2.get(9) << endl;
    cout << "fuinished       >> " << temp2.get(10) << endl;
    cout << "facilities      >> " << temp2.get(11) << endl;
    cout << "additional      >> " << temp2.get(12) << endl;
    cout << "region          >> " << temp2.get(13) << endl;
    cout << "================================" << endl;
    cout << "Time Taken >> " << time_taken << endl;
    system("pause");
    system("cls");
}
```

6.2.5 Save tenant's favorite property



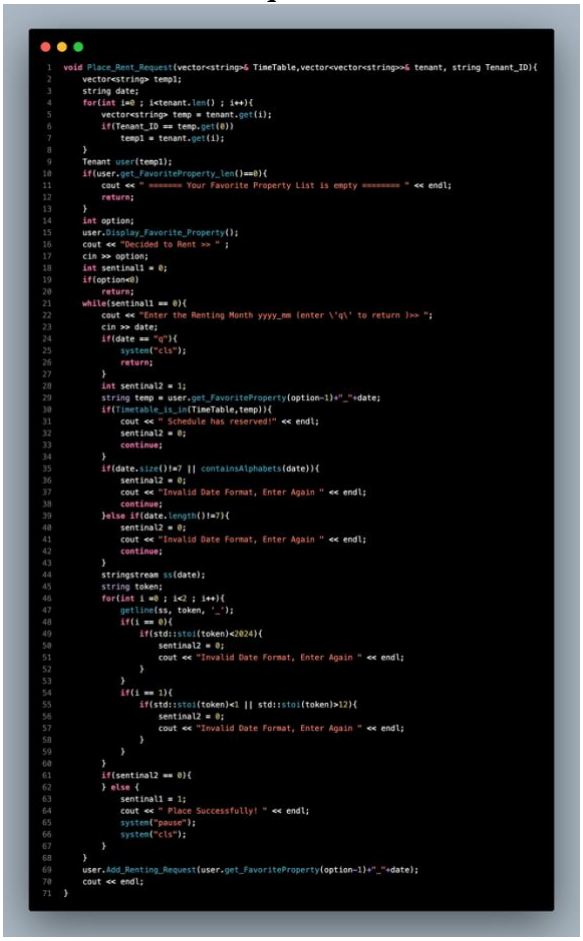
```
void Add_to_Favorite(vector<vector<string>>& tenant, string Tenant_ID){
    string favorite_property;
    readCSV file("mudah-apartment-kl-selangor.csv");
    Tenant user(tenant.get(0));
    cout << endl;
        cout << endl;
    cout << "Enter your Favorite Property ID >> " ;
    cin >> favorite_property;

    if(containsAlphabets(favorite_property)||file.LinearSearch(0,std::stoi(favorite_property))<0){
        cout << endl;
        cout << "       Invalid Property ID!  "<<endl;
        cout << endl;
        cout << endl;
        system("pause");
        system("cls");
    }else{
        cout << endl;
        cout << "       Add Successfully!  "<<endl;
        cout << endl;
        cout << endl;
        user.Add_Favorite_Property(favorite_property);
        user.Update(tenant);
        system("pause");
        system("cls");
    }
}
```

```
void Add_Favorite_Property(string favorite_property){
            FavoriteProperty.push(favorite_property);
        }
```

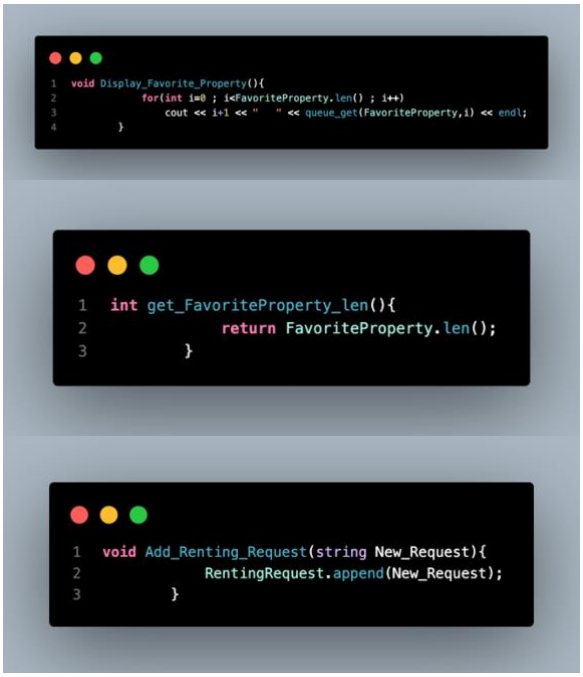
```
void Update(vector<vector<string>>& tenant){
            vector<string> temp1;
            string temp2 = "";
            temp1.push_back(userID);
            temp1.push_back(name);
            temp1.push_back(password);
            temp1.push_back(email);
            temp1.push_back(to_string(LastActive));
            for(int i=0 ; i<FavoriteProperty.len() ; i++){
                temp2+=queue_get(FavoriteProperty,i);
                if(i!=FavoriteProperty.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingRequest.len() ; i++){
                temp2+=RentingRequest.get(i);
                if(i!=RentingRequest.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingConfirm.len() ; i++){
                temp2+=RentingConfirm.get(i);
                if(i!=RentingConfirm.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingHistory.len() ; i++){
                temp2+=stack_get(RentingHistory,i);
                if(i!=RentingHistory.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<tenant.len() ; i++){
                vector<string> temp = tenant.get(i);
                if(userID == temp.get(0))
                tenant.set(i,temp1);
            }
        }
```

Add_to_Favorite(vector<vector<string>>& tenant, string Tenant_ID) will linearly search for the apartment information and then add it to the tenant's favorite list by calling Add_Favorite_Property(string favorite_property) function in all_user.h. Finally, update the tenant information by calling Update(vector<vector<string>>& tenant) function in all_user.h.

6.2.6 Place rent request



```
void Place_Rent_Request(vector<string>& TimeTable,vector<vector<string>>& tenant, string Tenant_ID){
    vector<string> temp1;
    string date;
    for(int i=0 ; i<tenant.len() ; i++){
        vector<string> temp = tenant.get(i);
        if(Tenant_ID == temp.get(0))
            temp1 = tenant.get(i);
    }
    Tenant user(temp1);
    if(user.get_FavoriteProperty_len()==0){
        cout << " ======= Your Favorite Property List is empty ======== " << endl;
        return;
    }
    int option;
    user.Display_Favorite_Property();
    cout << "Decided to Rent >> " ;
    cin >> option;
    int sentinal1 = 0;
    if(option<0)
        return;
    while(sentinal1 == 0){
        cout << "Enter the Renting Month yyyy_mm (enter \'q\' to return )>> ";
        cin >> date;
        if(date == "q"){
            system("cls");
            return;
        }
        int sentinal2 = 1;
        string temp = user.get_FavoriteProperty(option-1)+"_"+date;
        if(Timetable_is_in(TimeTable,temp)){
            cout << " Schedule has reserved!" << endl;
            sentinal2 = 0;
            continue;
        }
        if(date.size()!=7 || containsAlphabets(date)){
            sentinal2 = 0;
            cout << "Invalid Date Format, Enter Again " << endl;
            continue;
        }else if(date.length()!=7){
            sentinal2 = 0;
            cout << "Invalid Date Format, Enter Again " << endl;
            continue;
        }
        stringstream ss(date);
        string token;
        for(int i =0 ; i<2 ; i++){
            getline(ss, token, '_');
            if(i == 0){
                if(std::stoi(token)<2024){
                    sentinal2 = 0;
                    cout << "Invalid Date Format, Enter Again " << endl;
                }
            }
            if(i == 1){
                if(std::stoi(token)<1 || std::stoi(token)>12){
                    sentinal2 = 0;
                    cout << "Invalid Date Format, Enter Again " << endl;
                }
            }
        }
        if(sentinal2 == 0){
        } else {
            sentinal1 = 1;
            cout << " Place Successfully! " << endl;
            system("pause");
            system("cls");
        }
    }
    user.Add_Renting_Request(user.get_FavoriteProperty(option-1)+"_"+date);
    cout << endl;
}
```



```
void Display_Favorite_Property(){
        for(int i=0 ; i<FavoriteProperty.len() ; i++)
            cout << i+1 << "    " << queue_get(FavoriteProperty,i) << endl;
    }
```

```
int get_FavoriteProperty_len(){
        return FavoriteProperty.len();
    }
```

```
void Add_Renting_Request(string New_Request){
        RentingRequest.append(New_Request);
    }
```

Place_Rent_Request(vector<string>& TimeTable, vector<vector<string>>& tenant, string Tenant_ID) resides in all_interface.h. The function using includes Display_Favorite_Property(), get_FavoriteProperty_len() and Add_Renting_Request(string New_Request) from all_user.h. The RentingRequest is using the append function in doubly_circular_linked_list class.

6.2.7 Property renting history

```
void Renting_History(vector<vector<string>>& tenant, string Tenant_ID){
    vector<string> temp1;
    for(int i=0 ; i<tenant.len() ; i++){
        vector<string> temp = tenant.get(i);
        if(Tenant_ID == temp.get(0))
            temp1 = tenant.get(i);
    }
    Tenant user(temp1);
    user.Display_Renting_Property_History();
}
```

```
void Display_Renting_History(){
        for(int i=0 ; i<RentingHistory.len() ; i++)
            cout << i+1 << "    " << stack_get(RentingHistory,i) << endl;
    }
```

Renting_History(vector<vector<string>>& tenant, string Tenant_ID) will call Display_Renting_Property_History() from all_user.h and it will get the history from the RentingHistory stack.

6.2.8 Tenant confirm rental payment

```
void Confirm_Payment(vector<vector<string>> tenant, string Tenant_ID){
    vector<string> temp1;
    for(int i=0 ; i<tenant.len() ; i++){
        vector<string> temp = tenant.get(i);
        if(Tenant_ID == temp.get(0))
            temp1 = tenant.get(i);
    }
    Tenant user(temp1);
    if(user.get_RentingHistory_len()==0){
        cout << endl;
        cout << endl;
        cout << "There is no any Confimation\n\n" << endl;
        system("pause");
        system("cls");
        return;
    }
    user.Display_Renting_Confirm();
    int option;
    cout << "Enter the option that you want to pay >> ";
    cin >> option;
    vector<string> temp = split_(user.get_RentingConfirm(option-1));
    readCSV file2("mudah-apartment-kl-selangor.csv");
    vector<string> temp2 = file2.get_row(file2.search(temp.get(0),0));
    cout << temp2.get(3) <<endl;
    string answer;
    cout << "Confirm to pay?";
    cin >> answer;
    if(answer  == "yes"){
        user.Add_Renting_Property_History(user.get_RentingConfirm(option-1));;
        user.Delete_Renting_Confirm(option-1);
        user.Update(tenant);
        cout << " Successful Payment! " << endl;
    } else {
        cout << " Unsuccessful Payment! " << endl;
    }
    cout << endl;
}
```

```
void Display_Renting_Confirm(){
            cout << "index      id         year         month "<<endl;
            for(int i=0 ; i<RentingConfirm.len() ; i++){
                vector<string> temp = split_(RentingConfirm.get(i));
                cout << i+1 << "          ";
                cout << temp.get(0) << "    ";
                cout << temp.get(1) << "    ";
                cout << temp.get(2) << endl;
            }
        }
```

```
void Add_Renting_Property_History(string New_Renting_Property){
            RentingHistory.push(New_Renting_Property);
        }
```

```
void Delete_Renting_Confirm(int index){
            RentingConfirm.remove(index);
        }
```

