CSC4202

DESIGN AND ANALYSIS ALGORITHM

QUICKDROP PROJECT

Presented By Super Saiyan

The Group Members

Zul aiman

211512

Arif Aiman

211981

Diyana

211641

PROBLEM SCENARIO

In a bustling metropolitan area, a drone delivery service company, "QuickDrop," operates to ensure packages are delivered promptly and efficiently. Each delivery job has a deadline (by when it must be completed) and a profit associated with it. the company aims to maximize its total profit by optimally scheduling deliveries within their respective deadlines.

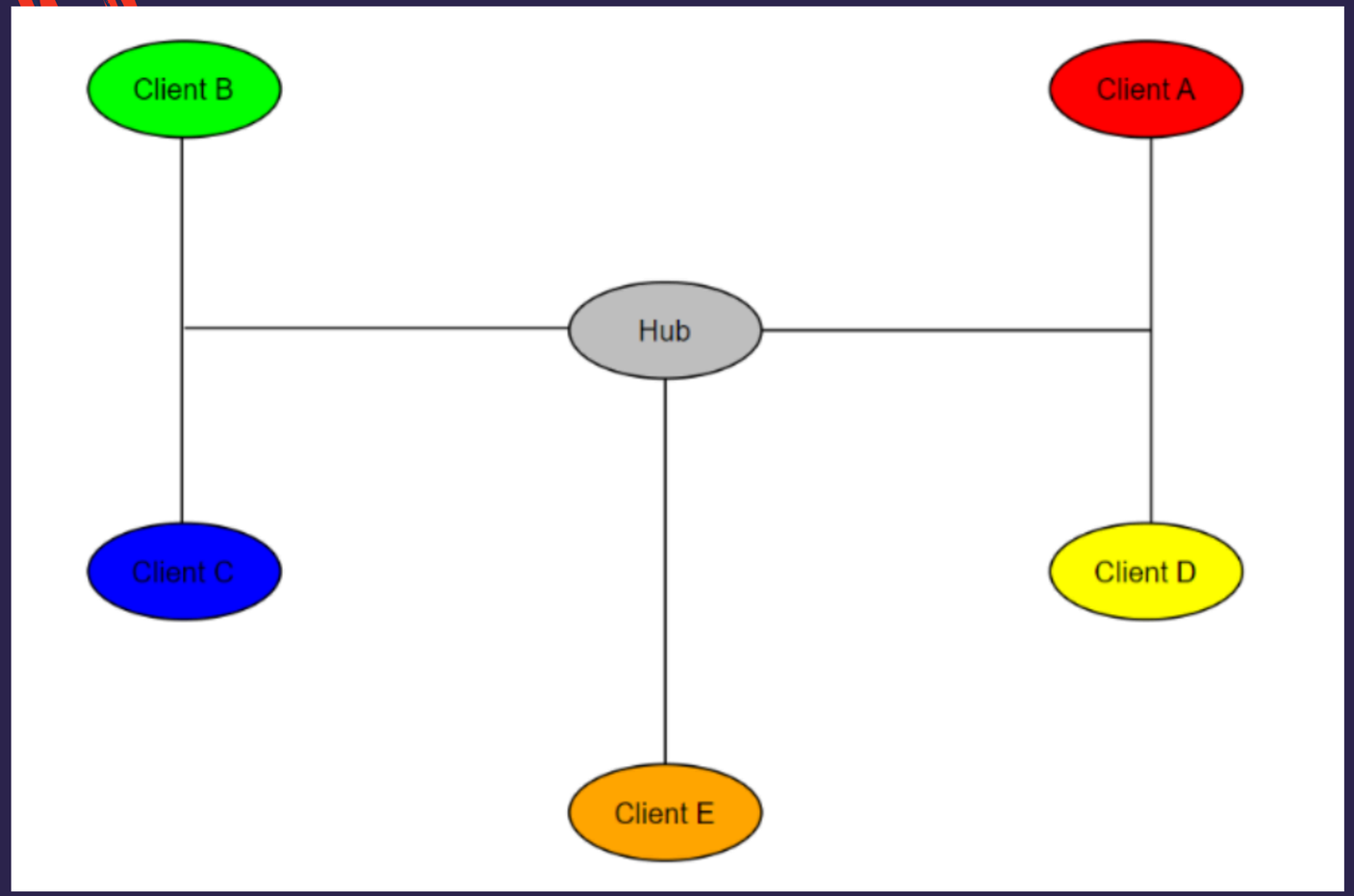
Geographical Setting: the city is divided into several zones, each with varying traffic conditions and distances. the scheduling must account for these factors to ensure timely deliveries.

