**School of Science and Engineering**

**Course Code: CSE 2202** **Course Title: Algorithms Lab**

**Section:02** **Semester: Spring 2024**

---

# Algorithm Lab Group Project Report

**Submitted to:**

Khairum Islam

Lecturer

Department of CSE

**Submitted by:**

Ayush Hassan Raiyan (223014015)

Sumaia Afroz Sayka (223014034)

Tasphia Islam(223014186)

Lamia Zaman (223014227)

## Introduction:

This C program presents an online shop management system that incorporates various algorithms for tasks such as product organization, order processing, and task scheduling. It offers insights into product details, order rankings, truck loading, factory task scheduling, and optimized delivery sequence using BFS.

## Problem Statement:

The task is to develop an online shop management system that efficiently handles product organization, order processing, and task scheduling. This involves managing product details, processing orders, optimizing delivery sequences, and scheduling factory tasks.

## Objective:

The objective is to create a robust system that streamlines online retail operations, improves order processing efficiency, optimizes delivery routes, and schedules factory tasks effectively.

## Problem Solution:

The solution involves implementing various algorithms such as knapsack for product loading, quicksort for order sorting, BFS for optimizing delivery sequences, and activity selection for scheduling tasks within the factory. These algorithms enable efficient management of product inventory, order processing, delivery logistics, and factory operations.

## Code:

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

// Product attributes
struct product {
    char name[30];
    int weight;
    int profit;
};

// Order attributes
struct order {
    char product_name[30];
    char rank[6]; // Increased size to accommodate 5 stars and null terminator
    int order_perday;
};

// Activity attributes
struct activity {
    char name[20];
    float start;
    float end;
};

// Queue Node
struct QueueNode {
    int data;
    struct QueueNode* next;
};

// Queue
struct Queue {
    struct QueueNode *front, *rear;
};

// Function to create a new queue node
struct QueueNode* newQueueNode(int data) {
    struct QueueNode* temp = (struct QueueNode*)malloc(sizeof(struct QueueNode));
    temp->data = data;
    temp->next = NULL;
    return temp;
}

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to check if the queue is empty
bool isEmpty(struct Queue* queue) {
    return queue->front == NULL;
}
```

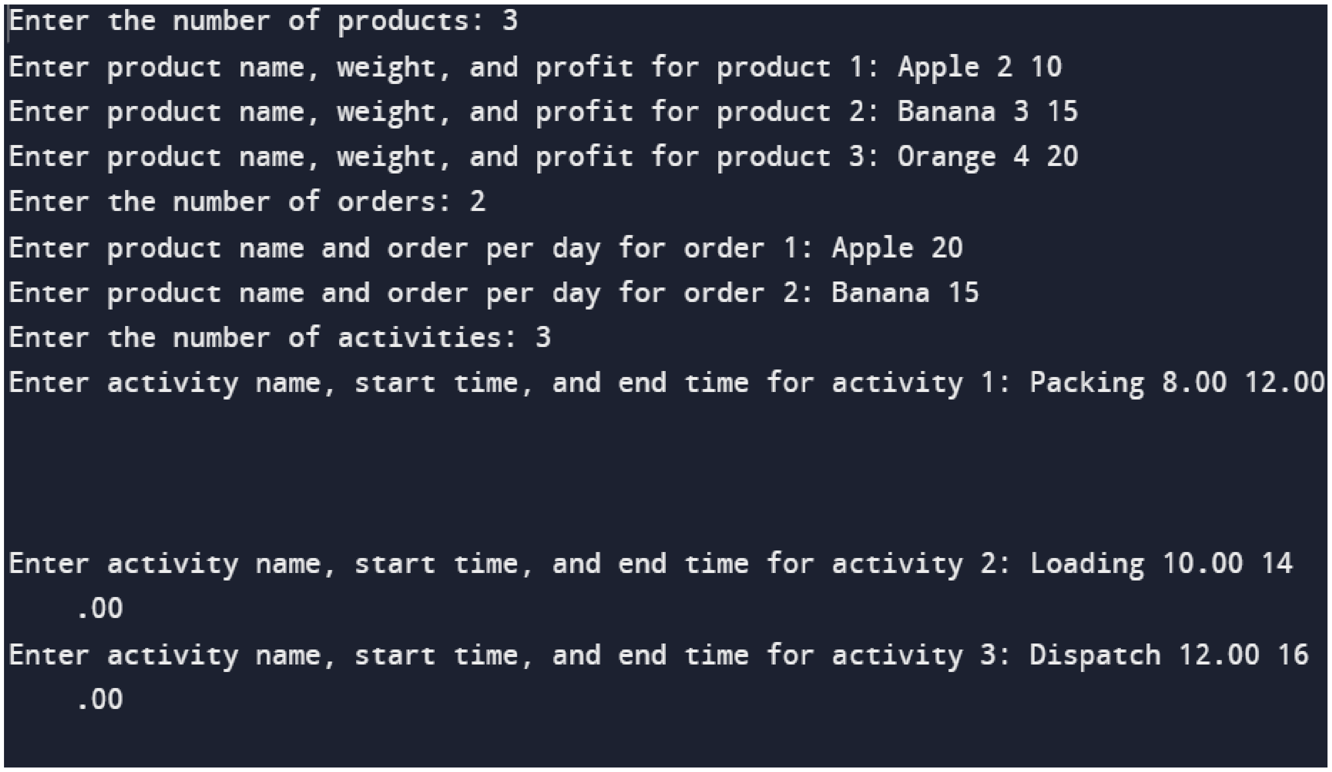
```
// Function to enqueue an item
void enqueue(struct Queue* queue, int data) {
    struct QueueNode* temp = newQueueNode(data);
    if (queue->rear == NULL) {
        queue->front = queue->rear = temp;
        return;
    }
    queue->rear->next = temp;
    queue->rear = temp;
}