```
void Update(vector<vector<string>>& tenant){
            vector<string> temp1;
            string temp2 = "";
            temp1.push_back(userID);
            temp1.push_back(name);
            temp1.push_back(password);
            temp1.push_back(email);
            temp1.push_back(to_string(LastActive));
            for(int i=0 ; i<FavoriteProperty.len() ; i++){
                temp2+=queue_get(FavoriteProperty,i);
                if(i!=FavoriteProperty.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingRequest.len() ; i++){
                temp2+=RentingRequest.get(i);
                if(i!=RentingRequest.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingConfirm.len() ; i++){
                temp2+=RentingConfirm.get(i);
                if(i!=RentingConfirm.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<RentingHistory.len() ; i++){
                temp2+=stack_get(RentingHistory,i);
                if(i!=RentingHistory.len()-1)
                temp2+=',';
            }
            temp1.push_back(temp2);
            temp2= "";
            for(int i=0 ; i<tenant.len() ; i++){
                vector<string> temp = tenant.get(i);
                if(userID == temp.get(0))
                tenant.set(i,temp1);
            }
        }
```

The Confirm_Payment(vector<vector<string>>& tenant, string Tenant_ID) is declared in all_interface.h. The functions required has Display_Renting_Confirm(), Add_Renting_property_History(), Delete_Renting_Confirm(), and Update in the all_user.h, therein the RentingConfirm is a circular linked list, RentingHistory is a stack, RentingRequest is a doubly circular linked list, and FavoriteProperty is a queue.

As the payment is confirmed, it will update the tenant profile using the update function. The functions it needs include push_back() and set() from the vector class, queue_get() from queue class, stack_get() from the stack class.

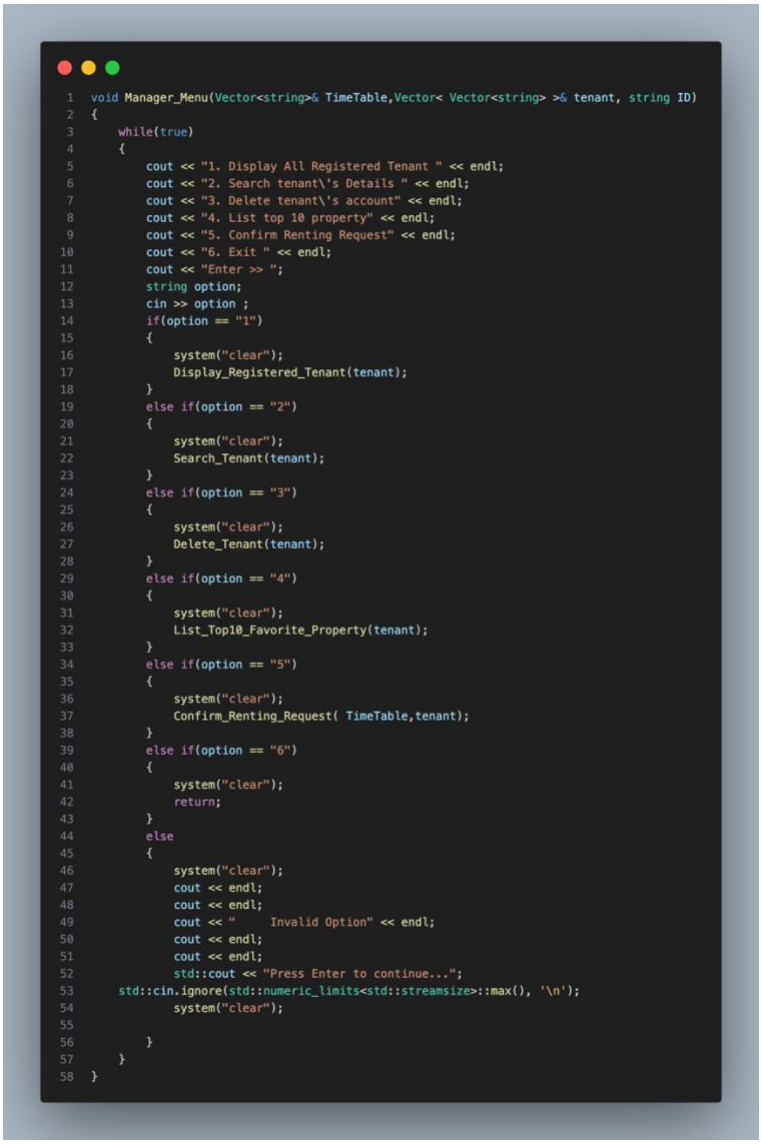
6.2.9 Tenant profile

```
void Self_Information(vector<vector<string>> tenant,string ID){
    string userID = ID;
    vector<string> temp;
    for(int i=0 ; i<tenant.len() ; i++){
        if(tenant.get(i).get(0) == userID)
        temp = tenant.get(i);
    }
    cout << "=========================================" << endl;
    cout << setiosflags(ios::left)<< setw(20) << "ID";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(0) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(1) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(2) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(3) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "LastActive";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Favorite Property";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(5) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Renting Request";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(6) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Renting Confirm";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(7) << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Renting History";
    cout << setiosflags(ios::left)<< setw(20) << temp.get(8) << endl;
    cout << "=========================================" << endl;
    system("pause");
    system("cls");
}
```

Self_Information(vector<vector<string>>& tenant, string ID) is the function to search for the tenant information using for loop linear search and display the tenant's information to the screen.

6.3 Manager Function

6.3.1 Manager's menu



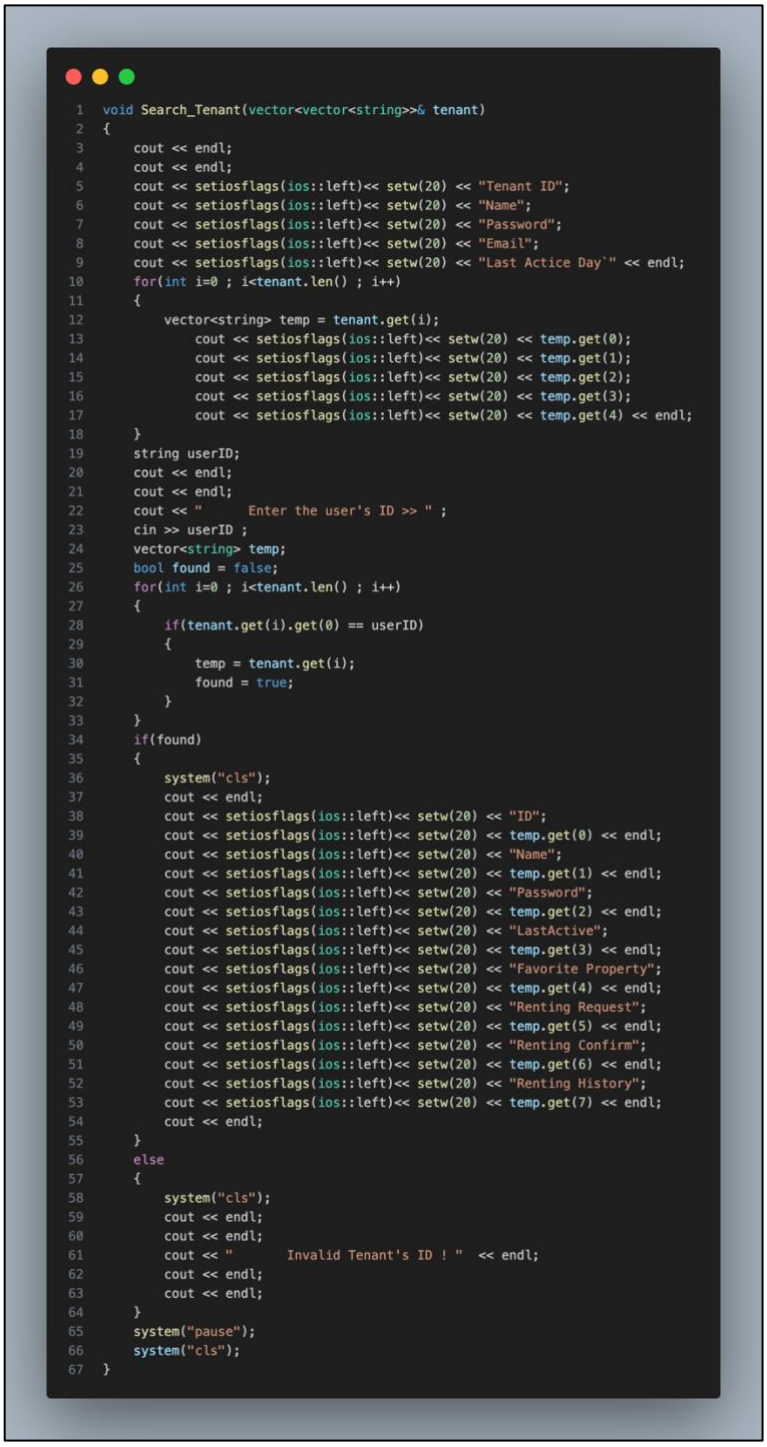
```
void Manager_Menu(Vector<string>& TimeTable,Vector< Vector<string> >& tenant, string ID)
{
    while(true)
    {
        cout << "1. Display All Registered Tenant " << endl;
        cout << "2. Search tenant\'s Details " << endl;
        cout << "3. Delete tenant\'s account" << endl;
        cout << "4. List top 10 property" << endl;
        cout << "5. Confirm Renting Request" << endl;
        cout << "6. Exit " << endl;
        cout << "Enter >> ";
        string option;
        cin >> option ;
        if(option == "1")
        {
            system("clear");
            Display_Registered_Tenant(tenant);
        }
        else if(option == "2")
        {
            system("clear");
            Search_Tenant(tenant);
        }
        else if(option == "3")
        {
            system("clear");
            Delete_Tenant(tenant);
        }
        else if(option == "4")
        {
            system("clear");
            List_Top10_Favorite_Property(tenant);
        }
        else if(option == "5")
        {
            system("clear");
            Confirm_Renting_Request( TimeTable,tenant);
        }
        else if(option == "6")
        {
            system("clear");
            return;
        }
        else
        {
            system("clear");
            cout << endl;
            cout << endl;
            cout << "     Invalid Option" << endl;
            cout << endl;
            cout << endl;
            std::cout << "Press Enter to continue...";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            system("clear");

        }
    }
}
```

6.3.2 Display tenants' details

```
void Display_Registered_Tenant(vector<vector<string>>& tenant)
{
    cout << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day";
    cout << setiosflags(ios::left)<< setw(20) << "Status" << endl;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        vector<string> temp = tenant.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4);
            int temp2 = stoi(temp.get(4));
            if(temp2<10)
            cout << setiosflags(ios::left)<< setw(20) << "Active" << endl;
            else
            cout << setiosflags(ios::left)<< setw(20) << "Inactive" << endl;
    }
    cout << endl;
    system("pause");
    system("cls");
}
```

The Display_Registered_Tenant() function is located in the all_interface.cpp and is used to display a list of information regarding registered tenants, such as ID, name, password, email, last active day and status.

6.3.3 Search tenants' details.



```
void Search_Tenant(vector<vector<string>>& tenant)
{
    cout << endl;
    cout << endl;
    cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        vector<string> temp = tenant.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    }
    string userID;
    cout << endl;
    cout << endl;
    cout << "      Enter the user's ID >> " ;
    cin >> userID ;
    vector<string> temp;
    bool found = false;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        if(tenant.get(i).get(0) == userID)
        {
            temp = tenant.get(i);
            found = true;
        }
    }
    if(found)
    {
        system("cls");
        cout << endl;
        cout << setiosflags(ios::left)<< setw(20) << "ID";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(0) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Name";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(1) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Password";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(2) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "LastActive";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(3) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Favorite Property";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Renting Request";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(5) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Renting Confirm";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(6) << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Renting History";
        cout << setiosflags(ios::left)<< setw(20) << temp.get(7) << endl;
        cout << endl;
    }
    else
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "       Invalid Tenant's ID ! "  << endl;
        cout << endl;
        cout << endl;
    }
    system("pause");
    system("cls");
}
```

The Search_Tenant() function is also located in the all_interface.cpp and functions as a search bar for manager to search tenants' information.

6.3.4 Delete tenants' account.

