Data Structures and Algorithms
Project Evaluation Sheet

Name: Soukhya Madan Nayak SRN: 02FE22BCS148
Roll:49

Implementation Analysis

Algorithm/Data Structure	Used? (Yes/No)	How and where?	Space Efficiency	Time Efficiency
Arrays	Yes	To store the details of the person	O(1)	O(1)
Structures and files	Yes	To store the information		
List				
Stack				
Queue				
Binary Tree				
Binary Search Tree				
AVL Tree				
2-3 Tree				
Red-Black Tree				
Trie				
Heap				
Lookup Table				
Sparse Table				
Fenwick Tree				
Segment Tree				
Skip List				
Union-Find				
Hashing				
DFS				
BFS				
Bubble Sort	Yes	To sorting based on shortest distance		O(n^2)
Selection Sort				
Insertion Sort				
Quick Sort	Yes	To sorting based on time		O(NlogN)
Merge Sort				
Brute Force String Search				
Rabin Karp				
Boyer-Moore				
Knuth-Morris-Pratt	Yes	Search the person		O(n+k)
Heap Sort				
Kruskal				
Prim				
Dijkstra	yes	To find shortest path		O(V^2)
Floyd				

Warshall				
Bellman-Ford				
Any Other				

Other Analysis:
Number of Lines of Code Written:

```
        #include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <limits.h>
#include <stdlib.h>

#define MAX_USERS 50
#define MAX_CUSTOMERS 50
#define FILENAME "user_data.txt"
#define CUSTOMER_FILENAME "customer_data.txt"

int n, i, j;

typedef struct info {
   char name[10];
   char dish[10];
   char location[10];
   int quantity;
   int delivery_time;
   int bill;
} info;

info a[MAX_CUSTOMERS];

// Number of vertices in the graph
int V;

int **graph;
int *dist;
typedef struct {
   char username[20];
   char password[20];
} User;

User users[MAX_USERS];
int numUsers = 0;

// Function declarations
void saveUserData();
void loadUserData();
void registerUser();
bool authenticate(char enteredUsername[], char enteredPassword[]);
void customerSwitchCases(char username[]);
void agentSwitchCases(info a[],int n);
void detailc(info a[]);
void saveCustomerData();
```

```
void loadCustomerData();
void dijkstra(info a[], int **graph, int src,int parent[]);
int minDistance(int dist[], int sptSet[]);
void printSolution(info a[], int dist[], int parent[]);
void search(info a[], int V);
void sort(info a[], int V, int dist[]);
void sortbyquantity(info a[], int V);
void sorttime(info a[], int V);
void sum(info a[], int n);
void quicksort(info a[], int low, int high);
void quicksort(info a[], int low, int high);


int main() {
   int r;
   char enteredUsername[20];
   char enteredPassword[20];

   // Load user data from file
   loadUserData();

   // Load customer data from file
   loadCustomerData();
printf("Do you want to register?");
   printf("\n1) YES enter 1 \n2)NO enter 2\n");
   scanf("%d",&r);
   switch(r){
   case 1: registerUser();
         break;
   case 2:printf("\n WELCOME \n");
   break;
   }

   // Get user input for login
   printf("Enter username: ");
   scanf("%s", enteredUsername);

   printf("Enter password: ");
   scanf("%s", enteredPassword);

   // Authenticate user
   if (authenticate(enteredUsername, enteredPassword)) {
      printf("Login successful. Welcome, %s!\n", enteredUsername);

      // Handle switch cases after successful login
      if (strcmp(enteredUsername, "agent") == 0) {
         agentSwitchCases(a,n);
      } else {
         customerSwitchCases(enteredUsername);
      }
   } else {
      printf("Login failed. Invalid username or password.\n");
   }
```

```
    // Save customer data to file
    saveCustomerData();

    // Free dynamically allocated memory
    for (int i = 0; i < V; i++) {
        free(graph[i]);
    }
    free(graph);
    free(dist);

    return 0;
}

void saveUserData() {
    FILE *file = fopen(FILENAME, "w");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    for (int i = 0; i < numUsers; i++) {
        fprintf(file, "%s %s\n", users[i].username, users[i].password);
    }

    fclose(file);
    printf("User data saved to %s.\n", FILENAME);
}

void loadUserData() {
    FILE *file = fopen(FILENAME, "r");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    while (fscanf(file, "%s %s", users[numUsers].username, users[numUsers].password) == 2) {