// Function to dequeue an item
int dequeue(struct Queue* queue) {
    if (isEmpty(queue))
        return -1;
    struct QueueNode* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;
    if (queue->front == NULL)
        queue->rear = NULL;
    free(temp);
    return data;
}

// Function declarations
void quickSort(struct order arr[], int start, int end);
int partition(struct order arr[], int start, int end);
void assignRankStars(struct order arr[], int size);
void knapsack(struct product products[], int n, int capacity);
void selectActivities(struct activity activities[], int n, float openingTime, float closingTime);
void bfs(struct order orders[], int n);

// Sorting
void quickSort(struct order arr[], int start, int end) {
    if (start >= end)
        return;
    else {
        int pivotIndex = partition(arr, start, end);
        quickSort(arr, start, pivotIndex - 1); // L side
        quickSort(arr, pivotIndex + 1, end);   // R side
    }
}

int partition(struct order arr[], int start, int end) {
    int pivot = arr[end].order_perday;
    int i = start - 1;
    int j, temp;

    for (j = start; j <= end; j++) {
        if (arr[j].order_perday > pivot) {
            i++;
            // Swap elements
            struct order tempOrder = arr[i];
            arr[i] = arr[j];
            arr[j] = tempOrder;
        }
    }

    // Swap pivot with element at i+1
    struct order tempOrder = arr[i + 1];
    arr[i + 1] = arr[end];
    arr[end] = tempOrder;

    return i + 1;
}

// Function to assign stars to rank based on order_perday
void assignRankStars(struct order arr[], int size) {
    int i;
    for (i = 0; i < size; i++) {
        int stars = 5 - i; // 5 stars for the highest order_perday, 4 stars for the second highest, and so
on
        int j;
        for (j = 0; j < stars && j < sizeof(arr[i].rank) - 1; j++) {
            arr[i].rank[j] = '*';
        }
        arr[i].rank[j] = '\0'; // Null-terminate the rank string
    }
}

// Function to implement 0/1 knapsack algorithm
void knapsack(struct product products[], int n, int capacity) {
    int i, w;
    int K[n + 1][capacity + 1];

    // Build table K[][] in bottom-up manner
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (products[i - 1].weight <= w)
```

```
                K[i][w] = (products[i - 1].profit + K[i - 1][w - products[i - 1].weight] > K[i - 1][w])
                          ? products[i - 1].profit + K[i - 1][w - products[i - 1].weight]
                          : K[i - 1][w];
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // Print the selected products
    int res = K[n][capacity];
    w = capacity;
    printf("Selected products to load the truck:\n");
    for (i = n; i > 0 && res > 0; i--) {
        if (res != K[i - 1][w]) {
            printf("%s\n", products[i - 1].name);
            res -= products[i - 1].profit;
            w -= products[i - 1].weight;
        }
    }
}

// Function to select non-overlapping activities within the given time limit
void selectActivities(struct activity activities[], int n, float openingTime, float closingTime) {
    // Sort activities based on their end time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (activities[j].end > activities[j + 1].end) {
                struct activity temp = activities[j];
                activities[j] = activities[j + 1];
                activities[j + 1] = temp;
            }
        }
    }