```
void Delete_Tenant(vector<vector<string>>& tenant)
{
    cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        vector<string> temp = tenant.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    }
    string tenantID;
    string answer;
    cout << "Enter Tenant ID >>";
    cin >> tenantID;
    for(int i=0 ; i<tenant.len() ;i++)
    {
        vector<string> temp = tenant.get(i);
        if(tenantID == temp.get(0))
        {
            cout << "Are you want to remove" << tenantID  << "? (enter \'y\' to proceed) >> ";
            cin >> answer;
            if(answer == "y")
            {
                cout << "  remove sucessfully!" << endl;
                tenant.remove(i);
                system("pause");
                system("cls");
            }
            else
            {
                cout << "  remove unsucessfully!" << endl;
                system("pause");
                system("cls");
                return;
            }
        }
        else
        {
            cout << " Invalid ID !" << endl;
        }
    }
    system("cls");
}
```

The Delete_Tenant() function is also located in the all_interface.cpp and functions as a delete button for Managers to delete tenants' account based on activity status.

6.3.5 Generate report

```
void List_Top10_Favorite_Property(vector<vector<string>>& tenant)
{
    map<string,int> map1;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        vector<string> temp1 = tenant.get(i);
        Tenant user(temp1);
        queue<string> temp2 = user.get_FavoriteProperty_queue();

        for(int j=0 ; j<temp2.len() ; j++)
        {
            if(map1.contains(temp2.get(j)))
            map1.plus_one(temp2.get(j));
            else
            map1.insert(temp2.get(j),1);
        }
    }
    map1.sort();
    map1.report();
    system("pause");
    system("cls");
}
```

```
queue<string> get_FavoriteProperty_queue()
        {
            return FavoriteProperty;
        }
```

List_Top10_Favourite_Property resides in all_interface.h. The function uses get_FavouriteProperty_queue() from all_user.h.

6.3.6 Manage tenancy process.

```
void Confirm_Renting_Request(vector<string>& TimeTable,vector<vector<string>>& tenant)
{
    vector<string> temp1;
    string userID;
    cout << "new request :" <<endl;
    for(int i=0 ; i<tenant.len() ; i++)
    {
        temp1 = tenant.get(i);
        if(temp1.get(6)!="")
        {
            cout << temp1.get(0) << " : ";
            cout << temp1.get(6) << endl;
        }

    }
    cout << "Enter user's ID >> ";
    cin >> userID;


    for(int i=0 ; i<tenant.len() ; i++)
    {
        vector<string> temp = tenant.get(i);
        if(userID == temp.get(0))
        {
            temp1 = tenant.get(i);
        }
    }
    Tenant user(temp1);
    user.Display_Renting_Request();
    cout << "Confirm ID option >> " ;
    int option;
    cin >> option;
    if(option <= 0 || option >user.get_RentingRequest_length())
    {
        cout << " Invalid Option" <<endl;
        system("pause");
        system("cls");
    }
    else
    {
        cout << " confirmed renting request successfully!" <<endl;
        system("pause");
        system("cls");
    }
    string ID = user.get_RentingRequest(option-2);
    TimeTable.push_back(user.get_RentingRequest(option-2));
    user.Add_Renting_Confirm(user.get_RentingRequest(option-2));
    user.Delete_Renting_Request(option-1);
    user.Update(tenant);

    for(int i=0 ; i<tenant.len(); i++)
    {
        vector<string>temp1 = tenant.get(i);
        Tenant user(temp1);
        cout << "Tenant " << user.get_name() << endl;
        for(int j=0 ; j<user.get_RentingRequest_length() ; j++)
        {
                if(user.get_RentingRequest(j) == ID)
                {
                    user.Delete_Renting_Request(j);
                    user.Update(tenant);
                    continue;
                }
            }
    }
}
```

```
int get_RentingRequest_length()
        {
            return RentingRequest.len();
        }
```

```
void Add_Renting_Request(string New_Request)
        {
            RentingRequest.append(New_Request);
        }
```

```
string get_RentingRequest(int index)
        {
            return RentingRequest.get(index);
        }
```

```
void Delete_Renting_Request(int index)
        {
            RentingRequest.remove(index);
        }
```



```
void Display_Renting_Request()
        {
            cout << setiosflags(ios::left)<< setw(20) << "Option";
            cout << setiosflags(ios::left)<< setw(20) << "ID";
            cout << setiosflags(ios::left)<< setw(20) << "Year";
            cout << setiosflags(ios::left)<< setw(20) << "Month" << endl;
            for(int i=0 ; i<RentingRequest.len() ; i++){
                Vector<string> temp = split_(RentingRequest.get(i));
                cout << setiosflags(ios::left)<< setw(20) << i+1;
                cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
                cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
                cout << setiosflags(ios::left)<< setw(20) << temp.get(2) << endl;
            }
        }
```

The Confirm_Renting_Request() is a function residing in the all_interface.cpp. The function uses get_RentingRequest_length(), Add_Renting_Request(), get_RentingRequest(), Delete_Renting_Request() and Display_Renting_Request() from all_user.h.

6.4 Admin Function

6.4.1 Admin Menu

```
void Admin_Menu(vector<vector<string>>& manager,vector<vector<string>>& tenant, string ID){
    while(true){
        cout << "1. Add New Manager " << endl;
        cout << "2. Modify the manager\'s status " << endl;
        cout << "3. Delete tenant\'s account" << endl;
        cout << "4. Display Manager " << endl;
        cout << "5. Display Tenants" << endl;
        cout << "6. Property Information " << endl;
        cout << "7. Tenant Information " << endl;
        cout << "8. Exit " << endl;
        cout << "Enter >> ";
        string option;
        cin >> option ;
        if(option == "1"){
            system("cls");
            Add_Manager(manager);
        }
        else if(option == "2"){
            system("cls");
            Modify_Status(manager);
        }
        else if(option == "3"){
            system("cls");
            Delete_Tenant(tenant);
        }
        else if(option == "4"){
            system("cls");
            Display_Manager(manager);
        }
        else if(option == "5"){
            system("cls");
            display_tenant(tenant);
        }
        else if(option == "6"){
            system("cls");
            property_filter();
        }
        else if(option == "7"){
            system("cls");
            tenant_filter(tenant);
        }
        else if(option == "8"){
            system("cls");
            return;
        }else{
            system("cls");
            cout << endl;
            cout << endl;
            cout << "     Invalid Option" << endl;
        }

    }
}
```

6.4.2 Add New Manager

```
void Add_Manager(vector<vector<string>>& manager){
    vector<string> temp1;
    string temp2;

    vector<string> temp4;
    for(int i=0 ; i<manager.len() ; i++){
        vector<string> temp3 = manager.get(i);
        temp4.push_back(temp3.get(0));
    }
    string largestNumber = temp4.get(0);
    for (int i = 1; i < temp4.len(); i++) {
        if (temp4.get(i) > largestNumber) {
            largestNumber = temp4.get(i);
        }
    }

    std::string nextLargest = largestNumber;
    int numericPart = std::stoi(nextLargest.substr(1)) + 1;
    int temp = numericPart;
    int numericPart_length = 0;
    while(temp){
        numericPart_length++;
        temp = temp/10;
    }
    nextLargest = "M";
    for(int i=0 ; i<4-numericPart_length ; i++){
        nextLargest+="0";
    }

    nextLargest += std::to_string(numericPart);
    temp2 = nextLargest;
    cout << "New Manager ID >> " << temp2 << endl;
    temp1.push_back(temp2);
    cout << "Name >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    cout << "Password >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    cout << "Email >> ";
    cin >> temp2;
    temp1.push_back(temp2);
    temp1.push_back("active");
    manager.push_back(temp1);
}
```

The Add_Manager() function is located within the ALL_interface.cpp file. This function helps to add a new manager into the database by receiving 3 inputs by the user, Name, Password and Email. The manager ID is auto generated while the manager status is always set to Active when creating a new manager.

6.4.3 Modify Manager Status

```
void Modify_Status(vector<vector<string>>& manager){
    cout << setiosflags(ios::left)<< setw(20) << "Manager ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Status" << endl;
    for(int i=0 ; i<manager.len() ; i++){
        vector<string> temp = manager.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    }
    string userID;
    cout << "ManagerID >>" ;
    cin >> userID;
    for(int i=0 ; i<manager.len() ;i++){
        vector<string> temp = manager.get(i);
        if(userID == temp.get(0)){
            if(temp.get(4)=="active"){
                temp.set(4,"inactive");
                cout << "The status has turned into Inactive! " << endl;
            }
            else{
                temp.set(4,"active");
                cout << "The status has turned into Inactive! " << endl;
            }

        }
    }
    system("pause");
    system("cls");
}
```

The Modify_Manager() function is in the ALL_Interface.cpp file. This function allows the admin to change the status of the manager. The admin will have to key in the manager ID then press enter and the system will change the status of the manager to active or inactive according to the current status of the manager.

6.4.4 Delete Tenant's account

```
void Delete_Tenant(vector<vector<string>>& tenant){
    cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
    for(int i=0 ; i<tenant.len() ; i++){
        vector<string> temp = tenant.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    }
    string tenantID;
    string answer;
    cout << "Enter Tenant ID >>";
    cin >> tenantID;
    for(int i=0 ; i<tenant.len() ;i++){
        vector<string> temp = tenant.get(i);
        if(tenantID == temp.get(0)){
            cout << "Are you want to remove" << tenantID  << "? (enter \'y\' to proceed) >> ";
            cin >> answer;
            if(answer == "y"){
                cout << "  remove sucessfully!" << endl;
                tenant.remove(i);
                system("pause");
                system("cls");
            }else{
                cout << "  remove unsucessfully!" << endl;
                system("pause");
                system("cls");
                return;
            }
        }else{
            cout << " Invalid ID !" << endl;
        }
    }
    system("cls");
}
```

The Delete_Tenant() function is in the ALL_Interface.cpp file. This function allows the admin to delete the record of a tenant in the system. The admin will need to key in the tenant ID then press enter. The admin will first be prompted with a confirmation message to confirm the deletion process. If the admin inputs "y" the function will delete the tenant record from the system. If any other key is input, the function will give a message that the deletion was unsuccessful. Otherwise, if the tenant ID was initially wrong, a message will also prompt the admin that the ID is invalid.

6.4.5 Display Manager