Type of Disaster: Not applicable in this scenario.

Damage Impact: Delays in deliveries can lead to customer dissatisfaction, loss of future business, and reduced profits.

Importance: Finding an optimal solution for this scenario is crucial because it maximizes the company's revenue, ensures timely deliveries, enhances customer satisfaction, and improves operational efficiency.

Goal and Expected Output: the goal is to develop an algorithm that schedules jobs to maximize total profit while meeting all deadlines. the expected output is a schedule that lists the jobs to be completed within their deadlines and the total profit achieved.



Hub:

- Central point where drones start their delivery routes.
- All delivery jobs originate from the Hub.

Clients:

- Client A: Located to the northeast of the Hub.
- Client B: Located to the northwest of the Hub.
- Client C: Located to the southwest of the Hub.
- Client D: Located to the southeast of the Hub.
- Client E: Located directly south of the Hub.

Routes:

- Straight lines represent the delivery routes from the Hub to each client.
- Connections illustrate the possible paths the drones can take to deliver to each client.

EXAMPLE

Job ID	Client	Deadline (time units)	Profit
A001	Client A	3	100
B002	Client B	2	80
C003	Client C	1	60
D004	Client D	2	40
E005	Client E	1	20

Example Scheduling Using Greedy Algorithm

1.Sort Jobs by profit:

- Jobs sorted by profit: 1,2,3,4,5

2. Schedule Jobs:

- Slot 1: Job 1 (client A, Deadline = 3, Profit = 100)
- Slot 2: Job 2 (Client B, Deadline = 2, Profit = 80)
- Slot 3: Job 4 (CLient D, Deadline = 2, Profit = 40)

3. Total Profit Calculation:

- Total Profit = 100(Job 1) + 80(Job 2) + 40 (Job 4) = 220

Scheduled Jobs	Total Profit	Scheduled Time Slots
['1', '2', '4']	220	['1' at Slot 1, '2' at Slot 2, '4' at Slot 3]

ALGORITHMS COMPARISON

	Strengths	Weakness
Divide and Conquer (DAC)	Efficient for certain types of problems, easy to parallelize.	Not suitable for problems requiring global optimization.
Dynamic Programming (DP)	Provides optimal solutions by considering all possible solutions.	High memory and time complexity.
Greedy Algorithm	Simple to implement, efficient for many problems, provides good approximate solutions.	May not always provide the optimal solution.
Graph Algorithms	Suitable for network flow and connectivity problems.	Complex implementation for job scheduling problems.

CHOSEN ALGORITHM

Chosen

Divide and Conquer (DAC)

Dynamic Programming (DP)

Greedy Algorithm

Graph Algorithm

DEVELOPMENT OF A MODEL

Data Types

Jobs: Represented as an array of Job objects, each containing (jobId, deadline, profit).

For Example:
- jobId (String)
- deadline (int)
- profit (int)

Example:
- Job 1: ('A001', 2, 100)
- Job 2: ('B002', 1, 50)
- Job 3: ('C003', 2, 150)

Objective Function

Maximize Total Profit:
To maximize the total profit by selecting jobs that can be completed within their deadlines.