        numUsers++;
        if (numUsers >= MAX_USERS) {
            printf("Maximum number of users reached.\n");
            break;
        }
    }

    fclose(file);
}

void saveCustomerData() {
    FILE *file = fopen(CUSTOMER_FILENAME, "w");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    fprintf(file, "%d\n", n);
```

```
    for (int i = 1; i <= n; i++) {
        fprintf(file, "%s %s %s %d %d %d\n", a[i].name, a[i].dish, a[i].location,
a[i].bill,a[i].quantity,a[i].delivery_time);
    }

    fclose(file);
    printf("Customer data saved to %s.\n", CUSTOMER_FILENAME);
}

void loadCustomerData() {
    FILE *file = fopen(CUSTOMER_FILENAME, "r");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    fscanf(file, "%d", &n);
    for (i = 1; i <= n; i++) {
      fscanf(file, "%s %s %s %d %d %d", a[i].name, a[i].dish, a[i].location, &a[i].bill,
&a[i].quantity, &a[i].delivery_time);


    }

    fclose(file);
}

void registerUser() {
    printf("Enter a new username: ");
    scanf("%s", users[numUsers].username);

    printf("Enter a new password: ");
    scanf("%s", users[numUsers].password);

    numUsers++;
    printf("User registered successfully.\n");
    saveUserData();
}

bool authenticate(char enteredUsername[], char enteredPassword[]) {
    for (int i = 0; i < numUsers; i++) {
        if (strcmp(users[i].username, enteredUsername) == 0 && strcmp(users[i].password,
enteredPassword) == 0) {
            return true;  // Authentication successful
        }
    }
    return false;  // Authentication failed
}

void customerSwitchCases(char username[]) {
    int choice;
    while (1) {
        printf("\nCustomer Switch Cases:\n");
        printf("1. Customer name and order\n2. logout\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("List the orders\n");
                detailc(a);
                break;

            case 2:
            printf("logged out successfully");
            exit(0);
             break;

            default:
                printf("Invalid choice!\n");
                break;
        }
    }
}

void agentSwitchCases(info a[],int n) {
    int choice;
    while (1) {
        printf("\n\nAgent Switch Cases:\n");
        printf("1. View List of Orders\n");
        printf("2. Find Shortest Path\n");
        printf("3. Search place by name of customer\n");
        printf("4. display by Shortest Path\n");
        printf("5.display dish by quantity\n");
        printf("6.display by time\n");
        printf("7.Sales report\n");
        printf("8.logout\n");

        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("List of orders\n");
                printf("\n\tDETAILS\t\n");
                printf("\t*\t");
             loadCustomerData();
              printf("\n-----------------------------------------------------------------------------------------");
    printf("\nNAME\tDISH\tLOCATION\tBILL AMOUNT IN
(Rs)\tQUANTITY\tDELIVERY_TIME");
    printf("\n-----------------------------------------------------------------------------------------");
                for (i = 1; i <= n; i++) {
                    printf("\n%s\t%s\t%s\t\t\t%d\t\t\t%d\t\t\t%d", a[i].name, a[i].dish, a[i].location,
a[i].bill,a[i].quantity,a[i].delivery_time);
                }
                break;
```

```
        case 2:
           printf("Short path\n");
           printf("\n\tDISTANCE:\t\n");
           printf("\t*\t");
           printf("\n(Here 0 represents hotel-location)");
           //printf("Enter homes: ");
           //scanf("%d", &V);
           V=n;
           V = V + 1;
           graph = (int **)malloc(V * sizeof(int *));
           for (int i = 0; i < V; i++) {
              graph[i] = (int *)malloc(V * sizeof(int));
           }

           for (int i = 0; i < V; i++) {
              for (int j = 0;j < V; j++) {
                 printf("\nEnter the distance from %dth location to %dth location in (km)-: ", i,j);
                 scanf("%d", &graph[i][j]);
              }
           }

           for (int i = 0; i < V; i++) {
              for (int j = 0; j < V; j++)
