Discipline Core

ALGORITHMS

CS4003

Course Project

Sheikh Muhammed Tadeeb (AU19B1014)

❖ Problem Statement:

Restaurants these days are always trying to find a way to satisfy the customers and at the same time they are facing an issue on what to cook on a particular day (this will be based on customer's previous data for a particular day) so that their profit could be maximum on that day and the amount of different meals they have cooked on that day should be consumed fully i.e. In such a way that no dish/ food item should be left for the next day or as a wastage.

For this I have designed a secured system where only the admin is having the access of entering the quantity of different food items/dishes and the profits on those dishes, plus what is the total quantity (in Kg's) of food they are selling on a particular day. (Note: - The quantity of different food items and their profit on a particular day is completely based on previous data collected by the restaurant).

After having all the details, the algorithm will calculate the profit on 1 kg of all the food items and then it will check which one is coming out to be the maximum. Now the algorithm will check that on this day what is the total kg's of food sold completely. Note: this will include all the food items and we don't know what is the quantity of each being sold. Further the algorithm will suggest the food items and their quantities to be cooked so that the profit could be maximum and the wastage/leftover could be minimum.

❖ Problem Statement with an Example:

Example: - Say, if a particular restaurant had been gathering the data for 6 months i.e. what type of customers they usually had on Sunday's, Monday's etc… and for that particular day how much quantity (in kg's) of each different dish/food item they are cooking and at the end of the day what amount (in Rs) are they having as profit on each dish/food item on that day and what is the total number of customers they are having on this day. The data for Sunday's is:

Average Kg's of different dishes	8 kg of Meat	10 kg of Chicken	15 Kg Rice	5 Kg Fish
Average Profit	2000 Rs/-	1000 Rs/-	500 Rs/-	200 Rs/-

1 Kg's of different dishes	1 kg of Meat	1 kg of Chicken	1 Kg Rice	1 Kg Fish
Average Profit	250 Rs/-	100 Rs/-	33 Rs/-	40 Rs/-

Now Total no's of people coming on Sundays = 100 people.

And based on research, an average person eats = 200 grams.

So, total if 100 persons are visiting the restaurant and each is eating 200 grams than total is: - 20 kgs.

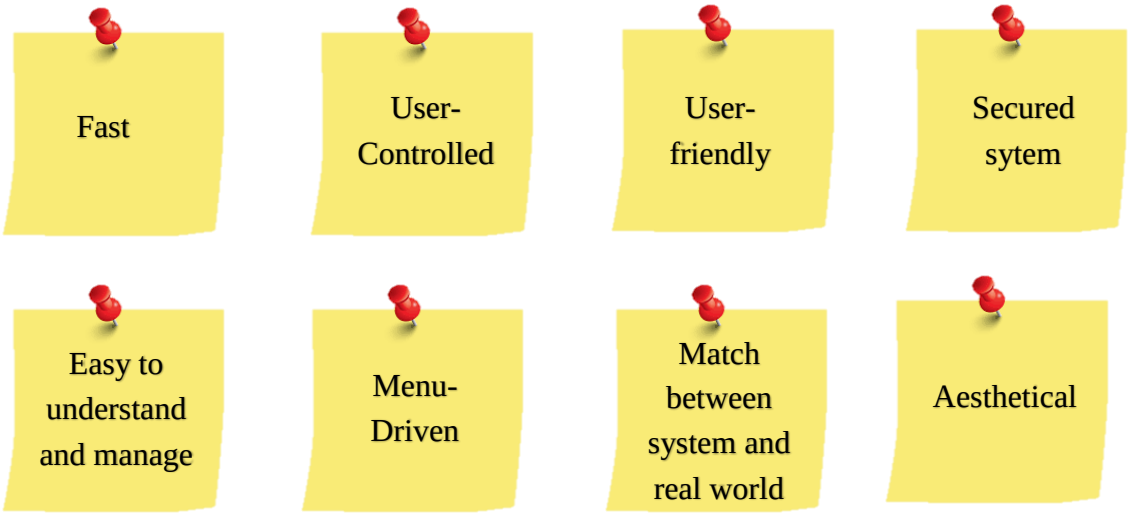
So, now we know that on Sunday's total of 20 kgs food is fully consumed in our restaurant, so if we make 20 kgs only. Hence we solved the problem of wastage of food or leftovers. Now we need to choose quantity of each food item in such a way that profit should be maximum and total quantity of all food items remains 20 kgs.