    // Select activities
    printf("Work Schedule Within The Factory Open Hours:\n");
    printf("%-20s %-10s %-10s\n", "Activity Name", "Start Time", "End Time");
    float lastEndTime = openingTime;
    for (int i = 0; i < n; i++) {
        if (activities[i].start >= lastEndTime && activities[i].end <= closingTime) {
            printf("%-20s %-10.2f %-10.2f\n", activities[i].name, activities[i].start, activities[i].end);
            lastEndTime = activities[i].end;
        }
    }
}
```

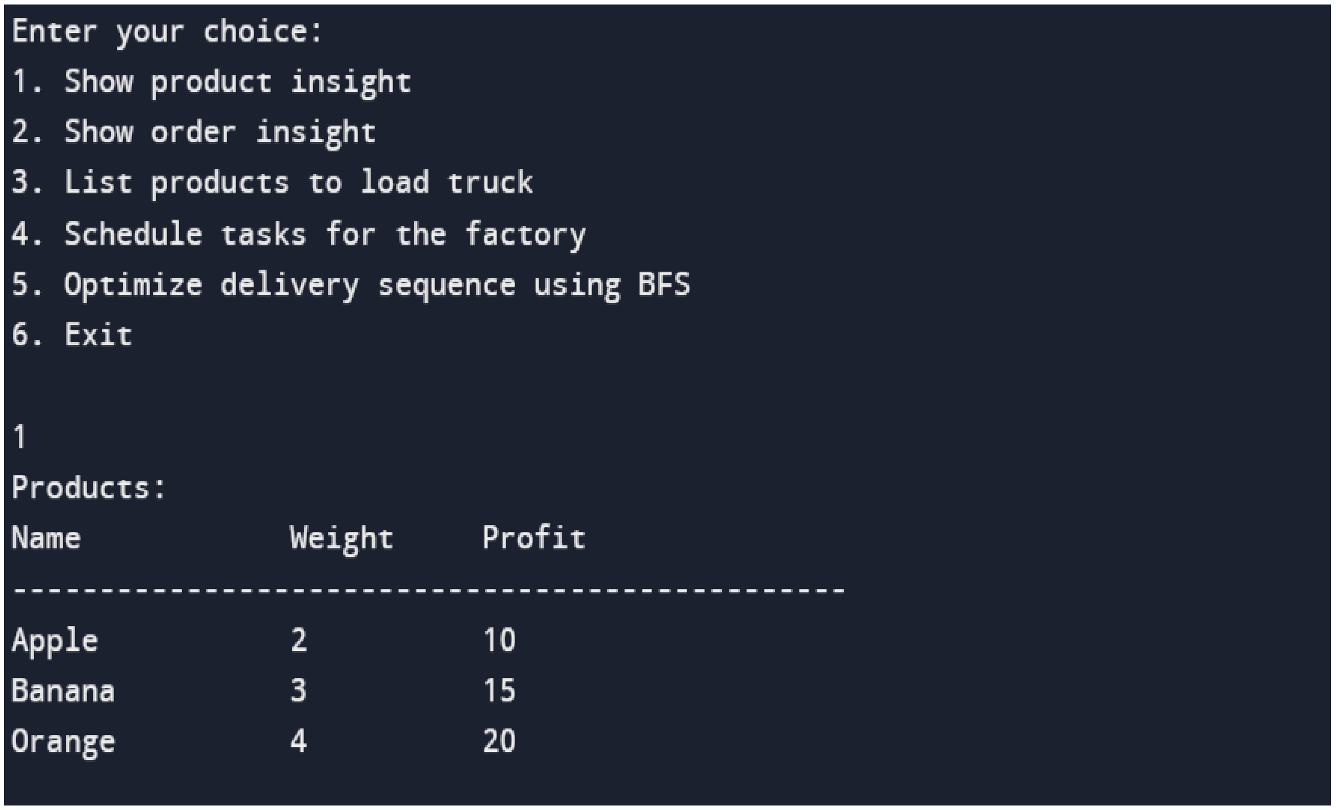
```
// Breadth-First Search for optimal delivery sequence
void bfs(struct order orders[], int n) {
    // Create a queue for BFS
    struct Queue* q = createQueue();

    // Enqueue all orders
    for (int i = 0; i < n; i++) {
        enqueue(q, i);
    }

    printf("Optimized delivery sequence:\n");

    // Dequeue and process orders
    while (!isEmpty(q)) {
        int idx = dequeue(q);
        printf("%s\n", orders[idx].product_name);
    }
}

int main() {
    int num_products, num_orders, num_activities;

    printf("Enter the number of products: ");
    scanf("%d", &num_products);
    struct product products[num_products];

    // Input products details
    for (int i = 0; i < num_products; i++) {
        printf("Enter product name, weight, and profit for product %d: ", i + 1);
        scanf("%s %d %d", products[i].name, &products[i].weight, &products[i].profit);
    }

    printf("Enter the number of orders: ");
    scanf("%d", &num_orders);
    struct order orders[num_orders];

