K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: E-2	**Roll No.: 16010123325**
Experiment No.__2_____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Study, Implementation, and Analysis of Job Sequencing with Deadlines.

Objective: To learn the Greedy strategy of solving the problems for different types of problems

---

CO to be achieved:

CO 2	Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://lcm.csa.iisc.ernet.in/dsa/node184.htm**
4. **http://students.ceid.upatras.gr/~papagel/project/kruskal.htm**
5. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.html**
6. **http://lcm.csa.iisc.ernet.in/dsa/node183.html**
7. **http://students.ceid.upatras.gr/~papagel/project/prim.htm**
8. **http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:The Job Sequencing with Deadlines problem is a classic optimization problem where the goal is to schedule jobs to maximize profit, ensuring each job is completed before its deadline. The algorithm follows a greedy approach.The Job Sequencing with Deadlines Problem is a foundational optimization problem in scheduling theory and computer science. Its historical development is closely tied to advancements in algorithm design, combinatorial optimization, and practical applications in operations research and industrial processes.

Origins and Early Development

Scheduling Problems in Industry (Mid-20th Century):The problem emerged during the industrial revolution and later in the mid-20th century with the rise of operational research.Scheduling tasks to maximize efficiency, minimize delays, or maximize profits became critical for industries like manufacturing and logistics.

Development of Mathematical Formulation:Researchers began formalizing scheduling problems, including constraints like deadlines and profit maximization, laying the groundwork for job sequencing problems.The focus shifted from heuristic or ad-hoc methods to formal algorithmic solutions.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Greedy method of problem solving Vs other methods of problem solving, optimality of the solution, knapsack problem and their applications

---

Algorithm:

Input: A set of n jobs, where each job i has:

1. pi: Profit if the job is completed.
2. di: Deadline (maximum time slot by which it should be completed).

Output: A schedule of jobs such that:

1. Each job is completed within its deadline.
2. The profit is maximized.

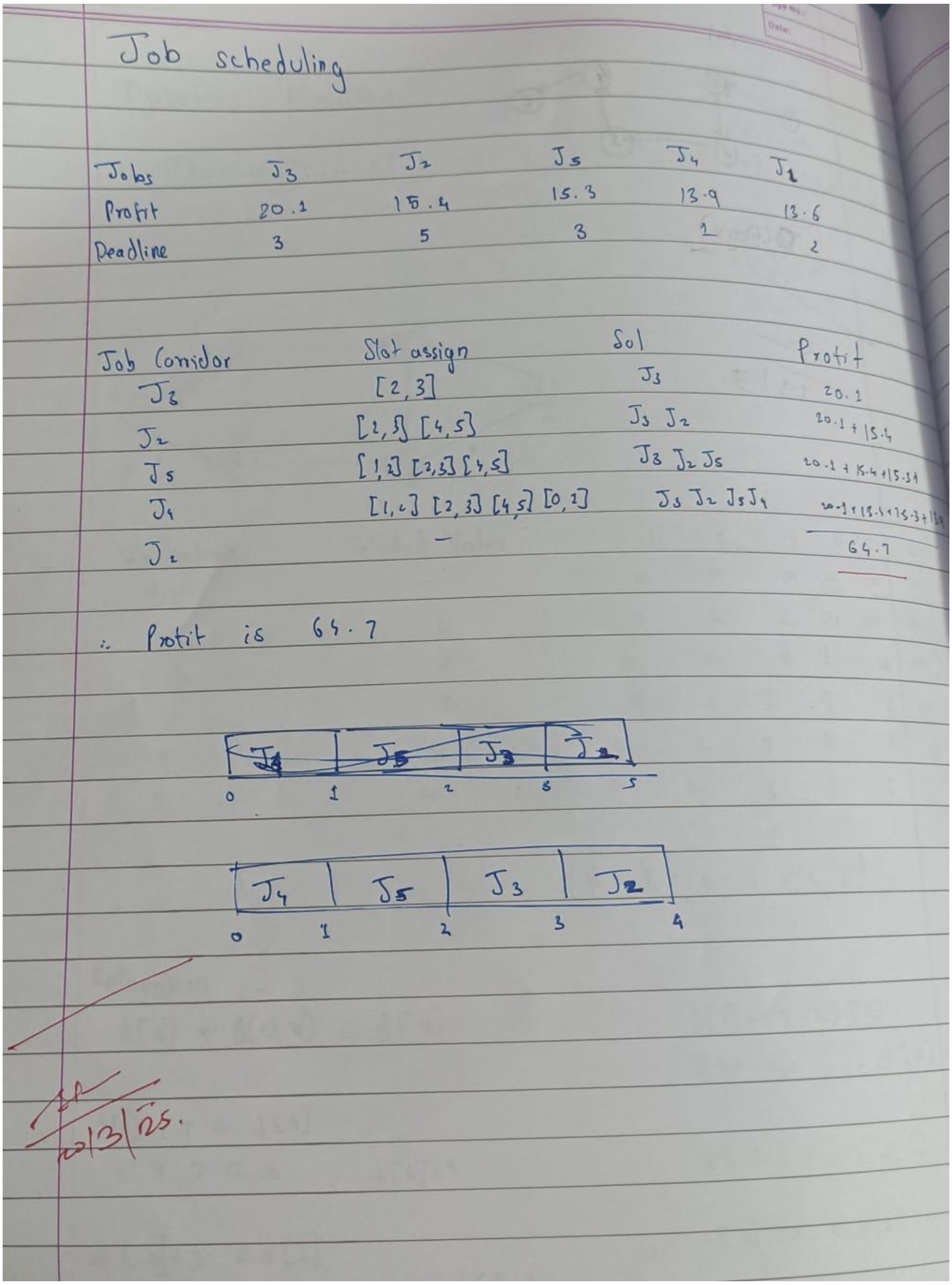
Step 1: Sort Jobs by Profit. Sort all jobs in descending order of their profit pi. Jobs with higher profits are prioritized.