```
void Display_Manager(vector<vector<string>>& manager){
    cout << setiosflags(ios::left)<< setw(20) << "Manager ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Status" << endl;
    for(int i=0 ; i<manager.len() ; i++){
        vector<string> temp = manager.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
    }
}
```

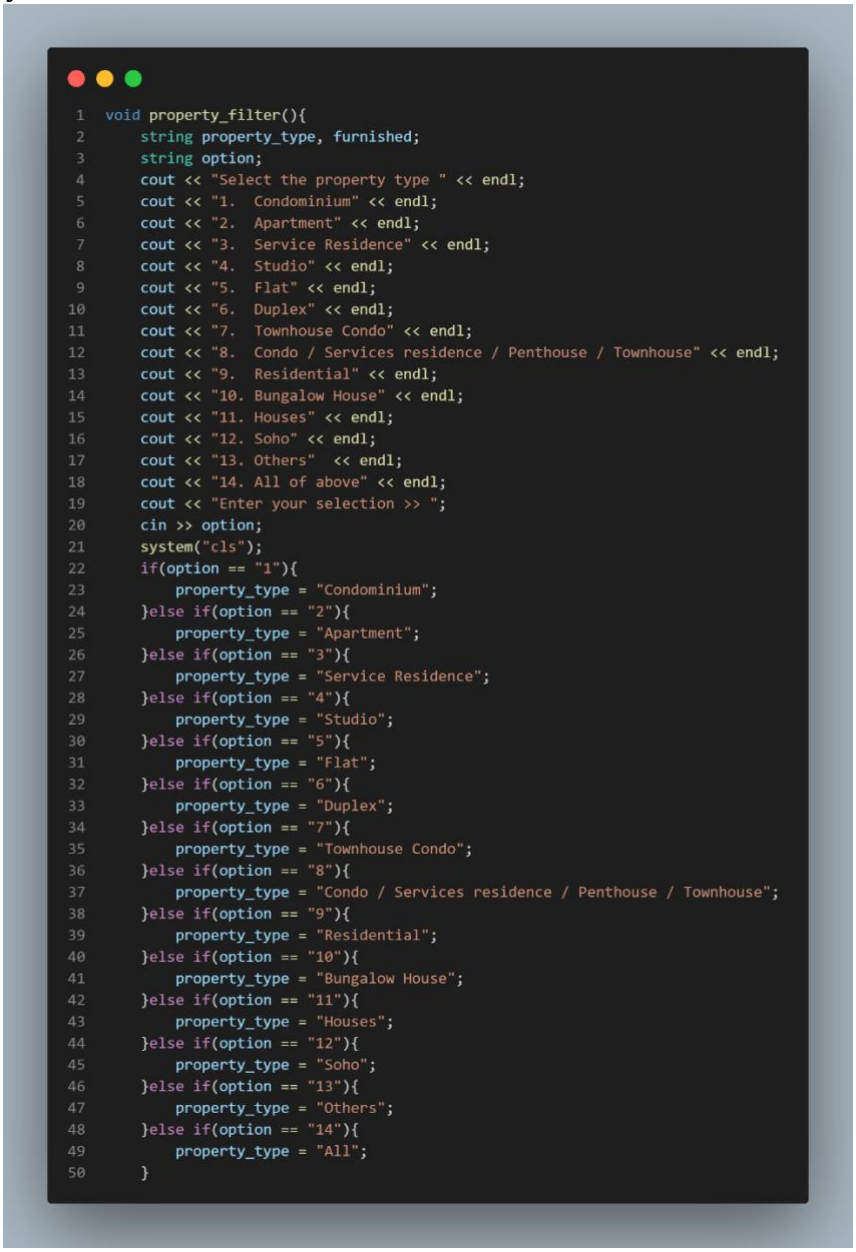
The Display_Manager() function is in the ALL_Interface.cpp file. This function will allow the admin to view all the current managers within the system. The function will call the records of the managers using a for loop.

6.4.6 Display Tenant

```
void display_tenant(vector<vector<string>>& tenant){
    cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
    cout << setiosflags(ios::left)<< setw(20) << "Name";
    cout << setiosflags(ios::left)<< setw(20) << "Password";
    cout << setiosflags(ios::left)<< setw(20) << "Email";
    cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
    for(int i=0 ; i<tenant.len() ; i++){
        vector<string> temp = tenant.get(i);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
            cout << setiosflags(ios::left)<< setw(20) << temp.get(4);
            int temp2 = stoi(temp.get(4));
            if(temp2<10)
            cout << setiosflags(ios::left)<< setw(20) << "Active" << endl;
            else
            cout << setiosflags(ios::left)<< setw(20) << "Inactive" << endl;
    }
}
```

The display_tenant() function is found in the ALL_Interface.cpp file. This function allows the admin to display all the tenant information. This function will call all the tenant records within the system and display it using a for loop. If the Last active day was more than 10 days, the function will also display the status of the tenant as ‘Inactive’ and ‘Active’ if otherwise.

6.4.7 Property Information



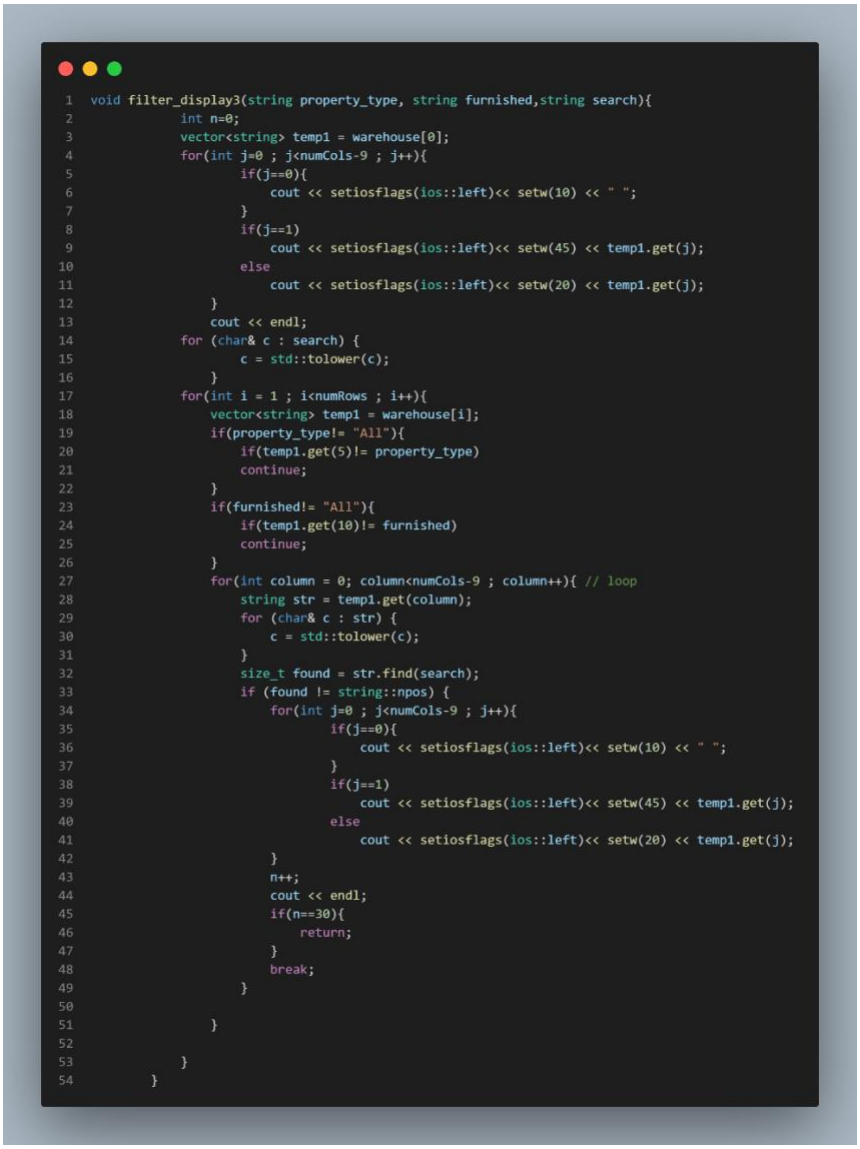
```
void property_filter(){
    string property_type, furnished;
    string option;
    cout << "Select the property type " << endl;
    cout << "1.  Condominium" << endl;
    cout << "2.  Apartment" << endl;
    cout << "3.  Service Residence" << endl;
    cout << "4.  Studio" << endl;
    cout << "5.  Flat" << endl;
    cout << "6.  Duplex" << endl;
    cout << "7.  Townhouse Condo" << endl;
    cout << "8.  Condo / Services residence / Penthouse / Townhouse" << endl;
    cout << "9.  Residential" << endl;
    cout << "10. Bungalow House" << endl;
    cout << "11. Houses" << endl;
    cout << "12. Soho" << endl;
    cout << "13. Others"  << endl;
    cout << "14. All of above" << endl;
    cout << "Enter your selection >> ";
    cin >> option;
    system("cls");
    if(option == "1"){
        property_type = "Condominium";
    }else if(option == "2"){
        property_type = "Apartment";
    }else if(option == "3"){
        property_type = "Service Residence";
    }else if(option == "4"){
        property_type = "Studio";
    }else if(option == "5"){
        property_type = "Flat";
    }else if(option == "6"){
        property_type = "Duplex";
    }else if(option == "7"){
        property_type = "Townhouse Condo";
    }else if(option == "8"){
        property_type = "Condo / Services residence / Penthouse / Townhouse";
    }else if(option == "9"){
        property_type = "Residential";
    }else if(option == "10"){
        property_type = "Bungalow House";
    }else if(option == "11"){
        property_type = "Houses";
    }else if(option == "12"){
        property_type = "Soho";
    }else if(option == "13"){
        property_type = "Others";
    }else if(option == "14"){
        property_type = "All";
    }
```

The property_filter() function is found in the ALL_Interface.cpp file. This function allows the admin to view the different property listings found within the database. It is broken down into two different sections here. The first code section prompts the admin to choose between 14 options to filter the property type.

```
cout << "Select the property type " << endl;
cout << "1.  Fully Furnished" << endl;
cout << "2.  Partially Furnished" << endl;
cout << "3.  Not Furnished" << endl;
cout << "4. All of above" << endl;
cout << "Enter your selection >> ";
cin >> option;
system("cls");
if(option == "1"){
    furnished = "Fully Furnished";
}else if(option == "2"){
    furnished = "Partially Furnished";
}else if(option == "3"){
    furnished = "Not Furnished";
}else if(option == "4"){
    furnished = "All";
}
```

The second half of the code here prompts the admin again to choose another filter for the property type out of 4 different options. Once the filter options have been set, the function then uses the readCSV class to call the records from the database file and the corresponding properties will be listed using the filter_display3() function as seen below.

```
    readCSV temp("mudah-apartment-kl-selangor.csv");
    char search[50];
    int i = 0;
    char ch;
    cout << "Property Type: " << property_type << "\nFurnished    : "<< furnished << "\nSearch >> " << endl;
    temp.filter_display3(property_type,furnished,"");
//    temp.filter_display2("");
    while (ch = _getch()) {

        system("cls");
        search[i] = '\0';
        cout << "Property Type: " << property_type << "\nFurnished    : "<< furnished << "\nSearch >> " << search;
        if (ch == 13){ // Enter key pressed
            system("cls");
            return;
        }
        else if (ch == 8) { // Backspace key pressed
            if (i > 0) {
                cout << "\b \b"; // Move cursor back, erase character, move cursor back again
                i--;
                search[i] = '\0';
            }
        } else {
            if (i < sizeof(search) - 1) {
                search[i] = ch;
                cout << ch;
                i++;
                search[i] = '\0';
            }
        }
        cout << endl;
        temp.filter_display3(property_type,furnished,search);
    }

}
```



```
void filter_display3(string property_type, string furnished,string search){
            int n=0;
            vector<string> temp1 = warehouse[0];
            for(int j=0 ; j<numCols-9 ; j++){
                    if(j==0){
                        cout << setiosflags(ios::left)<< setw(10) << " ";
                    }
                    if(j==1)
                        cout << setiosflags(ios::left)<< setw(45) << temp1.get(j);
                    else
                        cout << setiosflags(ios::left)<< setw(20) << temp1.get(j);
                }
                cout << endl;
            for (char& c : search) {
                    c = std::tolower(c);
                }
            for(int i = 1 ; i<numRows ; i++){
                vector<string> temp1 = warehouse[i];
                if(property_type!= "All"){
                    if(temp1.get(5)!= property_type)
                    continue;
                }
                if(furnished!= "All"){
                    if(temp1.get(10)!= furnished)
                    continue;
                }
                for(int column = 0; column<numCols-9 ; column++){ // loop
                    string str = temp1.get(column);
                    for (char& c : str) {
                        c = std::tolower(c);
                    }
                    size_t found = str.find(search);
                    if (found != string::npos) {
                        for(int j=0 ; j<numCols-9 ; j++){
                                if(j==0){
                                    cout << setiosflags(ios::left)<< setw(10) << " ";
                                }
                                if(j==1)
                                    cout << setiosflags(ios::left)<< setw(45) << temp1.get(j);
                                else
                                    cout << setiosflags(ios::left)<< setw(20) << temp1.get(j);
                        }
                        n++;
                        cout << endl;
                        if(n==30){
                            return;
                        }
                        break;
                    }