                 printf("%d\t", graph[i][j]);
              printf("\n");
           }
             int *parent = (int *)malloc(V * sizeof(int));  // Initialize parent array
for (int i = 0; i < V; i++)
   parent[i] = -1;

dist = (int *)malloc(V * sizeof(int));
dijkstra(a, graph, 0, parent);
           break;

        case 3:
           printf("\nSearch\n");
           search(a, n);
           break;

        case 4:
           printf("\nSort\n");
           sort(a, V, dist);
           break;
        case 5:
           printf("\nsort by quantity\n");
           sortbyquantity(a, n);
           break;
        case 6:
           printf("\nsort by time\n");
           sorttime(a,n);
            break;
        case 7:

           sum(a,n);
```

```
            break;
        case 8:
            printf("logged out successfully");
            exit(0);
            break;

        default:
            printf("\nInvalid choice!\n");
            break;
      }
   }
}

void detailc(info a[]) {
   printf("\nEnter the number of customers: ");
   scanf("%d", &n);
   for (i = 1; i <= n; i++) {
      printf("\nEnter the details of customer-%d\n", i);
      printf("");
      printf("\n\nNAME-");
      scanf("%s", a[i].name);
      printf("\nDISH-");
      scanf("%s", a[i].dish);
      printf("\nLOCATION-");
      scanf("%s", a[i].location);
      printf("\nBILL AMOUNT IN (Rs)-");
      scanf("%d", &a[i].bill);
      printf("\nDISH QUANTITY-");
      scanf("%d", &a[i].quantity);
      printf("\nDELIVERY TIMING-");
      scanf("%d", &a[i].delivery_time);
   }
   // Save customer data to file after input
   saveCustomerData();
}

void dijkstra(info a[], int **graph, int src,int parent[]) {
   int sptSet[V];

   // Initialize all distances as INFINITE and sptSet[] as false
   for (int i = 0; i < V; i++) {
      dist[i] = INT_MAX;
      sptSet[i] = 0;
   }

   // Distance of the source vertex from itself is always 0
   dist[src] = 0;

   // Find the shortest path for all vertices
   for (int count = 0; count < V - 1; count++) {
      int u = minDistance(dist, sptSet);
      sptSet[u] = 1;

      for (int v = 0; v < V; v++) {
```

```
        if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
           dist[v] = dist[u] + graph[u][v];
            parent[v] = u;
        }
      }
   }

   // Print the constructed distance array
   printSolution(a, dist,parent);
}

int minDistance(int dist[], int sptSet[]) {
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++) {
      if (sptSet[v] == 0 && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }
   }

   return min_index;
}
void printPath(int parent[], int j) {
   // Base Case : If j is the source
   if (j >= 0)
      printPath(parent, parent[j]);

   if (j > 0)
      printf("->%d-home-%s", j, a[j].name);
   else
     printf("hotel->");
}


void printSolution(info a[], int dist[], int parent[]) {
   printf("\n-----------------------------------------------------");
   printf("\n\tWELCOME TO MENU BOARD");
   printf("\n\t**\n");
   printf("\n\t-DETAILS-\t\n");
   printf("\t-----------\t");
   printf("\n-----------------------------------------------------------------");
   printf("\nNAME\tDISH\tLOCATION\tBILL AMOUNT IN (Rs)\tQUANTITY\tTIME");
   printf("\n-----------------------------------------------------");
   for (i = 1; i <= n; i++) {
      printf("\n%s\t%s\t%s\t\t%d\t\t%d\t%d\n\n", a[i].name, a[i].dish, a[i].location,
a[i].bill,a[i].quantity,a[i].delivery_time);
   }
   printf("\nHOME \t\tShortest Distance from HOTEL IN (Km)");
   printf("\n----\t\t-----------------------------------\n");
   for (int i = 1; i <V; i++) {
      printf("\n%d-home-%s \t\t\t\t %d\n", i, a[i].name, dist[i]);
      printPath(parent, i);
      printf("\n");}}
```

```
void sort(info a[], int V, int dist[]) {
   int i, j;
   info t;

   for (i = 1; i <= V - 1; i++) {
      for (j = 1; j <= V - i - 1; j++) {
         if (dist[j] > dist[j + 1]) {
            // Swap the details of customers, including delivery_time
            t = a[j];
            a[j] = a[j + 1];
            a[j + 1] = t;