Step 2: Allocate Time Slots

1. Iterate through the sorted jobs.
2. For each job: Find the latest available time slot t≤di (the deadline of the job).If a time slot is available, schedule the job in that slot and update the slot as occupied.

Step 3: Output the Job Schedule.(Return the scheduled jobs and the total profit.)

Example:

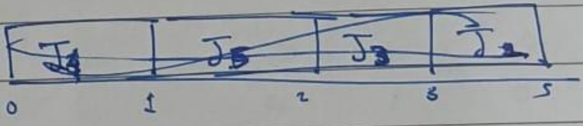


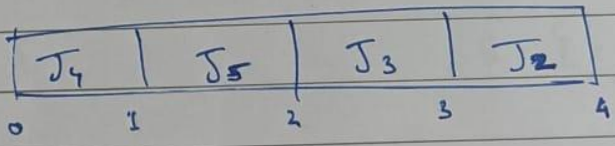
Job scheduling

Jobs	J_3	J_2	J_5	J_4	J_1
Profit	20.1	15.4	15.3	13.9	13.6
Deadline	3	5	3	1	2

Job Considor	Slot assign	Sol	Profit
J_3	[2, 3]	J_3	20.1
J_2	[2, 3] [4, 5]	J_3 J_2	20.1 + 15.4
J_5	[1, 2] [2, 3] [4, 5]	J_3 J_2 J_5	20.1 + 15.4 + 15.3
J_4	[1, 2] [2, 3] [4, 5] [0, 1]	J_3 J_2 J_5 J_4	20.1 + 15.4 + 15.3 + 13.9
J_1	—		64.7

∴ Profit is 64.7





20/3/25.

Analysis of algorithm:

The algorithm consists of the following steps:

1. **Sorting Jobs by Profit**:
    - Sorting takes **O(n log n)** using **Arrays.sort()** (which uses TimSort).
2. **Placing Jobs in Available Slots**:
    - For each job, we find the latest available slot **before its deadline**.
    - In the worst case, we may check-up to **d** slots (where d is the maximum deadline).
    - This takes **O(n * d)** in the worst case.