                }

            }
        }
```

Above is the filter_display3() function used to display the property records for the admin. This function is found in the readCSV_2.h file.

6.4.8 Tenant Information

```
1  void tenant_filter(vector<vector<string>>& tenant){
2      char search[50];
3      int i = 0;
4      char ch;
5      cout << "Search >> " << endl;
6      cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
7      cout << setiosflags(ios::left)<< setw(20) << "Name";
8      cout << setiosflags(ios::left)<< setw(20) << "Password";
9      cout << setiosflags(ios::left)<< setw(20) << "Email";
10     cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
11     for(int row = 0 ; row<tenant.len() ; row++){
12         vector<string> temp = tenant.get(row);
13         cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
14         cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
15         cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
16         cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
17         cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
18     }
```

This is the tenant_filter() function located in the ALL_Interface.cpp file. This function allows the admin to search for the desired tenant using their tenant ID. This function is separated into 2 sections. The first section displays the all the tenant information on the screen without the admin input for the tenant ID using a for loop.

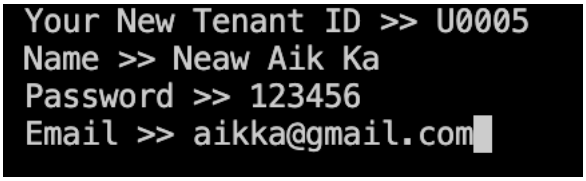
```
    while (ch = _getch()) {
        system("cls");
        search[i] = '\0';
        cout << "Search >> " << search;
        if (ch == 13){ // Enter key pressed
            system("cls");
            return;
        }
        else if (ch == 8) { // Backspace key pressed
            if (i > 0) {
                cout << "\b \b"; // Move cursor back, erase character, move cursor back again
                i--;
                search[i] = '\0';
            }
        } else {
            if (i < sizeof(search) - 1) {
                search[i] = ch;
                cout << ch;
                i++;
                search[i] = '\0';
            }
        }
        cout << endl;
        cout << setiosflags(ios::left)<< setw(20) << "Tenant ID";
        cout << setiosflags(ios::left)<< setw(20) << "Name";
        cout << setiosflags(ios::left)<< setw(20) << "Password";
        cout << setiosflags(ios::left)<< setw(20) << "Email";
        cout << setiosflags(ios::left)<< setw(20) << "Last Actice Day`" << endl;
        string search_temp = search;
        for (char& c : search_temp) {
                    c = std::tolower(c);
        }
        for(int row = 0 ; row<tenant.len() ; row++){
            vector<string> temp = tenant.get(row);
                    for(int column = 0; column<5 ; column++){ // loop
                        string str = temp.get(column);
                        for (char& c : str) {
                            c = std::tolower(c);
                        }
                        size_t found = str.find(search_temp);
                        if (found != string::npos) {
                            cout << setiosflags(ios::left)<< setw(20) << temp.get(0);
                            cout << setiosflags(ios::left)<< setw(20) << temp.get(1);
                            cout << setiosflags(ios::left)<< setw(20) << temp.get(2);
                            cout << setiosflags(ios::left)<< setw(20) << temp.get(3);
                            cout << setiosflags(ios::left)<< setw(20) << temp.get(4) << endl;
                            break;
                        }

                    }
        }
//        temp.filter_display3(property_type,furnished,search);
    }