    // Input order details
    for (int i = 0; i < num_orders; i++) {
        printf("Enter product name and order per day for order %d: ", i + 1);
        scanf("%s %d", orders[i].product_name, &orders[i].order_perday);
    }

    printf("Enter the number of activities: ");
    scanf("%d", &num_activities);
    struct activity activities[num_activities];
    // Input activity details
    for (int i = 0; i < num_activities; i++) {
        printf("Enter activity name, start time, and end time for activity %d: ", i + 1);
        scanf("%s %f %f", activities[i].name, &activities[i].start, &activities[i].end);
    }

    int choice, capacity;
    float openingTime, closingTime;
    do {
        printf("\nEnter your choice:\n");
        printf("1. Show product insight\n");
        printf("2. Show order insight\n");
        printf("3. List products to load truck\n");
        printf("4. Schedule tasks for the factory\n");
        printf("5. Optimize delivery sequence using BFS\n");
        printf("6. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                // Print table header for products
                printf("Products:\n");
                printf("%-15s %-10s %-10s\n", "Name", "Weight", "Profit");
                printf("------------------------------------------------------\n");
                // Print each product's details
                for (int i = 0; i < num_products; i++) {
                    printf("%-15s %-10d %-10d\n", products[i].name, products[i].weight,
products[i].profit);
                }
                break;

            case 2:
                // Sort orders based on order_perday
                quickSort(orders, 0, num_orders - 1);

                // Assign stars to rank based on order_perday
                assignRankStars(orders, num_orders);

                // Print table header for orders
                printf("Orders:\n");
                printf("%-15s %-10s %-15s\n", "Product Name", "Rank", "Order Per Day");
                printf("------------------------------------------------------\n");
                // Print each order's details
                for (int i = 0; i < num_orders; i++) {
                    printf("%-15s %-10s %-15d\n", orders[i].product_name, orders[i].rank,
orders[i].order_perday);
```

```
            }
            break;

        case 3:
            // Load truck with products
            printf("Enter the capacity of the truck: ");
            scanf("%d", &capacity);
            printf("\n");
            knapsack(products, num_products, capacity);
            break;

        case 4:
            // Select non-overlapping activities within a time range
            printf("Enter the opening time: ");
            scanf("%f", &openingTime);
            printf("Enter the closing time: ");
            scanf("%f", &closingTime);
            printf("\n");
            selectActivities(activities, num_activities, openingTime, closingTime);
            break;

        case 5:
            printf("Optimize delivery sequence using BFS\n");
            bfs(orders, num_orders);
            break;

        case 6:
            printf("Exiting...\n");
            break;

        default:
            printf("Invalid choice.\n");
            break;
      }
   } while (choice != 6);