Eg: - Either I can choose 8kg meat and 16kg chicken, so total is 20 kgs but this is not an efficient solution. So how the efficiently the quantities should be chosen is designed in the algorithm.

❖ Constraints:

S.nos	Constraint
1.	The restaurant should have gathered the data of quantity of dish and the profit on that dish, plus the total quantity of food item sold on that day, and based on that data the result of the above algorithm will be generated.
2.	Doesn't considers customer fluctuation and customer mood i.e. it may happen sometimes that we get heavy demand of something which the algorithm didn't considered.
3.	The algorithm doesn't consider fluctuation in price of raw food items.
4.	Other minor costs like spices and ingredient cost is considered negligible here.

❖ Features:



❖ Pseudocode:

```
Defining structure for linked list {
        string name;
        Float AvgQuantity;
        Float AvgPrice;
        pointer next;
}
Class Parent {
        Declaring all member functions here
        Under Protected Access:
            Call () {
                Create_linked_lists() ← first call this function;
                Login() ← for security of the system;
                Greedy () ← calling function where actual implementation takes place;
                }
}
Parent: Create_linked_list () {
        Integer n;
        Pointer head, p of type list;
        cout<<" how many dishes you want to enter";
        Input ← n;
        For (i ← 0 to n with +1 increment)
        {
                If (i == 0) {
                        head ← address of 1st node of list;
                        p ← copy of address stored in head for future reference;
                } else {
                        next address node of p ← create new node;
                        change p to → next of p;
                }
```

cout<<" Enter average quantity (in kg) of dish 1;

Input << p → quantity ;

cout<<" Enter average profit (in kg) of dish 1;

Input << p → profit ;

Cout<<" Enter name of the food-item "; Input << p → name ;

}

Next of last p ← NULL;

Return address of head node;

}

Parent: Login () {

cout<<" Enter your username";

Input ← user ;

cout<<" Enter your password";

Input ← pass ;

It will call the → binary_search(user, pass) function to check if user is valid or not ;

}

Parent: Greedy () {

Display(passing head of list) ← calling function display to see the entered data;

Sum(passing head) ← to get the total weight and profit when wastage was occurring

Cout<<" Enter total number of people visiting your restaurant on this day";

Input ← total_nos_of_people ;

Catch1 ← Calculate_quantity (total_nos_of_people) ← calling function calculate quantity which could be fully consumed ;

Catch2 ← Calculate_profit_per_kg (passing head of list) ← calling function calculate profit per kg of food item ;

merge_sort (Catch2);

}

```
Parent:  Display (taking head of list) {
        while (until list is empty)
        {
                print (elements of list);
        }
}

Parent:  Calculate_profit_per_kg (passing head of list) {
        new_list of same type storing per kg profit;
        while (list is not empty)
        {
                if (1st node of list)
                {
                        new_list_head ← create new node;
                        p2 ← new_list_head copy ;
                } else {
                        next of p2 ← create new node;
                        p2 = next of p2;
                }
                p2 → name = p → name ;
                p2 → quantity = 1;
                p2 → profit = p → profit / p → quantity  ;
        }
        next of last p2 ← NULL;
        Return address of new_head node;
}
```

Parent: merge_sort (passing new_head of per kg profit list) {

If (stop if either head or head→next is not valid)

return head;

find the 1st and last node of list and divide the list in halfs left and right

merge_sort (left);

merge_sort (right);

finally calling the merge (left , right)

}

Parent: merge (taking left and right sub-lists) {

If left node is not present we will return the right one and vice-versa.

if (! Left) ***{***

return Right;

}

if (! Right) ***{***

return Left;

}

Check whose per kg price is lesser and make a new list by rearranging old one

Return head address of this list.