Constraints

Deadline Constraint: Each job must be completed by its deadline.

Example:
If a job has a deadline of 2, it can only be scheduled in time slots 1 or 2.

Non-Overlapping Constraint:
No two jobs can be scheduled at the same time slot.

- Ensures that only one job is assigned per time slot.

Other Requirements

Space Constraint:
- Use space proportional to the number of jobs and the maximum deadline O(n + d).
- Includes arrays to store jobs and scheduled jobs.

Time Constraint:
- Sorting: O(n log n) for sorting jobs by profit.
- Scheduling: O(n * d) for scheduling jobs up to their deadlines.

Value Constraint:
- Profits and deadlines must be positive integers.

4.0

DESIGNING AN ALGORITHM

DESIGNING AN ALGORITHM

KEY COMPONENTS

1. Sorting Jobs by Profit :

- Purpose: To prioritize jobs with the highest profit.
- Process: Jobs are sorted in descending order based on their profit.

2. Iterative Scheduling:

- Purpose: To schedule jobs in the available time slots up to their deadlines
- Process: Iterate through the sorted jobs.
  - For each job, find the latest available time slot before its deadline.
  - If an available slot is found, schedule the job and update the total profit.

DESIGNING AN ALGORITHM

KEY COMPONENTS

1. Sorting Jobs by Profit :

- Purpose: To prioritize jobs with the highest profit.
- Process: Jobs are sorted in descending order based on their profit.

2. Iterative Scheduling:

- Purpose: To schedule jobs in the available time slots up to their deadlines
- Process: Iterate through the sorted jobs.
  - For each job, find the latest available time slot before its deadline.
  - If an available slot is found, schedule the job and update the total profit.

DESIGNING AN ALGORITHM

RECURRENCE AND OPTIMIZATION

1. RECURRENCE:

THERE IS NO EXPLICIT RECURRENCE RELATION IN THIS GREEDY APPROACH.

2. OPTIMIZATION FUNCTION:

THE TOTAL PROFIT, WHICH IS INCREMENTALLY UPDATED AS JOBS ARE SCHEDULED.

```
for (int j = Math.min(maxDeadline - 1, jobs[i].deadline - 1); j >= 0; j--) {
      if (result[j] == null) {
          result[j] = jobs[i];
          totalProfit += jobs[i].profit; (Optimization Part)
          break;
      }
 }
```

DESIGNING AN ALGORITHM

CODE BREAKDOWN

1. SORTING:

java library sort:
Arrays.sort(jobs, (a, b) -> b.profit - a.profit);

Time Complexity: O(n log n)