}
```

This second section shows how the function will loop back the filtering according to the tenant ID input by the admin. If the desired tenant ID is found it will be displayed on the screen.

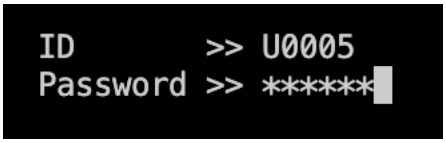
7 System Flow

7.1 Login and sign up

The main interface when the system starts will shows 3 options for users.



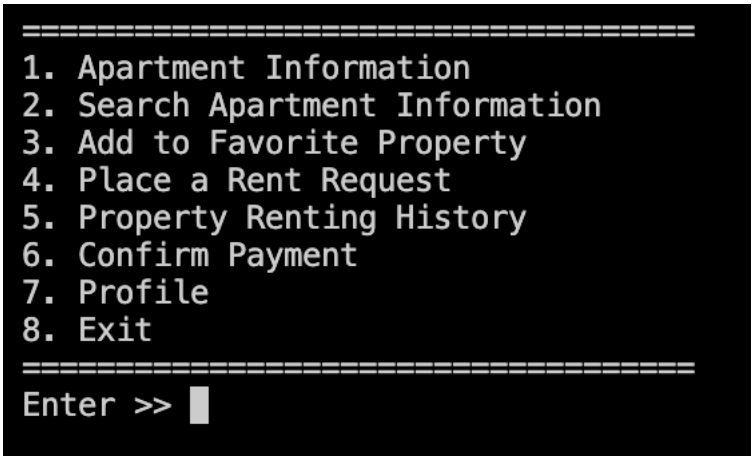
The register page will generate the user id automatically and ask the user to enter the name, password, and email.



For the login page, user can enter their ID and password to login. The password will be hide using '*'.

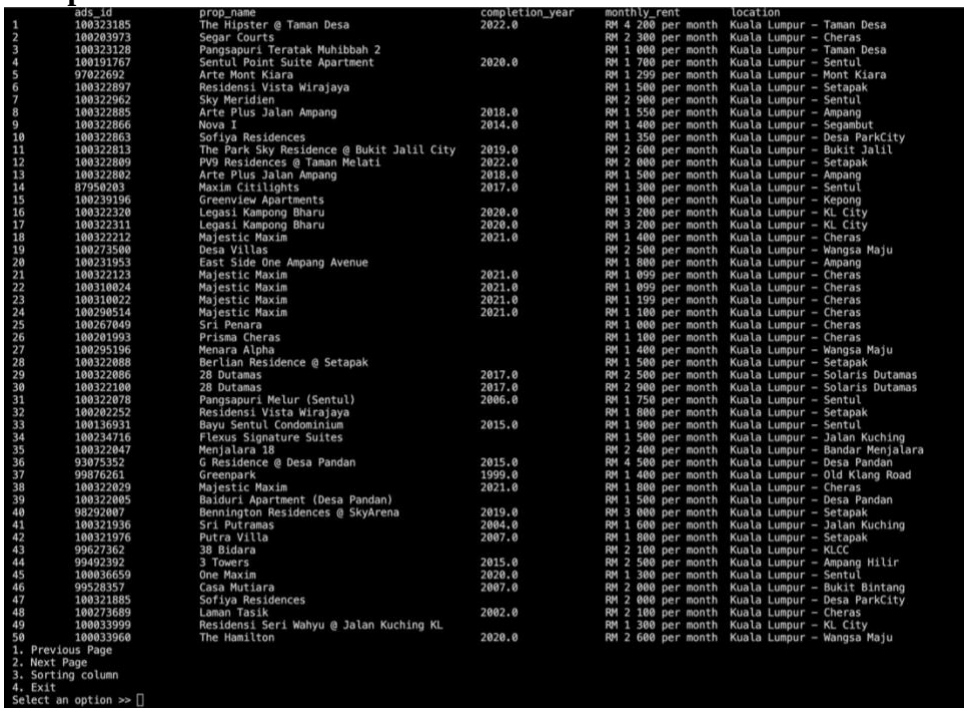
7.2 Tenant

7.2.1 Tenant menu



The main menu of the tenant has 8 functions that the user needs to enter the number to make their choice.

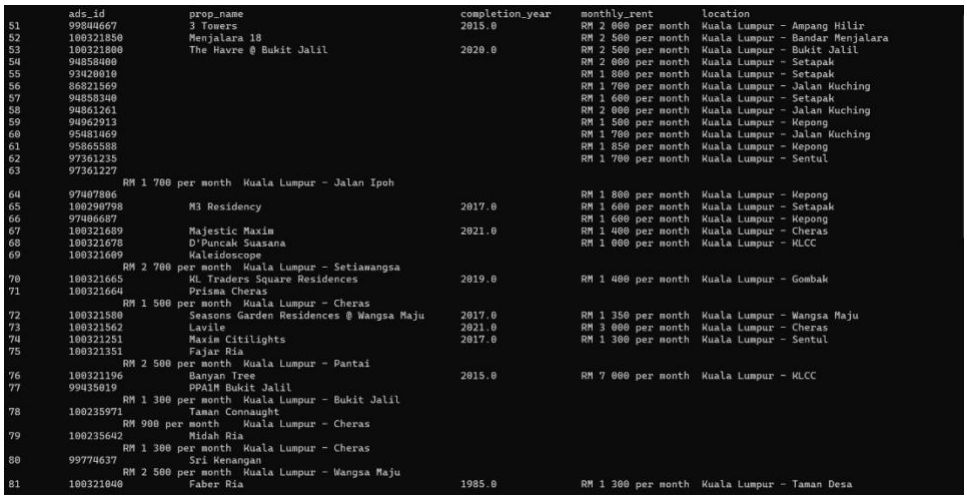
7.2.2 Check apartment information.



	ads_id	prop_name	completion_year	monthly_rent	location
1	100323185	The Hipster @ Taman Desa	2022.0	RM 4 200 per month	Kuala Lumpur - Taman Desa
2	100203973	Segar Courts		RM 2 300 per month	Kuala Lumpur - Cheras
3	100323128	Pangsapuri Teratak Muhibbah 2		RM 1 000 per month	Kuala Lumpur - Taman Desa
4	100191767	Sentul Point Suite Apartment	2020.0	RM 1 700 per month	Kuala Lumpur - Sentul
5	97022692	Arte Mont Kiara		RM 1 299 per month	Kuala Lumpur - Mont Kiara
6	100322897	Residensi Vista Wirajaya		RM 1 500 per month	Kuala Lumpur - Setapak
7	100322962	Sky Meridien		RM 2 900 per month	Kuala Lumpur - Sentul
8	100322885	Arte Plus Jalan Ampang	2018.0	RM 1 550 per month	Kuala Lumpur - Ampang
9	100322866	Nova I	2014.0	RM 1 400 per month	Kuala Lumpur - Segambut
10	100322863	Sofiya Residences		RM 1 350 per month	Kuala Lumpur - Desa ParkCity
11	100322813	The Park Sky Residence @ Bukit Jalil City	2019.0	RM 2 600 per month	Kuala Lumpur - Bukit Jalil
12	100322809	PV9 Residences @ Taman Melati	2022.0	RM 2 000 per month	Kuala Lumpur - Setapak
13	100322802	Arte Plus Jalan Ampang	2018.0	RM 1 500 per month	Kuala Lumpur - Ampang
14	87950203	Maxim Citilights	2017.0	RM 1 300 per month	Kuala Lumpur - Sentul
15	100239196	Greenview Apartments		RM 1 000 per month	Kuala Lumpur - Kepong
16	100322320	Legasi Kampong Bharu	2020.0	RM 3 200 per month	Kuala Lumpur - KL City
17	100322311	Legasi Kampong Bharu	2020.0	RM 3 200 per month	Kuala Lumpur - KL City
18	100322212	Majestic Maxim	2021.0	RM 1 400 per month	Kuala Lumpur - Cheras
19	100273500	Desa Villas		RM 2 500 per month	Kuala Lumpur - Wangsa Maju
20	100231953	East Side One Ampang Avenue		RM 1 800 per month	Kuala Lumpur - Ampang
21	100322123	Majestic Maxim	2021.0	RM 1 099 per month	Kuala Lumpur - Cheras
22	100310024	Majestic Maxim	2021.0	RM 1 099 per month	Kuala Lumpur - Cheras
23	100310022	Majestic Maxim	2021.0	RM 1 199 per month	Kuala Lumpur - Cheras
24	100290514	Majestic Maxim	2021.0	RM 1 100 per month	Kuala Lumpur - Cheras
25	100267049	Sri Penara		RM 1 000 per month	Kuala Lumpur - Cheras
26	100201993	Prisma Cheras		RM 1 100 per month	Kuala Lumpur - Cheras
27	100295196	Menara Alpha		RM 1 400 per month	Kuala Lumpur - Wangsa Maju
28	100322088	Berlian Residence @ Setapak		RM 1 500 per month	Kuala Lumpur - Setapak
29	100322086	28 Dutamas	2017.0	RM 2 500 per month	Kuala Lumpur - Solaris Dutamas
30	100322100	28 Dutamas	2017.0	RM 2 900 per month	Kuala Lumpur - Solaris Dutamas
31	100322078	Pangsapuri Melur (Sentul)	2006.0	RM 1 750 per month	Kuala Lumpur - Sentul
32	100202252	Residensi Vista Wirajaya		RM 1 800 per month	Kuala Lumpur - Setapak
33	100136931	Bayu Sentul Condominium	2015.0	RM 1 900 per month	Kuala Lumpur - Sentul
34	100234716	Flexus Signature Suites		RM 1 500 per month	Kuala Lumpur - Jalan Kuching
35	100322047	Menjalara 18		RM 2 400 per month	Kuala Lumpur - Bandar Menjalara
36	93075352	G Residence @ Desa Pandan	2015.0	RM 4 500 per month	Kuala Lumpur - Desa Pandan
37	99876261	Greenpark	1999.0	RM 1 400 per month	Kuala Lumpur - Old Klang Road
38	100322029	Majestic Maxim	2021.0	RM 1 800 per month	Kuala Lumpur - Cheras
39	100322005	Baiduri Apartment (Desa Pandan)		RM 1 500 per month	Kuala Lumpur - Desa Pandan
40	98292007	Bennington Residences @ SkyArena	2019.0	RM 3 000 per month	Kuala Lumpur - Setapak
41	100321936	Sri Putramas	2004.0	RM 1 600 per month	Kuala Lumpur - Jalan Kuching
42	100321976	Putra Villa	2007.0	RM 1 800 per month	Kuala Lumpur - Setapak
43	99627362	38 Bidara		RM 2 100 per month	Kuala Lumpur - KLCC
44	99492392	3 Towers	2015.0	RM 2 500 per month	Kuala Lumpur - Ampang Hilir
45	100036659	One Maxim	2020.0	RM 1 300 per month	Kuala Lumpur - Sentul
46	99528357	Casa Mutiara	2007.0	RM 2 000 per month	Kuala Lumpur - Bukit Bintang
47	100321885	Sofiya Residences		RM 2 000 per month	Kuala Lumpur - Desa ParkCity
48	100273689	Laman Tasik	2002.0	RM 2 100 per month	Kuala Lumpur - Cheras
49	100033999	Residensi Seri Wahyu @ Jalan Kuching KL		RM 1 300 per month	Kuala Lumpur - KL City
50	100033960	The Hamilton	2020.0	RM 2 600 per month	Kuala Lumpur - Wangsa Maju

1. Previous Page
2. Next Page
3. Sorting column
4. Exit
Select an option >>

Pressing 1 will let the user to look at the apartment information with 50 properties in one page.



	ads_id	prop_name	completion_year	monthly_rent	location
51	99844667	3 Towers	2015.0	RM 2 000 per month	Kuala Lumpur - Ampang Hilir
52	100321850	Menjalara 18		RM 2 500 per month	Kuala Lumpur - Bandar Menjalara
53	100321860	The Havre @ Bukit Jalil	2020.0	RM 2 500 per month	Kuala Lumpur - Bukit Jalil
54	94858400			RM 2 000 per month	Kuala Lumpur - Setapak
55	93420010			RM 1 800 per month	Kuala Lumpur - Setapak
56	86821569			RM 1 700 per month	Kuala Lumpur - Jalan Kuching
57	94858340			RM 1 600 per month	Kuala Lumpur - Setapak
58	94861261			RM 2 000 per month	Kuala Lumpur - Jalan Kuching
59	94962913			RM 1 500 per month	Kuala Lumpur - Kepong
60	95081469			RM 1 700 per month	Kuala Lumpur - Jalan Kuching
61	95865588			RM 1 850 per month	Kuala Lumpur - Kepong
62	97361235			RM 1 700 per month	Kuala Lumpur - Sentul
63	97361227				
	RM 1 700 per month	Kuala Lumpur - Jalan Ipoh			
64	97407806			RM 1 800 per month	Kuala Lumpur - Kepong
65	100290798	M3 Residency	2017.0	RM 1 600 per month	Kuala Lumpur - Setapak
66	97006687			RM 1 600 per month	Kuala Lumpur - Kepong
67	100321689	Majestic Maxim	2021.0	RM 1 400 per month	Kuala Lumpur - Cheras
68	100321678	D'Puncak Suasana		RM 1 000 per month	Kuala Lumpur - KLCC
69	100321609	Kaleidoscope			
	RM 2 700 per month	Kuala Lumpur - Setiawangsa			
70	100321665	KL Traders Square Residences	2019.0	RM 1 400 per month	Kuala Lumpur - Gombak
71	100321664	Prisma Cheras			
	RM 1 500 per month	Kuala Lumpur - Cheras			
72	100321580	Seasons Garden Residences @ Wangsa Maju	2017.0	RM 1 350 per month	Kuala Lumpur - Wangsa Maju
73	100321562	Lavile	2021.0	RM 3 000 per month	Kuala Lumpur - Cheras
74	100321251	Maxim Citilights	2017.0	RM 1 300 per month	Kuala Lumpur - Sentul
75	100321351	Fajar Ria			
	RM 2 500 per month	Kuala Lumpur - Pantai			
76	100321196	Banyan Tree	2015.0	RM 7 000 per month	Kuala Lumpur - KLCC
77	99435019	PPAIM Bukit Jalil			
	RM 1 300 per month	Kuala Lumpur - Bukit Jalil			
78	100235971	Taman Connaught			
	RM 900 per month	Kuala Lumpur - Cheras			
79	100235642	Midah Ria			
	RM 1 300 per month	Kuala Lumpur - Cheras			
80	99774637	Sri Kenangan			
	RM 2 500 per month	Kuala Lumpur - Wangsa Maju			
81	100321040	Faber Ria	1985.0	RM 1 300 per month	Kuala Lumpur - Taman Desa

According to the choice provided, pressing 1 and 2 can help user to navigate between pages of properties information. The 3rd option is the sorting function, and the 4th options can exit to the main menu.

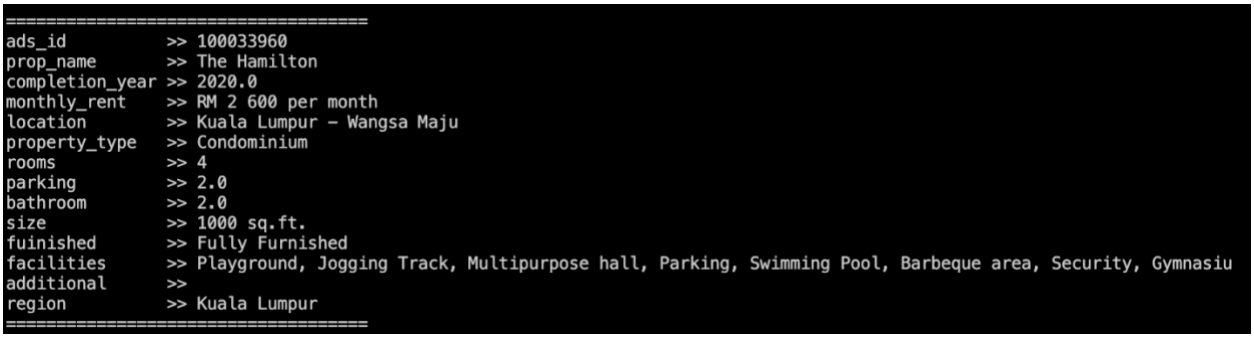
7.2.3 Sorting apartment information

```
1. Previous Page
2. Next Page
3. Sorting column
4. Exit
Select an option >> 3
Sorting Column >>
1.  ID
2.  Property Name
3.  Completion Year
4.  Monthly Rent
5.  Location
6.  Property Type
7.  Number of Rooms
8.  Number of Parking
9.  Number of Bathroom
10. Square Feet
11. Furnishing Status
12. Facilities
13. Additional Facilities
Sorting which column? >> 1
Sorting Method >>
1.  Quick Sort
2.  Merge Sort
With which method? >> 1
```

When choosing the 3 in the property information page, user can choose which column to be used in the sorting and the sorting algorithm to be used.

7.2.4 Searching apartment information

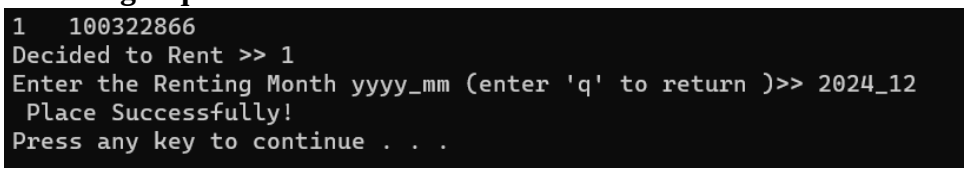
The searching apartment in the main menu will redirect user to the searching page and the user has to enter the apartment ID to search for the property details.



7.2.5 Add favorite property

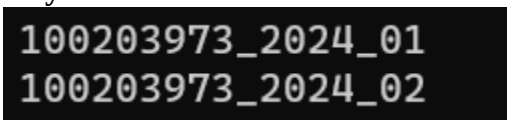
Choosing the add favorite property function in the main menu will ask the user to enter the property ID to be saved. After filling in the ID, the user will back to the main menu for further movement.

7.2.6 Place renting request



The number 4 choice will display the favorite property for the user to rent. Then, user should enter the given number to rent the desired property. Then, the user will be asked for the renting starting date. In this system, the date accepted is only starting from January 2024.

7.2.7 Property renting history



The 5th option in the tenant main menu is for user to check their renting history. The system will display the string as above. The format is property ID_renting start year_renting start month.

7.2.8 Confirm payment

```
index      id          year          month
1          100203973     2024     03
Enter the option that you want to pay >> 1
RM 2 300 per month
Confirm to pay?yes
 Successful Payment!
```

If the user chooses the confirm payment option, the system will show the property that has already rented by the users. The user will have to choose the property to make their option and pay for their monthly rental fees.

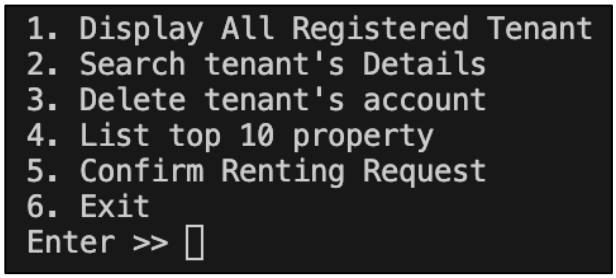
7.2.9 Tenant profile

