NAME: Atharva Kangralkar

ROLL NO: 54

CLASS: CS(AIML) - A

PRN NO: 12311493

BATCH: 2

LAB ASSIGNMENT 5

Aim: Job sequencing Problem

Step 1: Pseudo code

```
// Define the structure for a job
STRUCTURE Job
   id: INTEGER
   deadline: INTEGER
   profit: INTEGER
END STRUCTURE

// Main function to find and print the optimal job schedule
FUNCTION findOptimalJobSchedule(allJobs: LIST of Job)

   // 1. Sort all jobs in descending order of their profit.
   SORT allJobs in descending order based on profit.

   // 2. Find the maximum deadline among all jobs.
   maxDeadline = 0
   FOR EACH job IN allJobs
      IF job.deadline > maxDeadline THEN
         maxDeadline = job.deadline
      END IF
```

```
    END FOR

    // 3. Initialize the time slots for the schedule.
    // The size is maxDeadline + 1 to include day 0.
    // Initialize all slots to an empty state (e.g., -1).
    timeSlots = ARRAY of size (maxDeadline + 1), initialized with -1.

    totalProfit = 0
    scheduledJobsCount = 0

    // 4. Iterate through the sorted jobs to schedule them.
    FOR EACH currentJob IN allJobs
        // Find a free time slot for the current job by checking from its deadline backwards to day 0.
        FOR day FROM currentJob.deadline DOWN TO 0
            IF timeSlots[day] is empty (-1) THEN
                // Assign the job to this empty slot.
                timeSlots[day] = currentJob.id

                // Update total profit and the count of scheduled jobs.
                totalProfit = totalProfit + currentJob.profit
                scheduledJobsCount = scheduledJobsCount + 1

                // Once scheduled, break the inner loop and move to the next job.
                BREAK
            END IF
        END FOR
    END FOR

    // 5. Display the results.
```

```
    PRINT "Maximum possible profit: " + totalProfit
    PRINT "Number of jobs scheduled: " + scheduledJobsCount
    PRINT "Scheduled Job Sequence (Job ID in Time Slot):"

    FOR day FROM 0 TO maxDeadline
        IF timeSlots[day] is NOT empty THEN
            PRINT "Job " + timeSlots[day] + " on Day " + day
        END IF
    END FOR

END FUNCTION

// Main program execution block
PROCEDURE main
    // Get the number of jobs from the user.
    PRINT "Enter the number of jobs:"
    READ numberOfJobs

    // Create an empty list to store the jobs.
    CREATE a list named `jobs`

    // Get the details for each job from the user.
    PRINT "Enter details for each job (0-indexed Deadline and Profit):"
    FOR i FROM 1 TO numberOfJobs
        PRINT "Job " + i + ": "
        READ inputDeadline, inputProfit

        // Create a new job object and add it to the list.
        CREATE a new Job with id=i, deadline=inputDeadline, profit=inputProfit
```

```
        ADD the new Job to the `jobs` list
    END FOR

    // Call the function to find and display the optimal schedule.
    CALL findOptimalJobSchedule(jobs)

END PROCEDURE
```

Step2: Code

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Job {
    int id;
    int deadline;
    int profit;
};

bool compareJobsByProfit(const Job& firstJob, const Job& secondJob) {
    return firstJob.profit > secondJob.profit;
}

// The main function that finds the optimal job schedule.
void findOptimalJobSchedule(vector<Job>& allJobs) {
    // 1. Sort all jobs in descending order of their profit.
    sort(allJobs.begin(), allJobs.end(), compareJobsByProfit);
```

```
// 2. Find the maximum deadline to determine the number of time slots.
int maxDeadline = 0;
for (const auto& job : allJobs) {
    if (job.deadline > maxDeadline) {
        maxDeadline = job.deadline;
    }
}

// 3. Create a schedule for time slots from 0 to maxDeadline.
// The size is maxDeadline + 1 to accommodate the 0th slot.
vector<int> timeSlots(maxDeadline + 1, -1); // -1 indicates an empty slot

int scheduledJobsCount = 0;
int totalProfit = 0;

// 4. Iterate through the sorted jobs.
for (const auto& currentJob : allJobs) {
    // Find a free slot, checking backwards from the job's deadline down to day 0.
    for (int day = currentJob.deadline; day >= 0; --day) {
        // If an empty slot is found.
        if (timeSlots[day] == -1) {
            // Assign this job to the free slot.
            timeSlots[day] = currentJob.id;

            // Update profit and job count.
            totalProfit += currentJob.profit;
            scheduledJobsCount++;
```

```
            break;
        }
    }
}

cout << "\n Job Scheduling Results (0-indexed) are:-\n";
cout << "Maximum possible profit: " << totalProfit << "\n";
cout << "Number of jobs scheduled: " << scheduledJobsCount << "\n";
cout << "Scheduled Job Sequence (Job ID in Time Slot):\n";

// The output loop now also starts from day 0.
for (int day = 0; day <= maxDeadline; ++day) {
    if (timeSlots[day] != -1) {
        cout << "  Job " << timeSlots[day] << " on Day " << day << "\n";
    }
}
}

int main() {
    int numberOfJobs;
    cout << "Enter the number of jobs: ";
    cin >> numberOfJobs;

    vector<Job> jobs(numberOfJobs);
    cout << "Enter details for each job (0-indexed Deadline and Profit):\n";
    for (int i = 0; i < numberOfJobs; ++i) {
        jobs[i].id = i + 1;
        cout << "Job " << jobs[i].id << ": ";
        cin >> jobs[i].deadline >> jobs[i].profit;
```

```
    }

    findOptimalJobSchedule(jobs);

    return 0;
}
```

Step3: Output

```
PS C:\TY CS(AIML)\DAA\Lab Codes> cd "c:\TY CS(AIML)\DAA\Lab Codes\" ; if ($?) { g++ job_sequencing.cpp -o job_sequencing } ; if ($?) { .\job_sequencing
}
Enter the number of jobs: 8
Enter details for each job (0-indexed Deadline and Profit):
Job 1: 4 20
Job 2: 5 60
Job 3: 6 70
Job 4: 6 65
Job 5: 4 25
Job 6: 2 80
Job 7: 2 10
Job 8: 2 22
```

```
 Job Scheduling Results (0-indexed) are:-
Maximum possible profit: 342
Number of jobs scheduled: 7
Scheduled Job Sequence (Job ID in Time Slot):
  Job 1 on Day 0
  Job 8 on Day 1
  Job 6 on Day 2
  Job 5 on Day 3
  Job 2 on Day 4
  Job 4 on Day 5
  Job 3 on Day 6
```

Step4: Time complexity Analysis

$T(n,m)=Tsort(n)+Tschedule(n,m)$

$T(n,m)=O(nlogn)+O(n\times m)=O(nlogn+n\times m)$