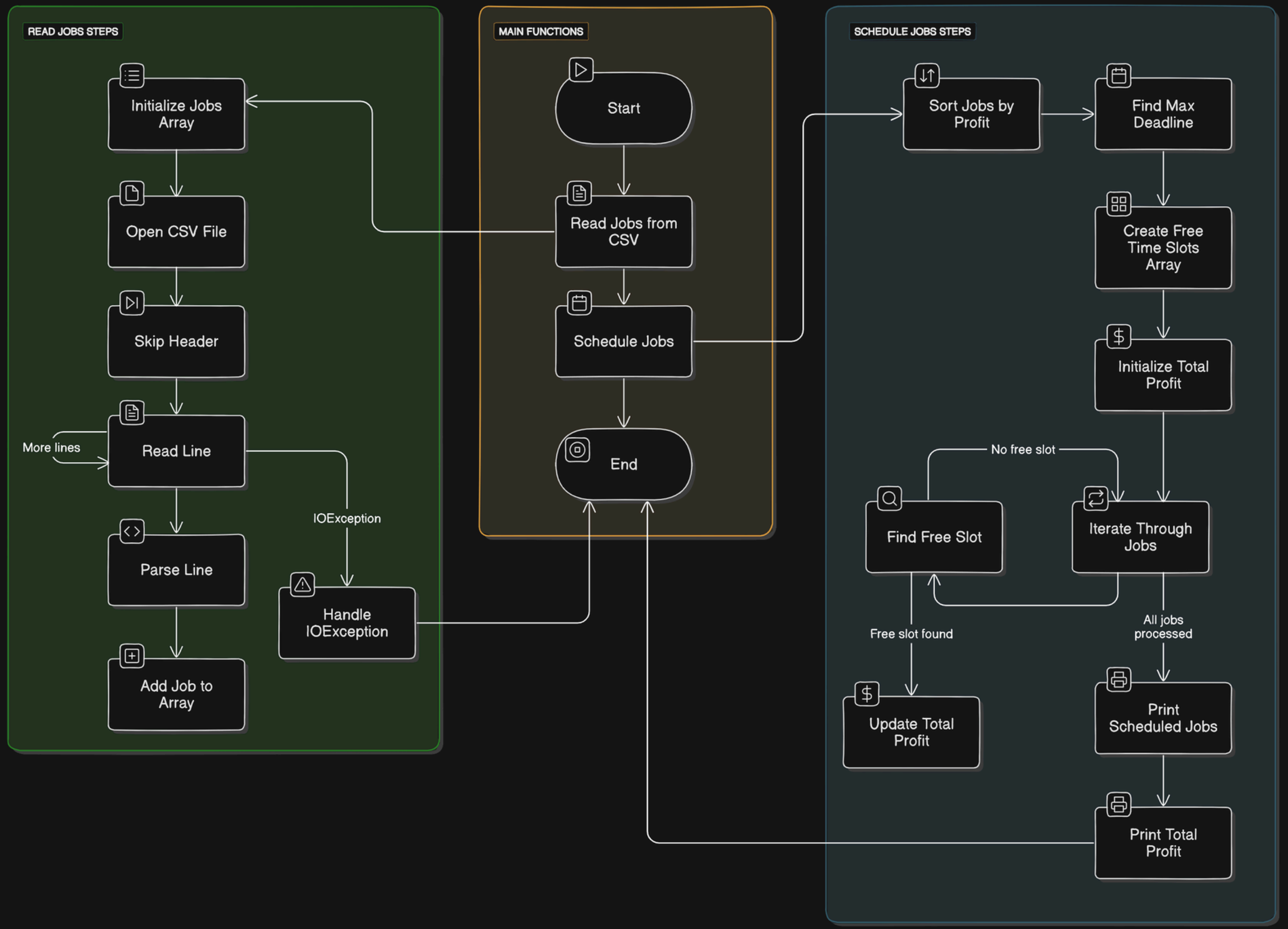
2. OPTIMIZATION FUNCTION:

Iterate through each job and find a free slot from its deadline backward.

Time Complexity: $O(n \times d)$, where d is the maximum deadline.

```
for (int j = Math.min(maxDeadline - 1, jobs[i].deadline - 1)
    if (result[j] == null) {
        result[j] = jobs[i];
        totalProfit += jobs[i].profit;
        break;
    }
}
```

FLOW-CHART


READ JOBS STEPS
Initialize Jobs Array
Open CSV File
Skip Header
More lines
Read Line
IOException
Parse Line
Handle IOException
Add Job to Array
MAIN FUNCTIONS
Start
Read Jobs from CSV
Schedule Jobs
End
SCHEDULE JOBS STEPS
Sort Jobs by Profit
Find Max Deadline
Create Free Time Slots Array
Initialize Total Profit
No free slot
Find Free Slot
Iterate Through Jobs
Free slot found
All jobs processed
Update Total Profit
Print Scheduled Jobs
Print Total Profit

5.0

CHECKING THE CORRECTNESS OF AN ALGORITHM

5.1 GREEDY CHOICE PROPERTY

- THE GREEDY CHOICE PROPERTY ENSURES THAT MAKING A LOCALLY OPTIMAL CHOICE (CHOOSING THE HIGHEST PROFIT JOB THAT FITS WITHIN ITS DEADLINE) LEADS TO A GLOBALLY OPTIMAL SOLUTION.