   return 0;
}
```

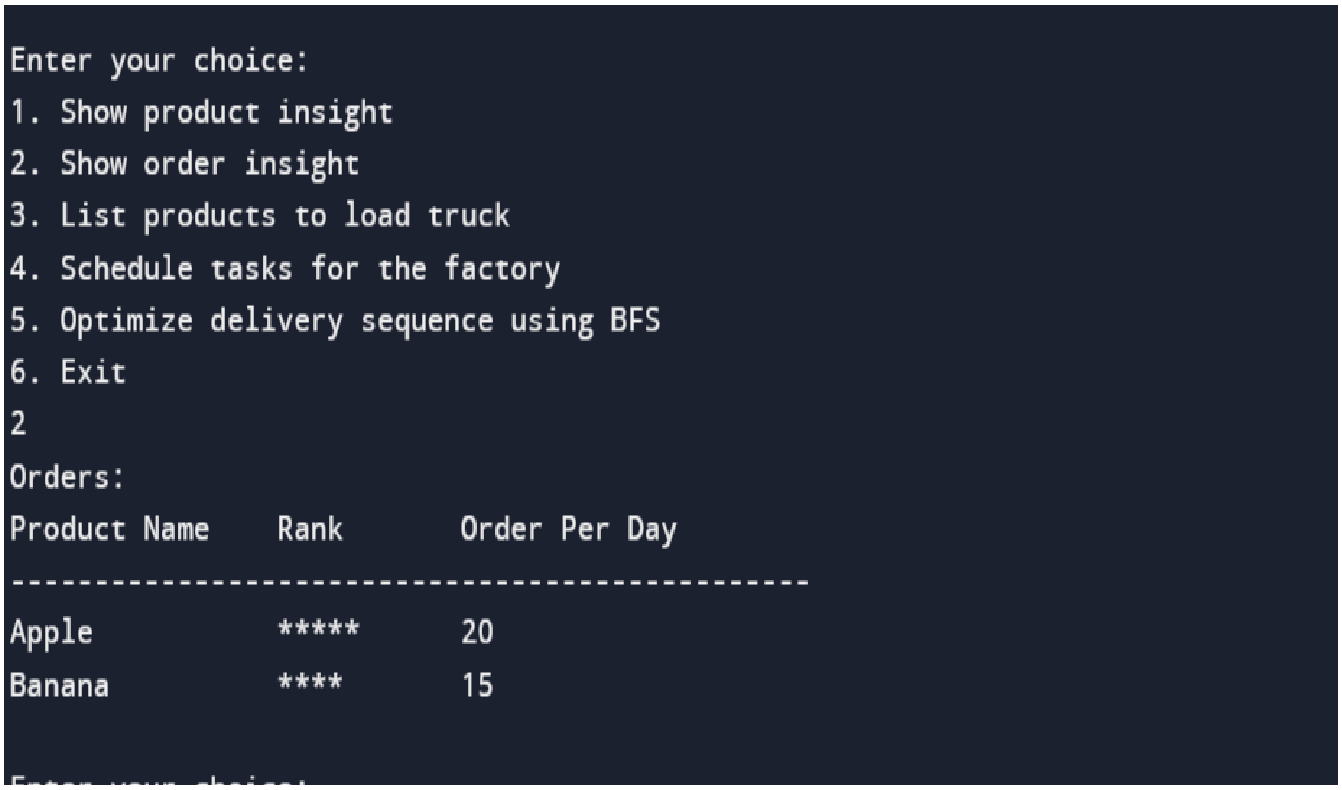
## Output of the Code:



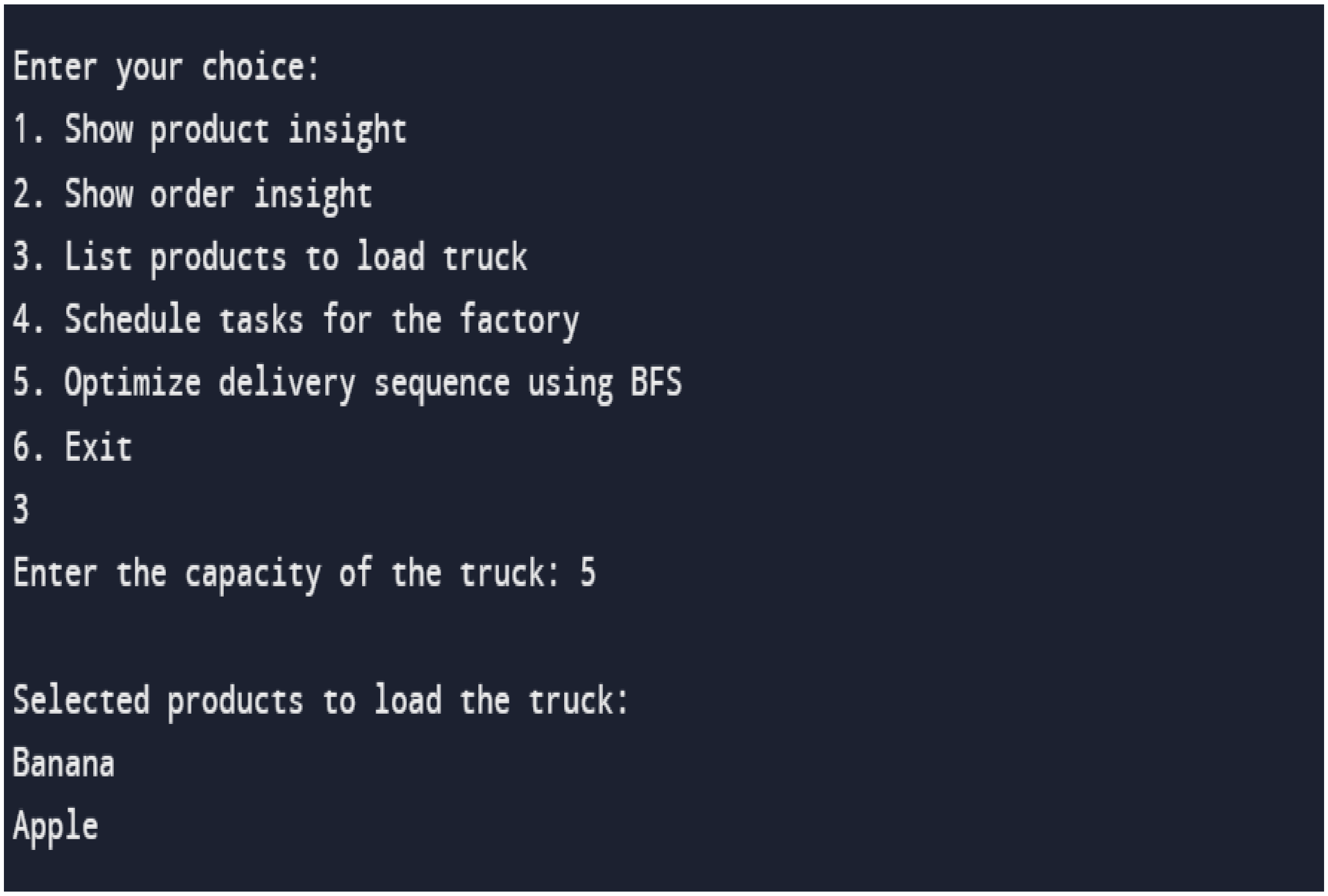
```
Enter the number of products: 3
Enter product name, weight, and profit for product 1: Apple 2 10
Enter product name, weight, and profit for product 2: Banana 3 15
Enter product name, weight, and profit for product 3: Orange 4 20
Enter the number of orders: 2
Enter product name and order per day for order 1: Apple 20
Enter product name and order per day for order 2: Banana 15
Enter the number of activities: 3
Enter activity name, start time, and end time for activity 1: Packing 8.00 12.00



