

12
.5

Job Sequencing Problem

MEDIUM

What You Will Learn

- ✓ Schedule jobs with deadlines to maximize total profit
- ✓ Greedy: process highest-profit jobs first
- ✓ How to assign jobs to the latest available time slot
- ✓ The slot array trick for efficient scheduling

Intuition First

You're a freelancer with N one-hour jobs. Each job has a deadline (it must finish by that time slot) and a profit. You can do one job per slot. Which jobs should you pick to earn the most?

Greedy insight: always try to schedule the highest-paying job first. For each job, try to put it in the latest available slot before its deadline — this keeps earlier slots free for other jobs.

The Algorithm

- Sort jobs by PROFIT (descending — highest first)
- For each job: find the latest free slot $\leq$ job.deadline
- If a free slot exists: assign job there, add profit
- If no free slot: skip this job
- Track with a boolean slot[] array: slot[t]=true means slot t is taken

Visual — Job Table

Jobs (sorted by profit desc):

Scheduled: $[B(\text{slot}1), A(\text{slot}2)] \rightarrow \text{Total profit} = 19 + 100 = 119 \checkmark$

Dry Run — Jobs sorted by profit: A(100,d=2), C(27,d=2), D(25,d=1), B(19,d=1)

Slots available: $[_, _]$ (indices 1 and 2)

Step 1

Job A profit=100, deadline=2. Try slot 2 $\rightarrow$ FREE. Assign A to slot 2. profit=100

slots = $[_, A]$

Step 2

Job C profit=27, deadline=2. Try slot 2 $\rightarrow$ TAKEN. Try slot 1 $\rightarrow$ FREE. Assign C to slot 1. profit=127

slots = [C, A]

Step 3 Job D profit=25, deadline=1. Try slot 1 → TAKEN. No earlier slots. SKIP.

Step 4 Job B profit=19, deadline=1. Try slot 1 → TAKEN. SKIP.

Final schedule: [C, A] Total profit = 27 + 100 = 127 ✓

■ Java Code

```
public static int[] jobScheduling(char[] jobs, int[] deadlines, int[] profits) {
    int n = jobs.length;
    int maxDeadline = Arrays.stream(deadlines).max().getAsInt();
    // Create and sort by profit (descending)
    int[][] jobArr = new int[n][2]; // [deadline, profit]
    for (int i = 0; i < n; i++) {
        jobArr[i][0] = deadlines[i];
        jobArr[i][1] = profits[i];
    }
    Arrays.sort(jobArr, (a, b) -> b[1] - a[1]); // desc profit
    boolean[] slot = new boolean[maxDeadline + 1]; // slot[i]=true if taken
    int totalProfit = 0, jobCount = 0;
    for (int[] job : jobArr) {
        // Try to assign latest available slot <= deadline
        for (int t = job[0]; t >= 1; t--) {
            if (!slot[t]) {
                slot[t] = true;
                totalProfit += job[1];
                jobCount++;
                break; // Slot found, move to next job
            }
        }
    }
    return new int[]{jobCount, totalProfit};
}
```

■■ Time & Space Complexity

Case	Time	Space	What it really means
All Cases	$O(n \log n + n \cdot d)$	$O(d)$	Sort $O(n \log n)$, slot search $O(d)$ per job. d = max deadline
Optimized	$O(n \log n)$	$O(n)$	Using Disjoint Set Union (DSU) for $O(1)$ slot finding

■ Why Latest Available Slot?

- Assigning to the latest slot $\leq$ deadline keeps EARLIER slots free for other jobs

- If we assigned to slot 1 for a job with deadline 3, slots 2 and 3 are wasted
- Using the latest slot is the space-efficient greedy choice — it's provably optimal

■ Common Mistakes

- Sorting by deadline instead of profit — we want max profit first
- Slot array of size n instead of $\text{maxDeadline}+1$ — deadline can be $> n$
- Inner loop going forward ($t++$) instead of backward ($t--$) — must try latest slot first

■ Pro Tips

- DSU optimization: `find(slot)` returns the latest free slot — $O(1)$ amortized
- This is a classic in OS scheduling and task management interviews
- Extension: fractional jobs (deadlines per unit) → fractional knapsack territory

■ Visual Thinking

Picture a wall calendar with columns = time slots (1, 2, 3...).

Jobs are sticky notes — each has a deadline (latest column it can go in) and a profit value.

You have highest-profit notes in hand. For each note, stick it in the rightmost open column $\leq$ its deadline.

If no open column exists: crumple and toss the note.

Animation: sticky notes fall from above sorted by profit. Each sticks to rightmost open slot, or floats away.

■ Practice Problems

[Med] Jobs: A($d=4, p=20$), B($d=1, p=10$), C($d=1, p=40$), D($d=1, p=30$) — max profit?

[Med] What if two jobs have the same profit — which do you prefer?

[Hard] Implement the DSU-optimized version for $O(n \log n)$ total

[Hard] Job scheduling with variable durations (not unit time)

■ Quick Revision Box

- Sort by profit DESCENDING
- For each job: try latest free slot $\leq$ deadline (scan $t--$ from deadline to 1)
- Mark slot taken, add profit
- $O(n \log n + n*d)$ naive, $O(n \log n)$ with DSU
- Key: latest slot $\rightarrow$ keeps earlier slots free for others