5.2 OPTIMAL SUBSTRUCTURE

- AN OPTIMAL SOLUTION TO THE PROBLEM CONTAINS OPTIMAL SOLUTIONS TO ITS SUBPROBLEMS.
- FOR THE JOB SCHEDULING PROBLEM, THIS IMPLIES THAT
    - IF WE SCHEDULE THE MOST PROFITABLE JOBS FIRST, THE REMAINING JOBS WILL STILL FORM AN OPTIMAL SCHEDULE FOR THE REMAINING SLOTS.

ANALYSIS OF THE CODE

No.	Steps	Explanation	Complexity	Description
1	Reading Job From CSV	This part of the code reads jobs from a CSV file and stores them in an array.	O(n)	where n is the number of jobs (assuming the file reading andparsing are linear operations).
2	Sorting Jobs by Profit	The jobs are sorted in descending order of profit.	O(n log n)	as the sorting operation (using Arrays.sort)
3	Finding the maximum deadline	The maximum deadline among all jobs is determined by iterating through the jobs array.	O(n)	-
4	Scheduling Jobs	• An array is used to keep track of free time slots. • The algorithm iterates through the sorted jobs and tries to schedule each job in the latest possible slot before its deadline. • For each job, it checks up to the maximum deadline slots to find a free slot.	O(n * d)	where d is the maximum deadline.
5	Printing Results	The scheduled jobs and total profit are printed.	O(d)	since it prints based on the number of time slots (maximum deadline).

COMBINED TIME COMPLEXITY

- The total time complexity is the sum of these complexities:
  - $T(n) = O(n) + O(n \log n) + O(n) + O(n * d) + O(d)$

- Since $O(n \log n)$ and $O(n * d)$ are the dominant terms, the final time complexity is:
  - $T(n) = O(n \log n) + O(n * d)$

RECURRENCE RELATION

- The recurrence relation describes how the time complexity grows with the input size. The code doesn't have a recursive structure, so instead, we derive the relation based on the iterative steps:

- 1. Sorting step:
  - T sort (n) = O(n log n)
- 2. Scheduling step:
  - T schedule(n, d) = O(n * d)

- Combining these, we get the overall time complexity:
  - T(n,d) = O(n log n) + O(n * d)

SUMMARY:

- Asymptotic Time Complexity: O(n log n) + O(n * d)
- Recurrence Relation: T(n,d) = O(n log n) + O(n * d)

6.0

WORST, BEST, AVERAGE ANALYSIS

TABLE COMPARING THE WORST-CASE, BEST-CASE, AND AVERAGE-CASE

Analysis Type	Sorting Jobs by Profit	Finding the Maximum Deadline	Scheduling Jobs	Overall Time Complexity	Where it Happens
Worst-Case	O(nlogn)	O(n)	$O(n \cdot d)$	$O(n\log n+n \cdot d)$	When each job must be checked against all deadlines
Best-Case	O(nlogn)	O(n)	$O(n \cdot d)$	$O(n\log n+n \cdot d)$	When each job is placed in its slot without conflicts
Average-Case	O(nlogn)	O(n)	$O(n \cdot d)$	$O(n\log n+n \cdot d)$	Generally half the maximum deadline slots checked

REASON

The reason the time complexity $O(n \log n + n \cdot d)$ remains the same in the worst-case, best-case, and average-case scenarios is because of the nature of the operations involved

- Consistency Across Cases:
  - Sorting always takes $O(n \log n)$
  - Finding the maximum deadline is $O(n)$
  - Scheduling jobs, in the worst case, needs to check up to d slots for each job, resulting in $O(n \cdot d)$