```
================================================
ID                  U0001
Name                Edmund
Password            123456
Email               edmund@gmail.com
LastActive          10
Favorite Property   100322866
Renting Request     100203973_2024_01,100203973_2024_02
Renting Confirm     100203973_2024_03
Renting History     100203973_2024_02,100203973_2024_01
================================================
Press any key to continue . . .
```

The second last option in the main menu is for the user to check their profile information including ID, name, password, email, last active status, favorite property, and so on.

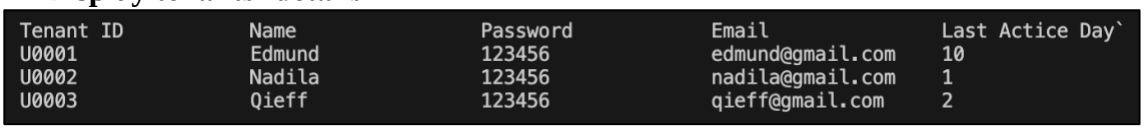
7.3 Manager

7.3.1 Manager's menu



The main menu for the user category "Manager" has six main functions, which it requires the user to key in a number to perform the function the user's desire.

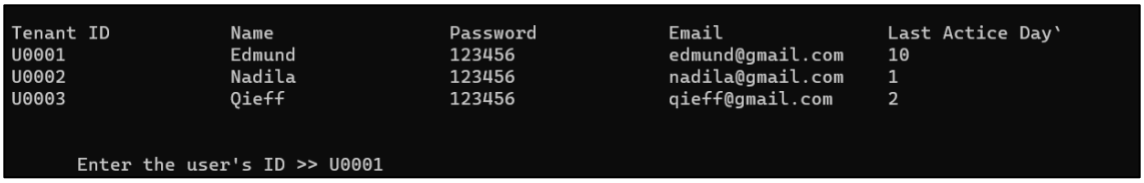
7.3.2 Display tenants' details



Tenant ID	Name	Password	Email	Last Actice Day`
U0001	Edmund	123456	edmund@gmail.com	10
U0002	Nadila	123456	nadila@gmail.com	1
U0003	Qieff	123456	qieff@gmail.com	2

By choosing option 1 in the Manager's menu, the user will be led to this interface which is a list of registered tenants.

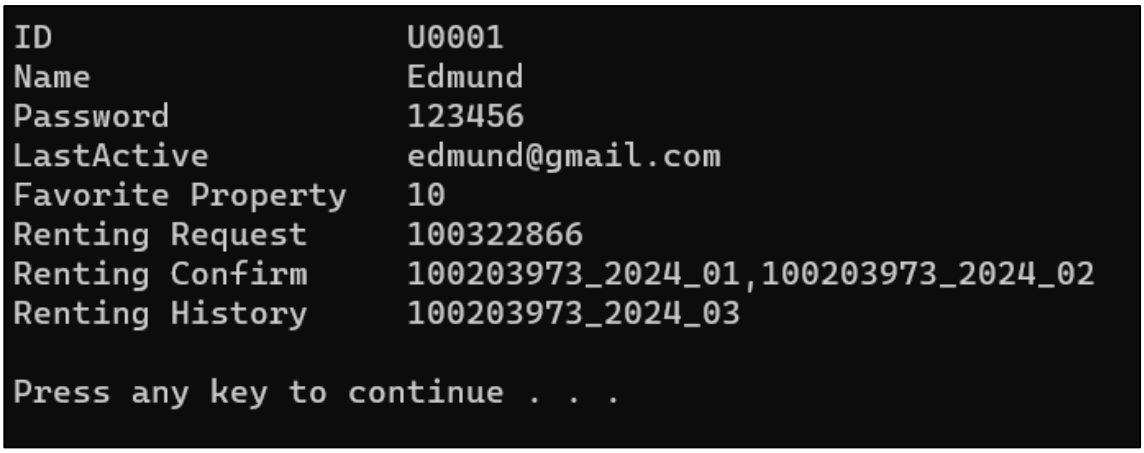
7.3.3 Search tenants' details



Tenant ID	Name	Password	Email	Last Actice Day`
U0001	Edmund	123456	edmund@gmail.com	10
U0002	Nadila	123456	nadila@gmail.com	1
U0003	Qieff	123456	qieff@gmail.com	2

Enter the user's ID >> U0001

By choosing option 2 in the Manager's menu, the user will be led to this interface which is a list of registered tenants, and the user is required to key in a tenant's ID if they wish to see further detailed information of the user.



The system will then showcase the full detail information on the chosen tenant.

7.3.4 Delete tenants' account

```
Tenant ID          Name               Password           Email              Last Actice Day`
U0001              Edmund             123456             edmund@gmail.com   10
U0002              Nadila             123456             nadila@gmail.com   1
U0003              Qieff              123456             qieff@gmail.com    2
Enter Tenant ID >>U0002
```

By choosing option 3 in the Manager's menu, the user will be led to this interface which is a list of registered tenants, and the user is required to input a tenant's id in order to delete their account from the system.

```
Are you want to removeU0002? (enter 'y' to proceed) >> y
```

The system will then ask a double confirmation whether they wish to delete the tenant's account.

```
Tenant ID          Name               Password           Email              Last Actice Day`
U0001              Edmund             123456             edmund@gmail.com   10
U0003              Qieff              123456             qieff@gmail.com    2
```

The tenant's account will then be deleted from the tenant's list.

7.3.5 Generate top 10 property report

```
============================
100322866       1  *
100322212       1  *
99263112        1  *
============================
Press any key to continue . . .
```

By choosing option 4 in Manager's menu, the user will be led to this interface which of the top 10 favorite property according to the tenants.

7.3.6 Manage tenancy process

```
new request :
U0001 : 100203973_2024_02
U0003 : 00203973_2024_03
Enter user's ID >> U0001
Option              ID                  Year                Month
1                   100203973           2024                02
Confirm ID option >> 1
```

By choosing option 5 in the Manager's menu, the user will be led to this interface which is a list of unapproved renting requests.

```
new request :
U0001 : 100203973_2024_01,100203973_2024_02
U0002 : 100203973_2024_01
U0003 : 100203973_2024_01,00203973_2024_03
Enter user's ID >> U0001
Option              ID                  Year                Month
1                   100203973           2024                01
2                   100203973           2024                02
Confirm ID option >> 1
 confirmed renting request successfully!
Press any key to continue . . .
```

The manager is then able to approve the desired renting request which will then be led to this interface.

7.4 Admin

7.4.1 Admin Menu

```
1. Add New Manager
2. Modify the manager's status
3. Delete tenant's account
4. Display Manager
5. Display Tenants
6. Property Information
7. Tenant Information
8. Exit
Enter >>
```

This is the admin menu displayed once an admin has logged into the system. The admin menu has 8 options, and each option can be chosen using the corresponding number input.

7.4.2 Add New Manager

```
New Manager ID >> M0004
Name >> Momo
Password >> 123456
Email >> momo@gmail.com
```

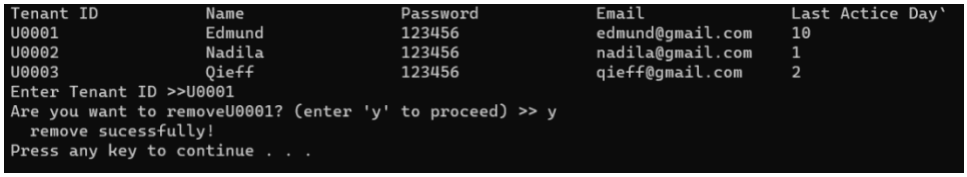
Choosing option 1 from the admin menu will allow the admin to add new managers. The admin can add new managers into the system by inputting the 3 required information, name, password, and email address. The manager ID is auto generated within the function.

7.4.3 Modify Manager Status

```
Manager ID      Name            Password        Email               Status
M0001           Mingliang       123456          mingliang@gmail.com active
M0002           Joel            123456          joel@gmail.com      inactive
M0003           Zahir           123456          zahir@gmail.com     active
M0004           Momo            123456          momo@gmail.com      active
ManagerID >>M0001
The status has turned into Inactive!
Press any key to continue . . .
```

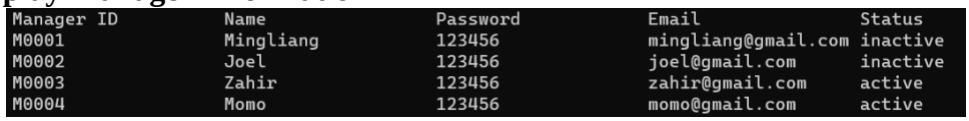
Choosing option 2 from the admin menu will allow the admin to modify a manager's status. The admin will have to input the manager ID and press enter to change the corresponding manager's status. Once done a message will be displayed to show that the manager's status has changed.

7.4.4 Delete Tenant Account



Choosing option 3 will allow the admin to delete a tenant's account form the system. The admin will have to input the tenant ID and press enter. The system will then show a confirmation message to the admin. The admin can either input "y" to complete the deletion process or any other key to cancel it. Once removed, the system will display another message to confirm the account deletion.

7.4.5 Display Manager Information

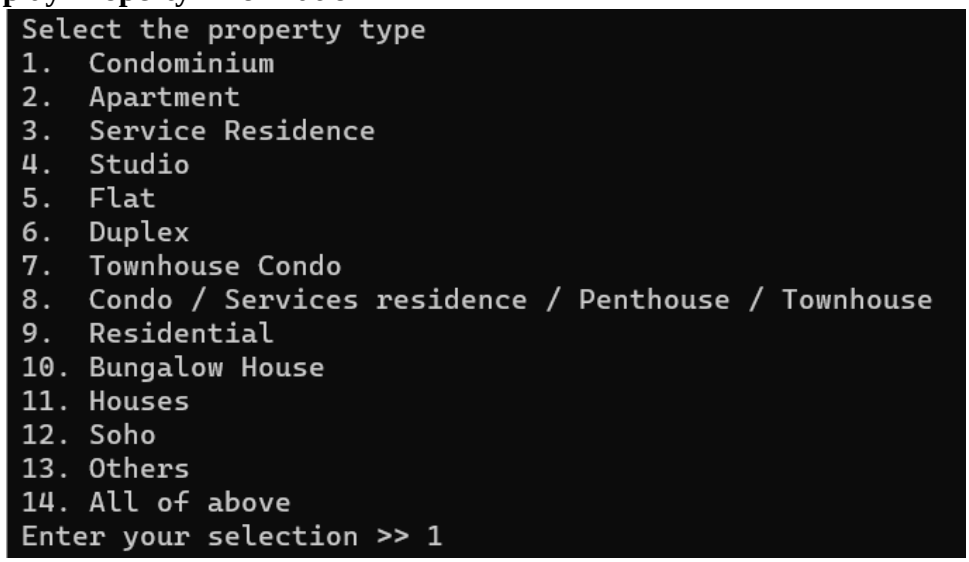


Choosing option 4 will display all the manager information to the admin such as ID, name, password, email, and status.

7.4.6 Display Tenants Information

Choosing option 5 will display all the tenant information to the admin such as ID, name, password, email, last active days, and status.

7.4.7 Display Property Information



Choosing option 6 will allow the admin to view the property information found in the database. The admin will first be prompted to specify the property type. The admin will have 14 different options for filtering. Once chosen, the admin will move on to the next section.

```
Select the property type
1.  Fully Furnished
2.  Partially Furnished
3.  Not Furnished
4. All of above
Enter your selection >> 1
```

This section will prompt the admin to select another filtering option by property type. The admin will have 4 options here and once chosen the corresponding properties according to the filtering chosen will be displayed.