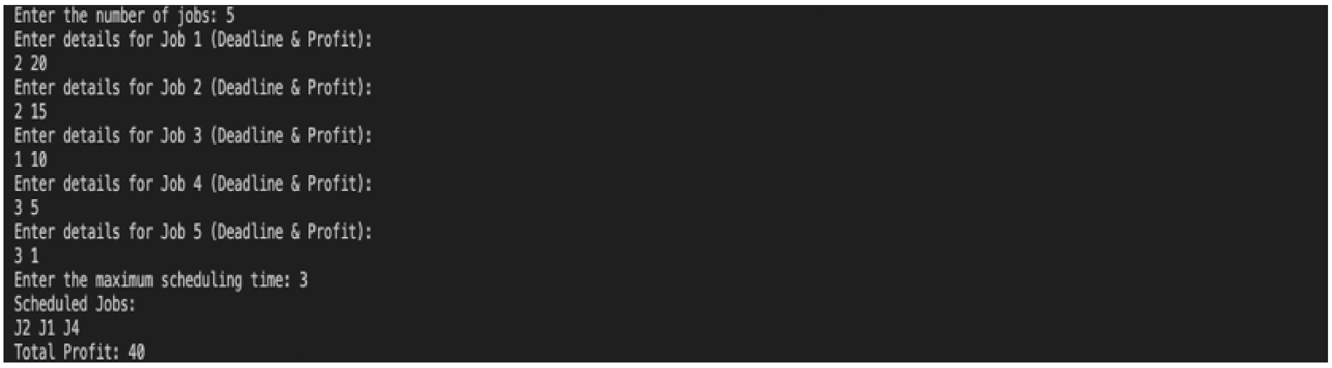
Overall Complexity

- If d (maximum deadline) is small:
    - Complexity ≈ **O(n log n) + O(n d) ≈ O(n log n)** (since sorting dominates).

Implementation of algorithm:

```java
import java.util.Arrays;
import java.util.Scanner;

class Job {
    int id, deadline, profit;

    public Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}

class JobScheduling {

    static void scheduleJobs(Job[] jobs, int maxTime) {

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

        int[] jobSequence = new int[maxTime + 1];
        Arrays.fill(jobSequence, -1);

        int totalProfit = 0, jobsScheduled = 0;
        for (Job job : jobs) {

            for (int j = Math.min(maxTime, job.deadline); j > 0; j--) {
                if (jobSequence[j] == -1) { // Slot is free
                    jobSequence[j] = job.id;
                    totalProfit += job.profit;
                    jobsScheduled++;
                    break;
                }
            }
        }

        System.out.println("Scheduled Jobs:");
        for (int i = 1; i <= maxTime; i++) {
            if (jobSequence[i] != -1) {
                System.out.print("J" + jobSequence[i] + " ");
            }
        }
        System.out.println("\nTotal Profit: " + totalProfit);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of jobs: ");
        int n = sc.nextInt();

        Job[] jobs = new Job[n];


        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for Job " + (i + 1) + " (Deadline &
Profit): ");
            int deadline = sc.nextInt();
            int profit = sc.nextInt();
            jobs[i] = new Job(i + 1, deadline, profit);
        }


        System.out.print("Enter the maximum scheduling time: ");
        int maxTime = sc.nextInt();


        scheduleJobs(jobs, maxTime);

        sc.close();
    }
}
```

Output:



```
Enter the number of jobs: 5
Enter details for Job 1 (Deadline & Profit):
2 20
Enter details for Job 2 (Deadline & Profit):
2 15
Enter details for Job 3 (Deadline & Profit):
1 10
Enter details for Job 4 (Deadline & Profit):
3 5
Enter details for Job 5 (Deadline & Profit):
3 1
Enter the maximum scheduling time: 3
Scheduled Jobs:
J2 J1 J4
Total Profit: 40
```

Conclusion:
The above experiment highlights implementation of job scheduling algorithm as part of the Greedy approach, and analysis of the implemented algorithm with an example.