}

Parent: Calculate_quantity (total_nos_of_people) {

Assuming if 1 person eats → 200 grams ;

Then total_nos_of_people will eat → 200 × total_nos_of_people ;

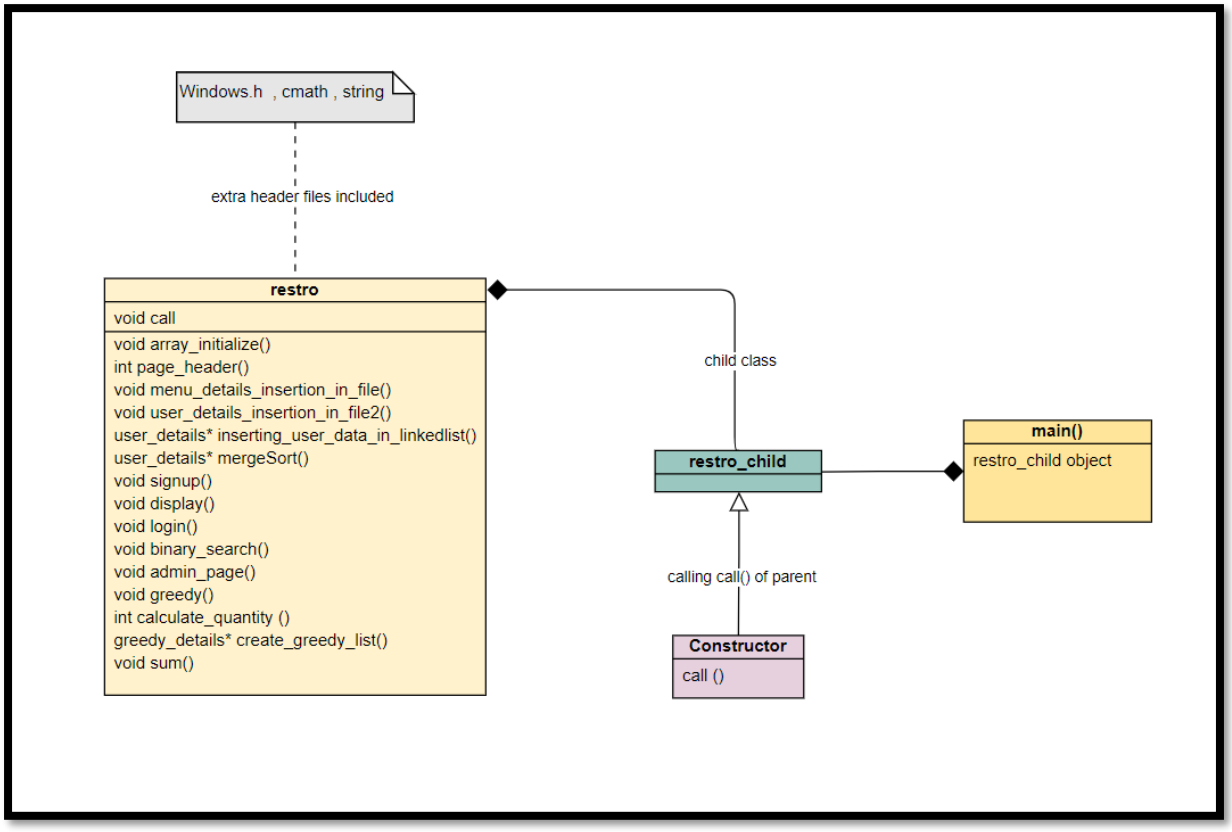
return *this weight (which will be fully consumed without wastage);*

