**Name: Sai Gulve**

**PRN No: 202201040145**

**Batch: A4**

**Roll No: 63**

**Problem Statement:**

Given an array of jobs where each job has a **deadline** and an associated **profit**, the objective is to schedule jobs such that the **total profit is maximized**. Each job takes exactly **one unit of time** to complete, and only **one job can be scheduled at a time**. Implement a **Greedy Algorithm** to solve this problem and analyze its execution time.

**Introduction:**

In real-world scenarios like **task scheduling, manufacturing processes, and job allocations**, we often encounter situations where limited resources (such as time) must be allocated to maximize **profit** or **efficiency**. The **Job Scheduling Problem** is a classical example of such optimization problems.

A **Greedy Algorithm** is an **optimal approach** for solving this problem since it ensures that **local choices (choosing the job with the highest profit first)** lead to a **globally optimal solution**.

**Approach & Algorithm:**

The algorithm follows these steps:

1. **Sort the jobs** in **descending order of profit** (Highest profit job is prioritized).
2. **Find the maximum deadline** among the given jobs.
3. **Create a scheduling array** to track available time slots.
4. **Assign each job** to the **latest available time slot before its deadline**.
5. **Calculate and return the maximum profit** achievable.
6. **Measure the execution time** of the algorithm using the chrono library.

**Time Complexity Analysis:**

1. **Sorting Jobs** (Bubble Sort) → **O(n²)**
2. **Finding Maximum Deadline** → **O(n)**
3. **Job Scheduling** → **O(n²)** (Nested loop to assign jobs)
4. **Overall Complexity** → **O(n²)**

**Code :-**

#include <iostream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Function to sort jobs in descending order of profit (Manual Bubble Sort)

void sortJobs(int profits[], int deadlines[], int n) {

for (int i = 0; i < n - 1; ++i) {

for (int j = 0; j < n - i - 1; ++j) {

if (profits[j] < profits[j + 1]) {

// Swap profits

int tempProfit = profits[j];

profits[j] = profits[j + 1];

profits[j + 1] = tempProfit;

// Swap corresponding deadlines

int tempDeadline = deadlines[j];

deadlines[j] = deadlines[j + 1];

deadlines[j + 1] = tempDeadline;

}

}

}

}

// Function to find the maximum deadline

int findMaxDeadline(int deadlines[], int n) {

int maxDeadline = 0;

for (int i = 0; i < n; ++i) {

if (deadlines[i] > maxDeadline) {

maxDeadline = deadlines[i];

}

}

return maxDeadline;

}

// Function to schedule jobs to maximize profit

int jobScheduling(int profits[], int deadlines[], int n) {

// Step 1: Sort jobs in descending order of profit

sortJobs(profits, deadlines, n);

// Step 2: Find the maximum deadline

int maxDeadline = findMaxDeadline(deadlines, n);

// Step 3: Create an array to keep track of free slots

int schedule[maxDeadline + 1]; // 1-based indexing

for (int i = 0; i <= maxDeadline; ++i) {

schedule[i] = -1; // -1 means slot is empty

}

// Step 4: Assign jobs to slots

int totalProfit = 0;

int jobsCount = 0;

for (int i = 0; i < n; ++i) {

// Try to schedule job at the last possible slot

for (int j = deadlines[i]; j > 0; --j) {

if (schedule[j] == -1) {

schedule[j] = i;

totalProfit += profits[i];

jobsCount++;

break;

}

}

}

// Display scheduled jobs

cout << "Scheduled Jobs: ";

for (int i = 1; i <= maxDeadline; ++i) {

if (schedule[i] != -1) {

cout << "(" << profits[schedule[i]] << ", " << deadlines[schedule[i]] << ") ";

}

}

cout << endl;

return totalProfit;

}

int main() {

int n;

cout << "Enter the number of jobs: ";

cin >> n;

int profits[n], deadlines[n];

cout << "Enter profits: ";

for (int i = 0; i < n; ++i) {

cin >> profits[i];

}

cout << "Enter deadlines: ";

for (int i = 0; i < n; ++i) {

cin >> deadlines[i];

}

// Start measuring time

auto start\_time = high\_resolution\_clock::now();

int maxProfit = jobScheduling(profits, deadlines, n);

// Stop measuring time

auto end\_time = high\_resolution\_clock::now();

// Calculate duration

auto duration = duration\_cast<microseconds>(end\_time - start\_time);

cout << "Maximum Profit: " << maxProfit << endl;

cout << "Execution Time: " << duration.count() << " microseconds" << endl;

return 0;

}