```
Property Type: Condominium
Furnished    : Fully Furnished
Search >> as
        ads_id          prop_name                               completion_year     monthly_rent        location
        100231953       East Side One Ampang Avenue                                 RM 1 800 per month  Kuala Lumpur - Ampang
        100322100       28 Dutamas                              2017.0              RM 2 900 per month  Kuala Lumpur - Solaris Dutamas
        100321936       Sri Putramas                            2004.0              RM 1 600 per month  Kuala Lumpur - Jalan Kuching
        100273689       Laman Tasik                             2002.0              RM 2 100 per month  Kuala Lumpur - Cheras
        100321664       Prisma Cheras                                               RM 1 500 per month  Kuala Lumpur - Cheras
        16717632        Villa Wangsamas                         2010.0              RM 2 000 per month  Kuala Lumpur - Wangsa Maju
        99884824        Bayu Tasik 2                            1998.0              RM 1 500 per month  Kuala Lumpur - Cheras
        100318448       Mont Kiara Astana                       1998.0              RM 4 500 per month  Kuala Lumpur - Mont Kiara
        99906090        Bukit Pandan 2                                              RM 1 450 per month  Kuala Lumpur - Cheras
        100198517       Angkasa Impian 1                        2004.0              RM 4 000 per month  Kuala Lumpur - Bukit Bintang
        100317749       Casa Green (Bukit Jalil)                                    RM 2 000 per month  Kuala Lumpur - Bukit Jalil
        100317852       M Vertica                               2023.0              RM 2 200 per month  Kuala Lumpur - Cheras
        100317337       Sri Putramas                            2004.0              RM 1 600 per month  Kuala Lumpur - Jalan Kuching
        100317300       Sri Putramas                            2004.0              RM 1 500 per month  Kuala Lumpur - Jalan Kuching
        100316974       Royal Regent (Sri Putramas 3)           2013.0              RM 2 000 per month  Kuala Lumpur - Jalan Kuching
        100316599       Casa Kiara 1                                                RM 3 000 per month  Kuala Lumpur - Mont Kiara
        100218018       Nusa Mewah Villa Condominium            2014.0              RM 1 400 per month  Kuala Lumpur - Cheras
        100314405       Royal Regent (Sri Putramas 3)           2013.0              RM 1 800 per month  Kuala Lumpur - Jalan Kuching
        100313974       M Vertica                               2023.0              RM 2 600 per month  Kuala Lumpur - Cheras
        99579743        M Vertica                               2023.0              RM 2 300 per month  Kuala Lumpur - Cheras
        100312066       THE FENNEL AT SENTUL EAST               2017.0              RM 3 400 per month  Kuala Lumpur - Sentul
        100309250       Icon Residence (Mont Kiara)                                 RM 3 000 per month  Kuala Lumpur - Solaris Dutamas
        100309061       Changkat View Condominium               2009.0              RM 1 800 per month  Kuala Lumpur - Solaris Dutamas
        100308105       Casa Green (Bukit Jalil)                                    RM 2 000 per month  Kuala Lumpur - Bukit Jalil
        100307462       Casa Green (Bukit Jalil)                                    RM 1 800 per month  Kuala Lumpur - Bukit Jalil
        100307422       Majestic Maxim                          2021.0              RM 600 per month    Kuala Lumpur - Cheras
        99143617                                                                    RM 1 250 per month  Kuala Lumpur - Cheras
        100307233       Suasana Sentral Loft                    2008.0              RM 3 800 per month  Kuala Lumpur - KL Sentral
        98049852        Angkasa Condominiums                    2005.0              RM 1 400 per month  Kuala Lumpur - Cheras
        100305380       Icon Residence (Mont Kiara)                                 RM 2 600 per month  Kuala Lumpur - Solaris Dutamas
```

Here, the properties are displayed according to the filters chosen.

7.4.8 Display tenant Information

```
Search >> U000
Tenant ID           Name                Password            Email               Last Actice Day'
U0002               Nadila              123456              nadila@gmail.com    1
U0003               Qieff               123456              qieff@gmail.com     2
```

Choosing option 7 will allow the admin to search for a specific tenant within the system. Firstly, all the tenant information will be displayed. The admin can then search for the specified tenant by inputting the tenant ID using the search bar above.

8 Reflection

In the pursuit of completing the assignment, an array of limitations and constraints emerges, underscoring the intricacies of the task These limitations span through various dimensions, encompassing temporal, computational, and human resource considerations. This section explores the technical aspects of the limitations, providing insight into the complex relationship between the limitation and the assignment completion.

The first limitation and most significant challenge is the time constraint. The multifaceted challenge of the temporal constraint of 31 weeks is evident in its impact on different stages of the assignment process. The overarching urgency of meeting deadlines frequently compromises the thorough examination and improvement of approaches. The need to meet the given deadlines may limit the extent of in-depth research, preventing a thorough examination of the current system.

Besides the time constraint, the assignment faces a significant manpower constraint. The availability of skilled human resources greatly influences the pace and depth of work achievable within the defined timeframe. Limited human resources can hinder the comprehensive exploration of potential solutions and impede the refinement process necessary for optimal results. Adapting to manpower limitations while maintaining rigorous standards is crucial to ensure a robust assignment outcome.

Within the assignment duration, it is crucial to prioritize the optimization and rigorous testing of the algorithms. Testing and optimizing the algorithm involves investigating the best settings for the algorithms, continuously modifying the parameters, and evaluating the performance. The successful completion of this process within the designated timeframe necessitates the implementation of strategic prioritization techniques and a keen understanding of the trade-offs between conducting a thorough analysis and the time constraint.

Finally, in evaluating the capacity of the underlying system to handle the growth and changing datasets effectively. The necessity for continuous evaluation and adaptation arises from the need to effectively handle growing workloads while maintaining seamless operations of the system. Moreover, it is essential to implement advanced data handling techniques, especially as the dataset expands and undergoes significant changes. It is necessary to address potential performance issues that may arise due to the increasing volume of data.

All in all, the 31-week timeframe challenges through analysis, often prioritizing deadlines over exploration. Limited skilled manpower compounds the issue, hindering comprehensive solutions development. Striking a balance between resource allocation and rigorous standards becomes pivotal. Prioritizing algorithm optimization and testing remains crucial, demanding adept trade-offs. Lastly, evaluating the system’s capacity ever evolving datasets highlights continuous adaptation and advanced data handling. Navigating through these constraints showcases resilience and sparks innovation within limitations, transforming the assignment into a profound lesson in pragmatic problem-solving.

9 Future Work

The developers can make a few enhancements to the system in the future. Firstly, the user interface should be developed, and also, to let the system users have a clearer understanding of the property condition, there should have some features that allow the admin to attach the property images. A more visualizable way will be also helpful in promoting the property.

Secondly, integrating the ratings and reviews feature into the APH system. Ratings and reviews can help people know the property's advantages and disadvantages, like being near to the LRT station, far from school, and so on. Henceforth, the users can choose the property to rent or buy based on their needs. This feature also prevents renters from renting a property that has not met their expectations, causing a bad user experience. The reviews and ratings can be rated anytime; therefore, this feature will also encourage the owner to keep their property's quality to not get low ratings.

Instead of using data analysis and algorithms to predict the users' preferences, the system can also record the users' search history. Then, it can show the recently viewed property at the top of the list so that the users can easily find the property they are interested in. Meanwhile, using the history and top 10 property features, the system can also implement the advertisement feature that shows the widespread property and find similar properties that could be liked by the users. By making the system more user-friendly, the APH system can have high customer retention in the long run.

10 Conclusion

In conclusion, the implementation of different data structures and algorithms in the APH's renting system has significantly improved the system's functionality, scalability, maintainability, and user experience. The use of data structures and algorithms has allowed the system to reduce resource usage and time complexity resulting in a streamlined and efficient rental experience for users. By strategically implementing appropriate data structures and search and sorting algorithms, the system has achieved significant advantages contributing to its overall effectiveness.

The usage of data structures like singly, circular, and doubly circular linked lists, stack, queue, and dynamic arrays have collectively yielded manifold benefits. The combination of these data structures in the system has heightened memory efficiency. The dynamic allocation based on actual data has greatly minimized wastage and facilitated the expansion or contraction of storage. Moreover, the insertion and deletion process has been simplified due to the utilization of liked lists. The usage of stacks and queues has also empowered rapid and orderly data manipulation.

Furthermore, the system's efficiency significantly increases after the integration of both searching and sorting algorithms. The utilization of these algorithms has elevated the data retrieval accuracy and speed, enabling smooth handling of voluminous datasets, which in return provided a better and more seamless user experience.

All in all, the implementation of different data structures, search, and sorting algorithms has ushered APH's renting system into an era of optimized performance. As a result, the system stands as a testament to the strategic utilization of data structures and algorithms, underscoring their pivotal role in enhancing functionality and user satisfaction.

11 References

S, R. A. (2023, July 27). *What is Linear Search Algorithm | Time Complexity*. Simplilearn.com. https://www.simplilearn.com/tutorials/data-structure-tutorial/linear-search-algorithm

Terra, J. (2023, February 28). *Binary Search Algorithms: Overview, When to Use, and Examples*. Simplilearn.com. https://www.simplilearn.com/binary-search-algorithm-article

Neeraj. (2023). *Advantages and Disadvantages of Linked List*. Retrieved from PrepBytes: https://www.prepbytes.com/blog/linked-list/advantages-and-disadvantages-of-linked-list/#:~:text=A%20singly%20linked%20list%20is,that%20it%20requires%20less%20memory

Shreyasnaphad. (2023a). *Applications, Advantages and Disadvantages of Circular Linked List*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-circular-linked-list/

Shreyasnaphad. (2023b). *Applications, Advantages and Disadvantages of Circular Doubly Linked List*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-circular-doubly-linked-list/

Aayushi2402. (2023). *Applications, Advantages and Disadvantages of Stack*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/

Shreyasnaphad. (2023c). *Applications, Advantages and Disadvantages of Queue*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-queue/

Great Learning. (2021). *Map in C++*. Retrieved from Great Learning: https://www.mygreatlearning.com/blog/map-in-c/#advan

Pkthapa. (2023). *Program to implement Singly Linked List in C++ using class*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/program-to-implement-singly-linked-list-in-c-using-class/

SoftwareTestingHelp. (2023). *Circular Linked List Data Structure In C++ With Illustration*. Retrieved from SoftwareTestingHelp: https://www.softwaretestinghelp.com/circular-linked-list/

Geeksforgeeks. (2023a). *Introduction to Doubly Linked List – Data Structure and Algorithm Tutorials*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/

Geeksforgeeks. (2023b). *Introduction to Stack – Data Structure and Algorithm Tutorials*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/

Geeksforgeeks. (2023c). *Queue Data Structure*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/queue-data-structure/

Itsadityash. (2023). *Introduction to Map – Data Structure and Algorithm Tutorials*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/introduction-to-map-data-structure-and-algorithm-tutorials/

12 Workload Matrix Table

ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATON

CT007-3-2-DSTR

Student Coursework Workload Matrix

INTAKE: APU2F2111CS(DA)	Edmund Chen Siang Zuan	Nadila Binti Ahmad Shahrul Nizam	Neaw Aik Ka	Muhammad Aqieff bin Mohd Noh
	TP067345	TP067349	TP065116	TP058190

Group Component

	ASSIGNMENT COMPONENT	CONTRIBUTION PERCENTAGE	CONTRIBUTION PERCENTAGE	CONTRIBUTION PERCENTAGE	CONTRIBUTION PERCENTAGE	TOTAL %
1a	Proposal	10.00	30.00	30.00	30.00	100
1a	Tenant function	35.00	20.00	25.00	20.00	100
1a	Manager function	35.00	25.00	20.00	20.00	100
1b	admin function	35.00	20.00	20.00	25.00	100
1b	data structures	40.00	20.00	20.00	20.00	100
1b	searching algorithms	40.00	20.00	20.00	20.00	100
1b	sorting algorithms	40.00	20.00	20.00	20.00	100
1c	documentation	25.00	25.00	25.00	25.00	100
	Total Marks and Contribution	33%	23%	23%	23%	100%

Edmund Chen Siang Zuan	Nadila Binti Ahmad Shahrul Nizam	Neaw Aik Ka	Muhammad Aqieff bin Mohd Noh