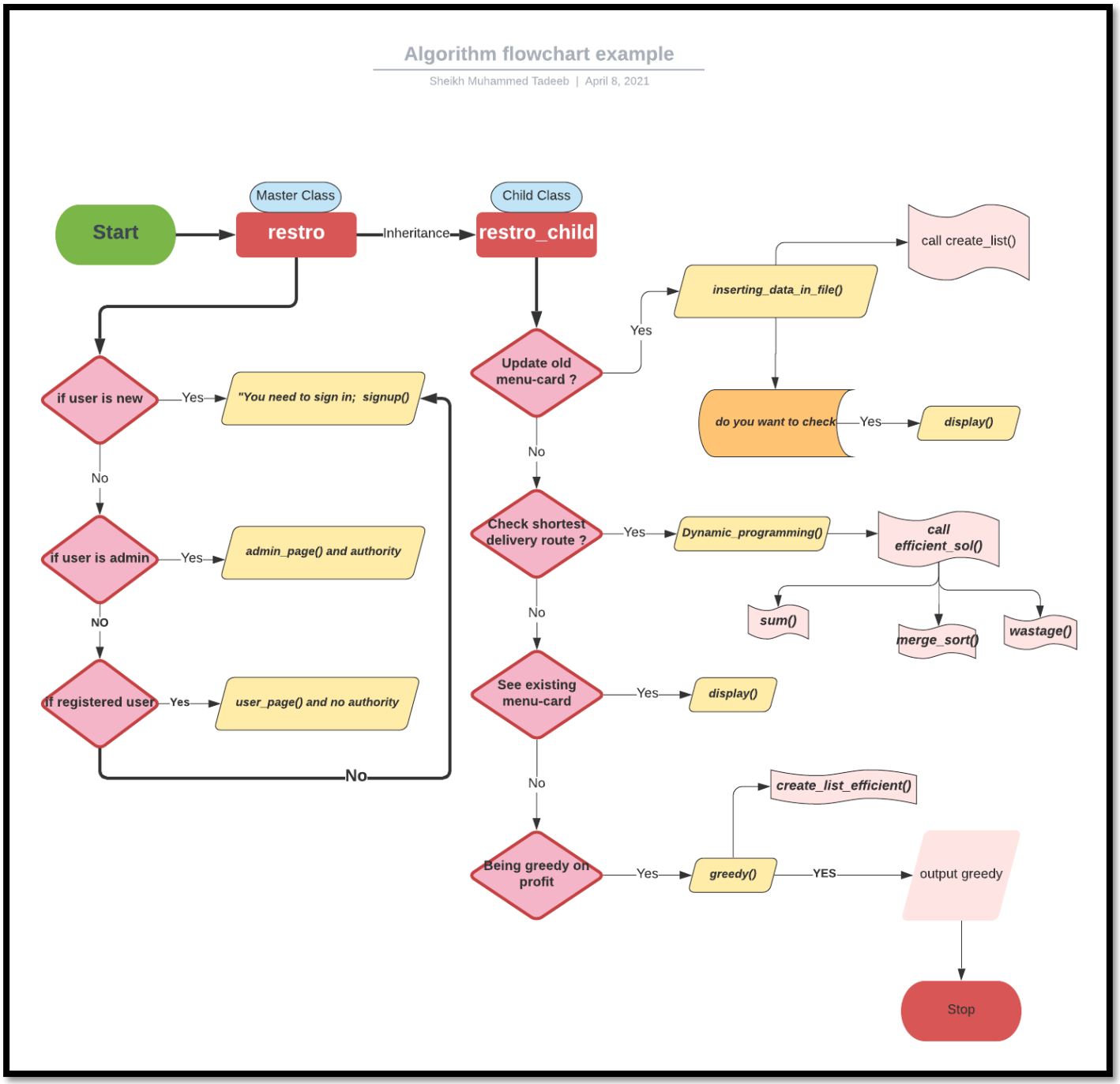
}

```
Parent:   sum (head of list) {
        // Initially considering weight and profit as 0.
        float wastage_weight = 0;
        float wastage_profit = 0;
        // Calculating weights and profits when wastage was there.
        while (greedy_head! = NULL) {
                wastage_weight = wastage_weight + greedy_head->quantity;
                wastage_profit = wastage_profit + greedy_head->profit;
                greedy_head = greedy_head->next;
        }
        cout<<endl<<endl<<"\t\t\t So the total quantity of food items =
        "<<wastage_weight<<" but this has lot of wastage";
        cout<<endl<<endl<<"\t\t\t And the total profit on all food items = cost price -
        selling price = "<<wastage_profit - 4000;
}

Class child of parent {
        Under public access:
                Constructor () {
                        Call ();
                }
}

int main () {
    creating object of child class;
     stopping main function;
}
```

```
Parent: efficient_choice (greedy_details* gh, float wastage_weight, int total_quantity) {
        Pointer variables* eh,*q;  bag ← 0;  i ← 0;
        while (gh! = NULL AND bag is empty) {
                Create head node and insert data in it;
                q->fname = gh->fname;
                q->percent = gh->percent;
                q->quantity = (q->percent / 100) * wastage_weight;
                q->profit = (q->quantity * gh->profit);
                bag = bag + q->quantity;
                gh = gh->next;
        }
        q->next = NULL;
        return eh;
}
```