Enter activity name, start time, and end time for activity 2: Loading 10.00 14
    .00
Enter activity name, start time, and end time for activity 3: Dispatch 12.00 16
    .00
```



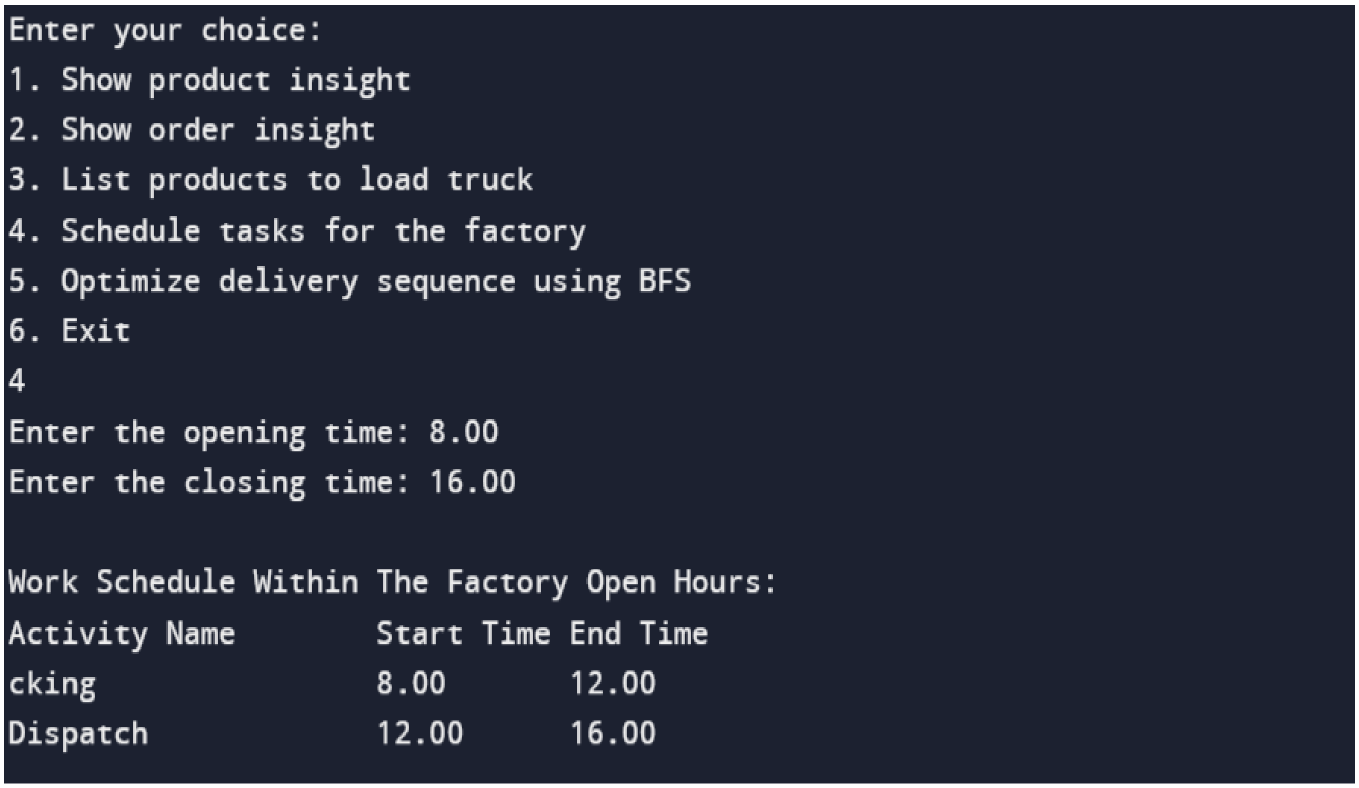
```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit

1
Products:
Name            Weight     Profit
------------------------------------------------
Apple           2          10
Banana          3          15
Orange          4          20
```



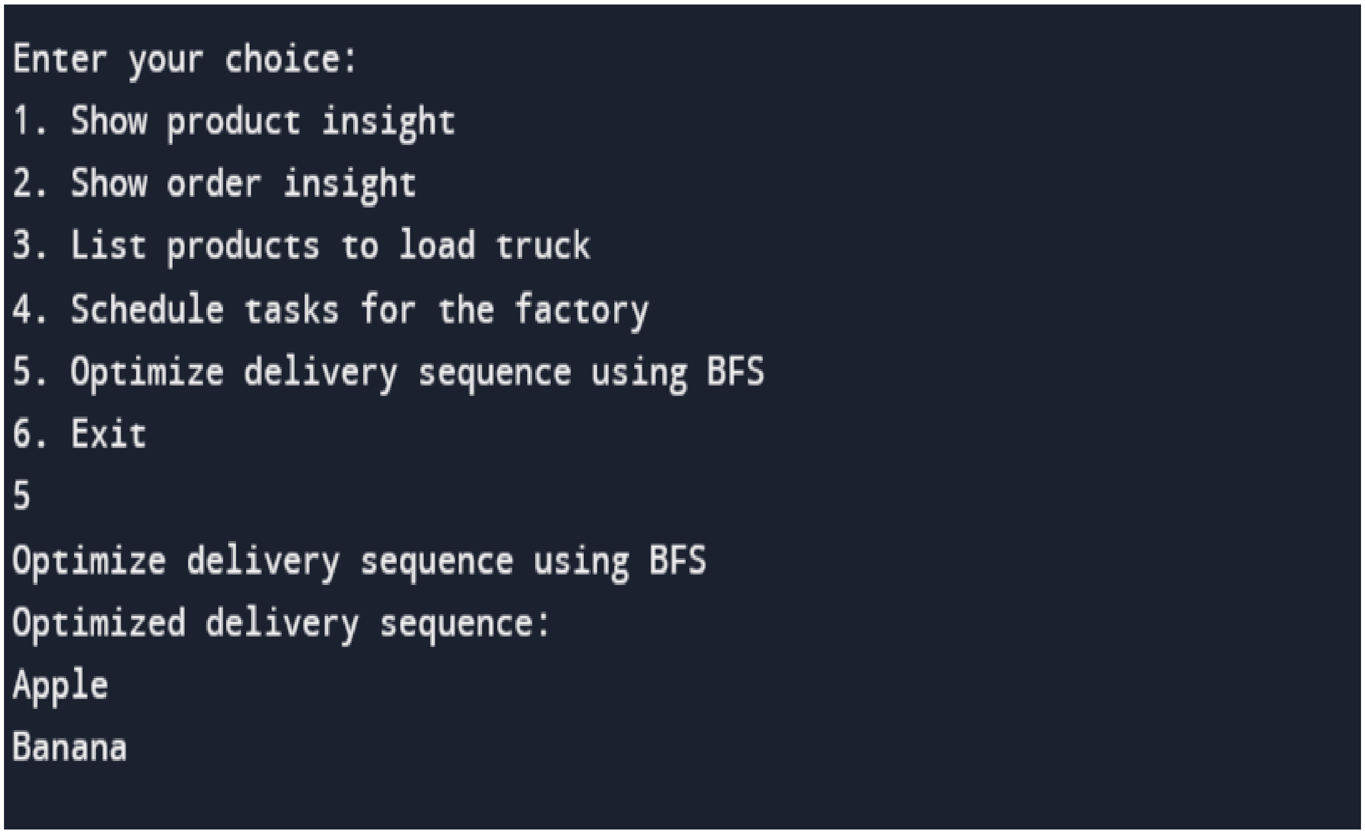
```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit
2
Orders:
Product Name    Rank       Order Per Day
------------------------------------------------
Apple           *****      20
Banana          ****       15
```



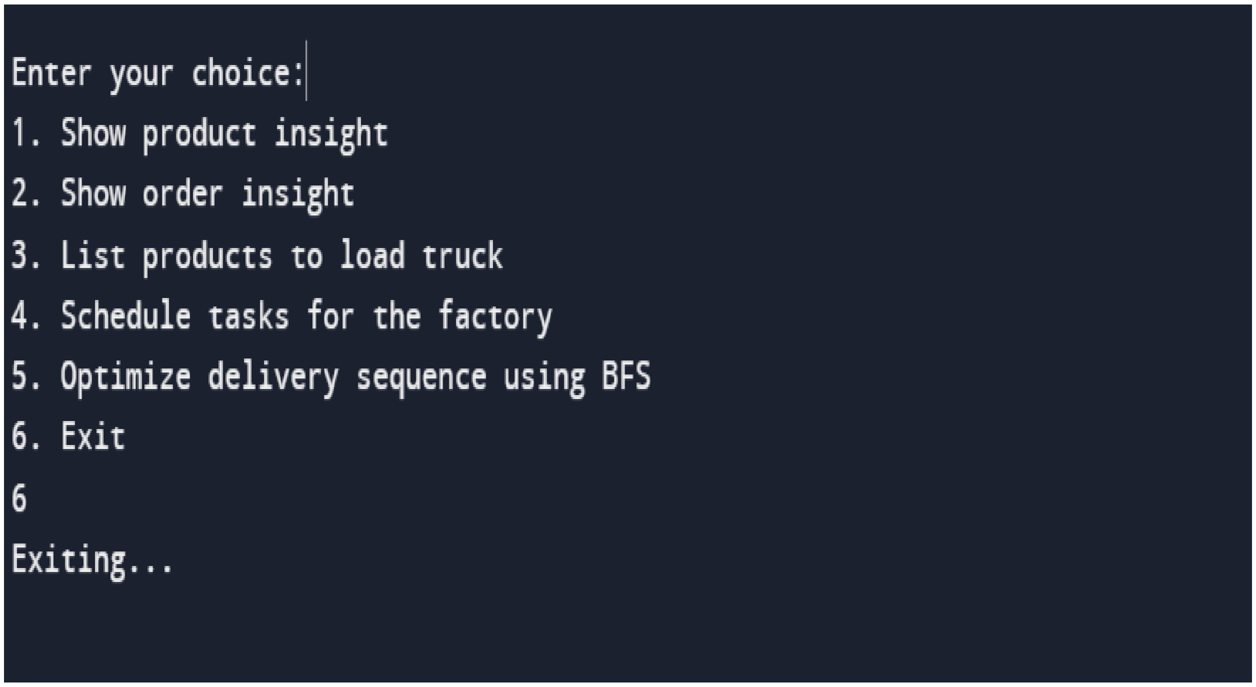
```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit
3
Enter the capacity of the truck: 5