IMPLEMENTATION OF ALGORITHM

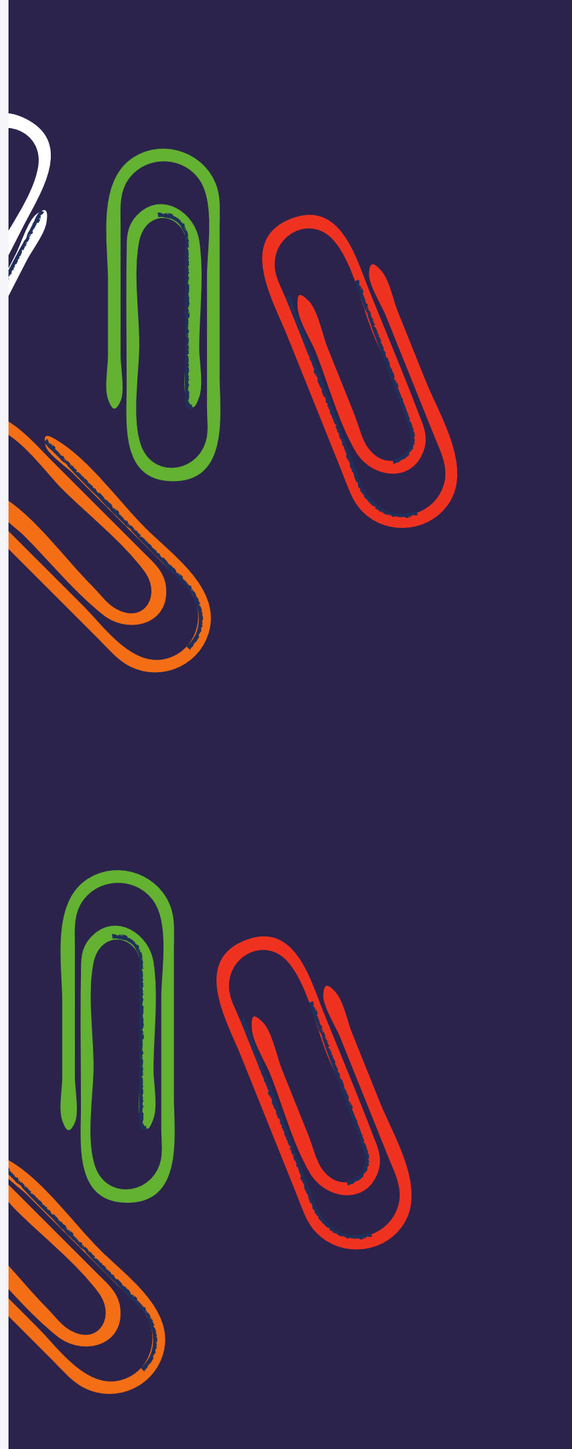
// Job.java

```
public class Job {
  String jobId;
  int deadline;
  int profit;

public Job(String jobId, int deadline, int profit) {
    this.jobId = jobId;
    this.deadline = deadline;
    this.profit = profit;
}
}
```

// JobScheduler.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;

public class JobScheduler {

  //Function to schedule jobs to maximize total profit
  public static void scheduleJobs(Job[] jobs) {
      // Sort jobs by profit in descending order
      Arrays.sort(jobs, (a, b) -> b.profit - a.profit);

      int n = jobs.length;

      // Find the maximum deadline
      int maxDeadline = 0;
      for (Job job : jobs) {
          if (job.deadline > maxDeadline) {
              maxDeadline = job.deadline;
          }
      }
```

```
// Create an array to keep track of free time slots
Job[] result = new Job[maxDeadline];
int[] timeSlots = new int[maxDeadline];

// Keep track of total profit
int totalProfit = 0;