❖ Class Diagram:




Algorithm flowchart example
Sheikh Muhammed Tadeeb | April 8, 2021
Master Class
Child Class
Start
restro
Inheritance
restro_child
call create_list()
inserting_data_in_file()
Yes
Update old menu-card ?
if user is new
Yes
"You need to sign in; signup()
do you want to check
Yes
display()
No
No
Check shortest delivery route ?
Yes
Dynamic_programming()
call efficient_sol()
if user is admin
Yes
admin_page() and authority
sum()
merge_sort()
wastage()
NO
No
See existing menu-card
Yes
display()
if registered user
Yes
user_page() and no authority
No
No
create_list_efficient()
Being greedy on profit
Yes
greedy()
YES
output greedy
Stop

❖ Testing:

1) Checking for the 1st constraint i.e. only admin is given authoritical access to the system.

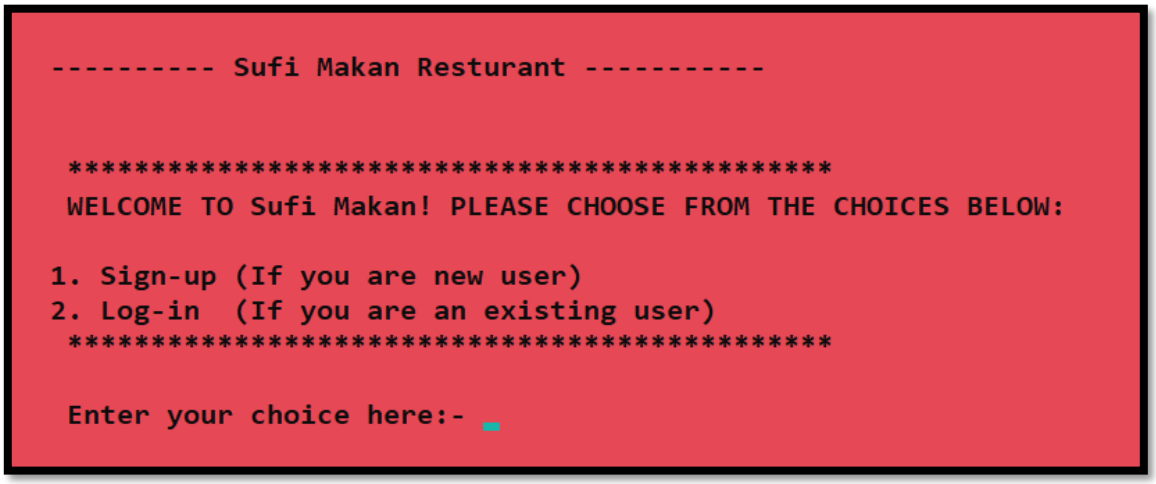
   ➢ Result:

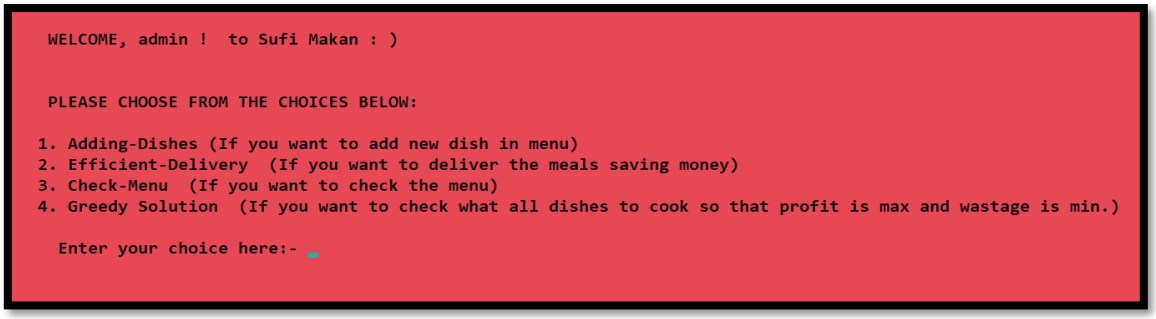
```
==============================================
LOG-IN BELOW:
==============================================

ENTER USERNAME: admin

ENTER PASSWORD: wrong_password


OOPS! SEEMS LIKE YOUR USERNAME OR PASSWORD IS WRONG!
LOADING...
```

```
==============================================
LOG-IN BELOW:
==============================================

ENTER USERNAME: admin

ENTER PASSWORD: password


CONGRATULATIONS! YOU ARE NOW LOGGED IN!
LOADING...
```

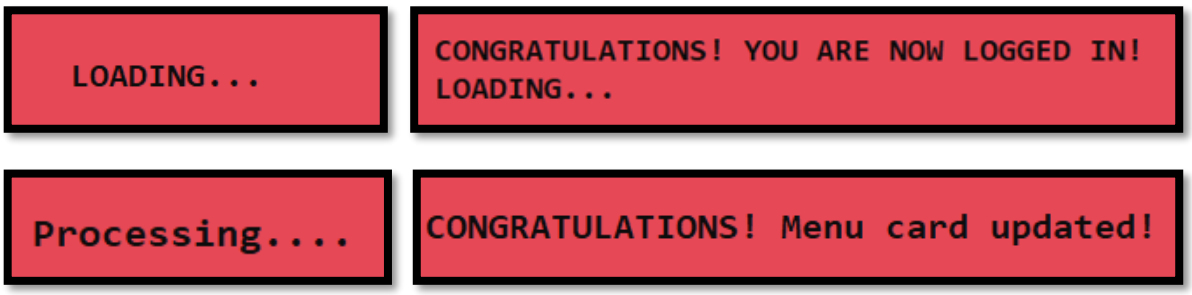
   ➢ Status:

2) Checking 2nd constraint i.e. system is user friendly and easy to access.





Clearly from above two figs we can see the system is menu driven i.e. giving user control and freedom



Clearly from above two figs we can see the system is well updated i.e. Visibility of system status could be found.