Selected products to load the truck:
Banana
Apple
```



```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit
4
Enter the opening time: 8.00
Enter the closing time: 16.00

Work Schedule Within The Factory Open Hours:
Activity Name        Start Time End Time
cking                8.00       12.00
Dispatch             12.00      16.00
```



```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit
5
Optimize delivery sequence using BFS
Optimized delivery sequence:
Apple
Banana
```


```
Enter your choice:
1. Show product insight
2. Show order insight
3. List products to load truck
4. Schedule tasks for the factory
5. Optimize delivery sequence using BFS
6. Exit
6
Exiting...
```

## Algorithms Used:

**Knapsack Algorithm:** Used for loading products onto trucks efficiently based on weight and profit considerations.

**Quicksort Algorithm:** Employed to sort orders based on their daily demand.

**BFS (Breadth-First Search):** Utilized to optimize delivery sequences by finding the shortest path.

**Activity Selection Algorithm:** Used for scheduling factory tasks within specified time constraints.

## Future Work:

Here are some potential future improvements of the project:

**Integration with real-time data:** Incorporate live sales data to dynamically adjust delivery routes and task schedules based on changing demand patterns.

**Enhanced optimization algorithms:** Implement more advanced algorithms to further improve resource allocation and optimize delivery routes for specific geographical regions.

**Integration with other systems**: Integrate with inventory management systems and route optimization software to streamline operations and enhance overall efficiency.

**Customer-centric features:** Develop features such as real-time order tracking and delivery notifications to improve customer experience and satisfaction.

**AI and machine learning:** Explore the use of AI and machine learning techniques to predict demand, optimize inventory levels, and personalize product recommendations for customers.

## Contribution List:

| | | |
|---|---|---|
| Ayush Hassan Raiyan (223014015) | - | 25% |
| Sumaia Afroz Sayka (223014034) | - | 25% |
| Tasphia Islam(223014186) | - | 25% |
| Lamia Zaman (223014227) | - | 25% |

## Conclusion:

The developed online shop management system provides a comprehensive solution for efficiently managing various aspects of online retail operations, addressing key challenges in product organization, order processing, and task scheduling. By employing algorithms like knapsack, quicksort, BFS, and activity selection, the system optimizes resource utilization, improves order processing efficiency, and enhances overall operational effectiveness.