// Iterate through all given jobs
for (int i = 0; i < n; i++) {
    // Find a free time slot for this job (start from the last possible slot)
    for (int j = Math.min(maxDeadline - 1, jobs[i].deadline - 1); j >= 0; j--) {
        // Free slot found
        if (result[j] == null) {
            result[j] = jobs[i];
            timeSlots[j] = j + 1; // record the slot (1-based index)
            totalProfit += jobs[i].profit;
            break;
        }
    }
}

// Print the scheduled jobs
System.out.println("Scheduled Jobs:");
for (int i = 0; i < maxDeadline; i++) {
    if (result[i] != null) {
        System.out.println("Job ID: " + result[i].jobId + ", Profit: " + result[i].profit + ", Sc
    }
}
```



```
        // Print the total profit
        System.out.println("Total Profit: " + totalProfit);
    }

    public static Job[] readJobsFromCSV(String filePath) {
        Job[] jobs = new Job[50];
        String line;
        int index = 0;

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            br.readLine(); // Skip header
            while ((line = br.readLine()) != null) {
                String[] values = line.split(",");
                jobs[index++] = new Job(values[0], Integer.parseInt(values[1]), Integer.parseInt(values[2
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return Arrays.copyOf(jobs, index); // Adjust size of array to actual number of jobs read
    }

    public static void main(String[] args) {
        String filePath = "data\\jobs_dataset.csv"; // Update with the actual path to the CSV file
        Job[] jobs = readJobsFromCSV(filePath);
        scheduleJobs(jobs);
    }
}
```

sample output

```
Scheduled Jobs:
Job ID: E056, Profit: 50, Scheduled Time Slot: 1
Job ID: R090, Profit: 55, Scheduled Time Slot: 2
Job ID: X056, Profit: 55, Scheduled Time Slot: 3
Job ID: O060, Profit: 60, Scheduled Time Slot: 4
Job ID: Y078, Profit: 65, Scheduled Time Slot: 5
Job ID: X600, Profit: 60, Scheduled Time Slot: 6
Job ID: T056, Profit: 60, Scheduled Time Slot: 7
Job ID: B200, Profit: 70, Scheduled Time Slot: 8
Total Profit: 475
```

explaination

- Job ID: E056, Profit: 50, Scheduled Time Slot: 1
  - this job with a profit of 50 is scheduled at the earliest available slot, which is slot 1.
- Job ID: R090, Profit: 55, Scheduled Time Slot: 2
  - this job with a profit of 55 is scheduled in slot 2.
- Job ID: X056, Profit: 55, Scheduled Time Slot: 3
  - this job with a profit of 55 is scheduled in slot 3.
- Job ID: O060, Profit: 60, Scheduled Time Slot: 4
  - this job with a profit of 60 is scheduled in slot 4.
- Job ID: Y078, Profit: 65, Scheduled Time Slot: 5
  - this job with a profit of 65 is scheduled in slot 5.
- Job ID: X600, Profit: 60, Scheduled Time Slot: 6
  - this job with a profit of 60 is scheduled in slot 6.
- Job ID: T056, Profit: 60, Scheduled Time Slot: 7
  - this job with a profit of 60 is scheduled in slot 7.
- Job ID: B200, Profit: 70, Scheduled Time Slot: 8
  - this job with a profit of 70 is scheduled in slot 8.

- Total Profit: 475
- the sum of the profits of the scheduled jobs: (50 + 55 + 55 + 60 + 65 + 60 + 60 + 70 = 475).

SUMMARY

the output shows that the algorithm successfully schedules jobs to maximize the total profit while adhering to their respective deadlines. the total profit of 475 is the sum of the profits of the scheduled jobs, and each job is placed in a time slot that allows it to be completed on time.

DISCUSSION & IMPROVEMENT

- **Non-Overlapping Constraint:** No two jobs can be scheduled at the same time slot. Have been become main issues to this problem scenario.

- **Not Clearly Justify how much drone used inside the company.** If there are several drones (eg. 5 drones), then the problem job scheduling will be a knapsack problem.