➢ Status:

3) Checking the 3rd constraint i.e. result generation based on recorded data.

```
Enter the number of dishes sold on Sunday (sunday is test case):- 5

Enter the name of food item 1 purchased :- meat

Enter meat's quantity sold (in Kg's):- 8

Enter meat's total profit earned (in Rs):- 2000

Enter meat's lovers crowd (in %):- 40
```

```
The inserted data has been saved and the data is:-

_____________________________________________________________

 Name      liker %      quantity(in Kgs)      profit(in Rs)
_____________________________________________________________

meat        40             8                    2000
chick       30             10                   500
rice        15             15                   500
fish        10             5                    700
paneer      5              20                   2400
```

```
-> So the total quantity of food items = 58 kg's
Note:- This includes lot of wastage food or lefover food

-> And the total profit on all food items = cost price - selling price = 2100  Rs

Enter total number of people visiting your restaurant on this day:- 200
```

```
The best way to make maximum profit and minimum wastage is by purchasing food items as given below:-

_____________________________________________________

 Name      liker %      quantity(in Kgs)      profit(in Rs)
_____________________________________________________

rice        15             8.7                  290
chick       30             17.4                 870
paneer      5              2.9                  348
fish        10             5.8                  812
meat        40             23.2                 5800
```

Clearly from above two figs we can see the system is taking user input and based on those input producing the results.

- Status:

4) Checking whether system is aesthetical or not.

The system has supports low-level aesthetics when we talk about C++ and hence could be considered aesthetical.

- Status:

5) The system is flexible i.e. if some new user wants to use the system then he she can easily signup and use the system.

```
================================================
              REGISTER BELOW:
================================================


ENTER FIRST NAME: Sheikj

ENTER LAST NAME: Tadeeb

ENTER USERNAME : _noorgul

ENTER PASSWORD : helloWorld
```

- Status:

❖ Data-Structure:

The data structure's used in the algorithm are: -

a. Linked-list:
They are used for saving the memory and if we need to do any update then it takes constant time to insert or delete a node from specified location. The second major reason to use linked list was that they are dynamic in nature i.e. at run time their size could be varied.

b. Array:
They are used as secondary data-type in the algorithm i.e. for keeping the address of corresponding linked-list nodes, they are used for simplicity and making the binary search efficient and fast, basically to reduce the overall time complexity of the code.

❖ Time Complexity of the system:

1) For Signup () function the details are going in the file so it takes constant time. i.e. $O(1)$.

2) For fetching this data in linked-list "while () loop" is used which will run until the end of the file. So if there are an element in the file the loop will run n times and hence the complexity will be: $O(n)$.

3) For login () function the user-input will take constant time i.e. O (1) but the mergeSort on string will take O (n×log n) and the binary search will take O (log n) and hence the total time complexity of login () function will be: $O(1) + O(n \times \log n) + O(\log n) = O(n \times \log n)$.

4) The efficient_delivery () function is working on the principle of dynamic programming and hence forming a minimum weight Hamilton cycle, since we are solving this using Dynamic Programming, we know that Dynamic Programming approach contains sub-problems. Here after reaching i^{th} node finding remaining minimum distance to that i^{th} node is a sub-problem. If we solve recursive equation we will get total $(n-1)\,2^{(n-2)}$ sub problems, which is $O(n2^n)$. Each sub-problem will take $O(n)$ time (finding path to remaining (n-1) nodes). Therefore, total time complexity is $O(n2^n) * O(n) = O(n^2 2^n)$.

5) In the greedy () function we are calling series of different function so the time complexity of this will be sum of time complexity of all the functions. i.e.

