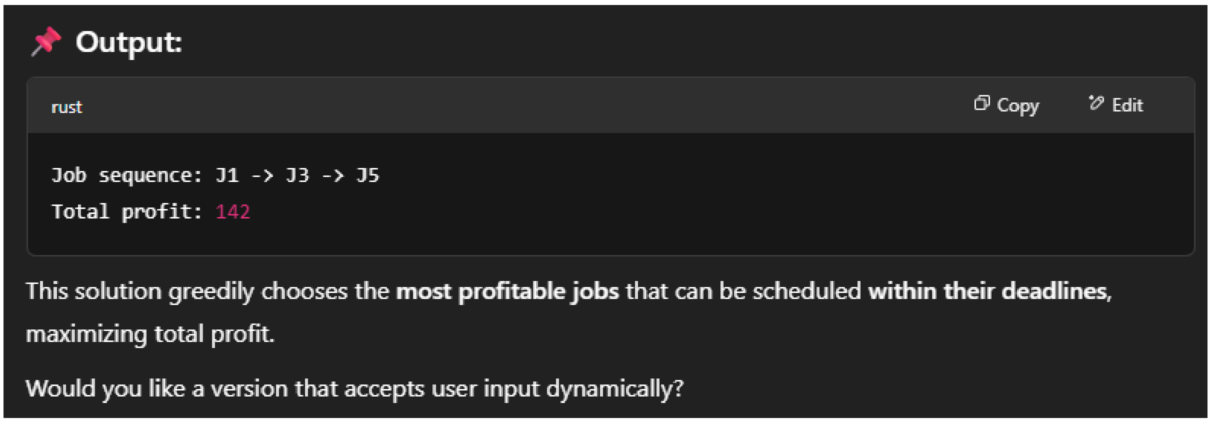
GREEDY: JOB

```
# Job class to store job id, deadline and profit
class Job:
    def __init__(self, job_id, deadline, profit):
        self.id = job_id
        self.deadline = deadline
        self.profit = profit


def job_scheduling(jobs):
    # Sort jobs by profit in descending order
    jobs.sort(key=lambda x: x.profit, reverse=True)

    # Find maximum deadline to create time slots
    max_deadline = max(job.deadline for job in jobs)
    slots = [False] * max_deadline
    job_sequence = [''] * max_deadline
    total_profit = 0

    for job in jobs:
        # Try to place job in the latest available slot before deadline
        for i in range(min(max_deadline, job.deadline) - 1, -1, -1):
            if not slots[i]:
                slots[i] = True
                job_sequence[i] = job.id
                total_profit += job.profit
                break
    print("Job sequence:", ' -> '.join(job_sequence))

    print("Total profit:", total_profit)


# Example usage

jobs = [

    Job('J1', 2, 100),

    Job('J2', 1, 19),

    Job('J3', 2, 27),

    Job('J4', 1, 25),

    Job('J5', 3, 15)

]


job_scheduling(jobs)
```



Output:

rust

```
Job sequence: J1 -> J3 -> J5
Total profit: 142
```

This solution greedily chooses the **most profitable jobs** that can be scheduled **within their deadlines**, maximizing total profit.

Would you like a version that accepts user input dynamically?

Possible Viva Questions on the Practical:

1. **What is the Job Scheduling Problem?**
    - The Job Scheduling Problem involves scheduling jobs in such a way that the total profit is maximized, and each job must be completed before its deadline.
2. **How does the Greedy Algorithm work for Job Scheduling?**
    - The Greedy Algorithm sorts the jobs based on their profit in descending order, then assigns each job to the latest available slot before its deadline, ensuring no job misses its deadline.

3. **Why is the algorithm greedy?**
    - The algorithm is greedy because it makes the locally optimal choice at each step (selecting the job with the highest profit), hoping that this will lead to a globally optimal solution.
4. **What data structure is used to store the scheduled jobs?**
    - The job_sequence array is used to store the job ids that are scheduled in available time slots.
5. **How do you handle jobs with the same deadline?**
    - The algorithm places the highest profit job first. If multiple jobs have the same deadline, the one with the highest profit is scheduled first.
6. **What is the time complexity of this algorithm?**
    - The time complexity is **O(n log n)** due to the sorting of jobs, where n is the number of jobs.
7. **What happens if two jobs have the same profit?**
    - If two jobs have the same profit, the one that comes first in the sorted list (i.e., the one listed earlier in the input) will be scheduled first.
8. **How do you ensure that jobs are scheduled before their deadlines?**
    - By iterating through available slots starting from the latest possible slot before the job's deadline, ensuring that each job is placed within the permissible time window.
9. **What are the benefits of using a greedy approach in this problem?**
    - The greedy approach is simple and effective for problems like job scheduling where making a local optimal choice leads to a global optimal solution.
10. **Can this algorithm handle unsorted input?**
    - Yes, the algorithm first sorts the jobs by profit, so the input does not need to be sorted beforehand.
11. **What would happen if we used a different criterion for sorting jobs?**
    - If we sorted by deadline or some other criterion, the resulting job sequence and total profit might not be optimal.
12. **What is the significance of checking from the latest available slot?**
    - Checking from the latest available slot maximizes the number of available slots for future jobs, which is crucial for maximizing the total profit.