            // Swap the corresponding distances as well
            int temp = dist[j];
            dist[j] = dist[j + 1];
            dist[j + 1] = temp;
         }
      }
   }
   printf("\nSORTED LIST OF DISTANCE");
   printf("\n*");
   printf("\n---------------------------------------");
   printf("\nDISTANCE\tNAME\tLOCATION");
   printf("\n---------------------------------------");
   for (i = 1; i < V; i++) {
      printf("\n%d\t\t%s\t%s", dist[i], a[i].name, a[i].location);

}

}
   void sortbyquantity(info a[], int V){
   int i, j;
   info t;
   for (i = 1; i <= V - 1; i++) {
      for (j = 1; j <= V - i - 1; j++) {
         if (a[j].quantity > a[j + 1].quantity) {
            // Swap customers' details based on quantity
            t = a[j];
            a[j] = a[j + 1];
            a[j + 1] = t;
         }
      }
   }

   printf("\nSORTED LIST BY DISH QUANTITY");
   printf("\n*");
   printf("\n---------------------------------------");
   printf("\nQUANTITY\tDISH\tNAME\tLOCATION");
   printf("\n---------------------------------------");
   for (i = 1; i <=V; i++) {
      printf("\n%d\t\t%s\t%s\t%s", a[i].quantity, a[i].dish,a[i].name, a[i].location);

}
   }void sorttime(info a[], int V) {
```

```
   quicksort(a, 1, V);
   // Display the sorted list by delivery time
   printf("\nSORTED LIST BY DELIVERY TIME");
   printf("\n*");
   printf("\n---------------------------------------");
   printf("\nDELIVERY TIME\tNAME\tLOCATION\tDISH");
   printf("\n---------------------------------------");
   for (int i = 1; i <= V; i++) {
      printf("\n%d\t\t%s\t%s\t\t%s", a[i].delivery_time, a[i].name, a[i].location, a[i].dish);
   }
}

   void swap(info *a, info *b)
    {
   info temp = *a;
   *a = *b;
   *b = temp;
}



int partition(info a[], int low, int high) {
   int pivot = a[high].delivery_time;
   int i = low - 1;

   for (int j = low; j < high; j++) {
      if (a[j].delivery_time < pivot) {
         i++;
         swap(&a[i], &a[j]);
      }}

   swap(&a[i + 1], &a[high]);
   return i + 1;
}

void quicksort(info a[], int low, int high) {
   if (low < high) {
      int pi = partition(a, low, high);

      quicksort(a, low, pi - 1);
      quicksort(a, pi + 1, high);
   }
}


void sum(info a[], int n)
{
   int sum;
   for(i=1;i<=n;i++)
   {
     sum=sum+a[i].bill;
   }
   printf("\nTotal amount of sales price through out the day is Rs-%d",sum);
}
```

```
void buildLPS(char *pattern, int m, int *lps) {
   int len = 0;
   int i = 1;
   lps[0] = 0;

   while (i < m) {
      if (pattern[i] == pattern[len]) {
         len++;
         lps[i] = len;
         i++;
      } else {
         if (len != 0) {
            len = lps[len - 1];
         } else {
            lps[i] = 0;
            i++;
         }
      }
   }
}

void KMPSearch(char *text, char *pattern, info customer) {
   int n = strlen(text);
   int m = strlen(pattern);

   int *lps = (int *)malloc(sizeof(int) * m);
   buildLPS(pattern, m, lps);

   int i = 0;  // index for text[]
   int j = 0;  // index for pattern[]

   while (i < n) {
      if (pattern[j] == text[i]) {
         j++;
         i++;
      }

      if (j == m) {
         printf("Pattern found at index %d\n", i - j);
          printf("\nCustomer found:");
      printf("\n-------------------------------------------------------------");
      printf("\nNAME\tDISH\tLOCATION\tBILL AMOUNT IN (Rs)");
       printf("\n-------------------------------------------------------------");
     printf("\n%s\t%s\t%s\t\t%d\n", customer.name, customer.dish, customer.location,
customer.bill);

         j = lps[j - 1];
      } else if (i < n && pattern[j] != text[i]) {
         if (j != 0) {
            j = lps[j - 1];
```

```
        } else {
            i++;
        }
      }
   }

   free(lps);
}

void search(info a[], int n) {
   char pattern[10];
   printf("\nEnter the name to search: ");
   scanf("%s", pattern);

   for (int m = 1; m <= n; m++) {
      char *text = a[m].name;
      KMPSearch(text, pattern,a[m]);
   }
}
```

Number of Functions: 4

Design Techniques and Principles used: sorting, searching, listing, finding shortest path.