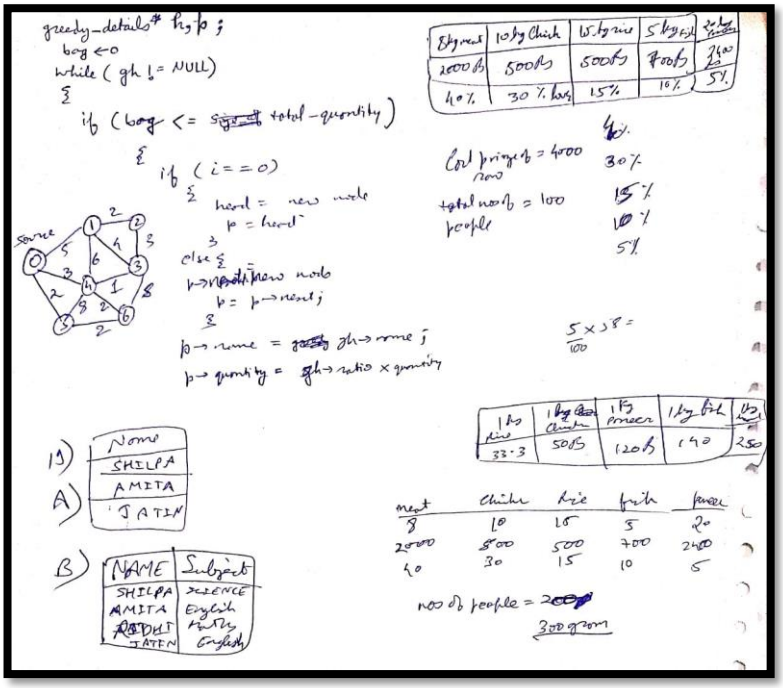
   a. Creating_linked_list () will take O (n).
   b. Display () will take O (n).
   c. Sum () will take O (n).
   d. Calcuate_quantity () will take O (1).
   e. Calculate_profit_per_kg () will take O (n).
   f. greedy_merge_sort () will take O (n×log n).
   g. efficient_choice () will take O (n).

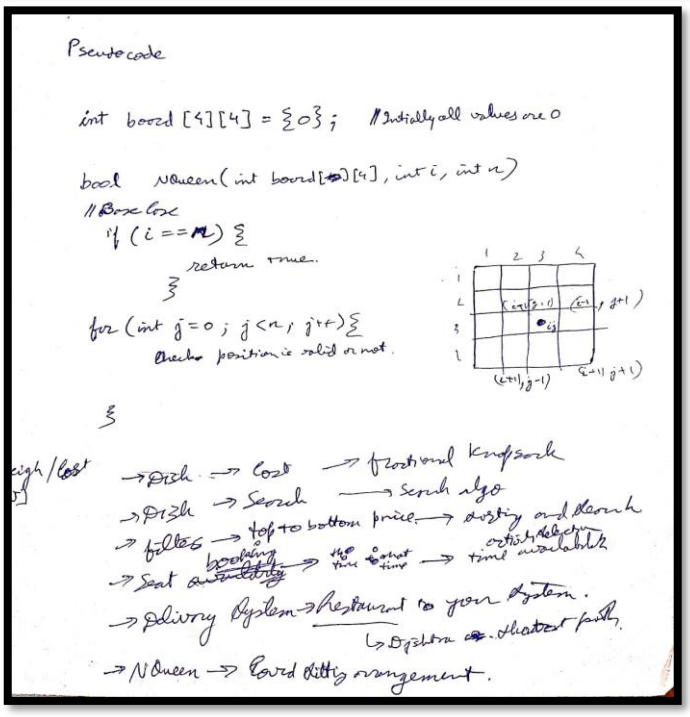
   Hence the total time complexity will be $O(n)$.

❖ Conclusion:

Algorithms are used in every part of computer science. They form the field's backbone. In computer science, an algorithm gives the computer a specific set of instructions, which allows the computer to do everything, be it running a calculator or running a rocket. Computer programs are, at their core, algorithms written in programming languages that the computer can understand. Computer algorithms play a big role in how social media works which posts show up, which ads are seen, and so on. These decisions are all made by algorithms. Programmers use algorithms to optimize searches, predict what users are going to type, and more. In problem-solving, a big part of computer programming is knowing how to formulate an algorithm.

❖ Paperwork:





❖ **Drive link (this is the code of my link):**

[code link (click here)]