PSO 2 — Week 6: Solutions

CS381, Fall 2026

Problem 1

(a) Both greedies are unboundedly bad

Earliest finish time. Two jobs: $[0, 1]$ with weight 1, and $[0, 2]$ with weight W. The greedy takes $[0, 1]$ because it finishes first, after which $[0, 2]$ is incompatible, and returns 1. The optimum is W, so taking $W \geq c$ gives a ratio of at least c.

Heaviest first. One long job $[0, 2k]$ of weight 2, together with k short jobs $[2i, 2i + 1]$ for $i = 0, \ldots, k - 1$, each of weight 1. The short jobs are pairwise disjoint and each overlaps the long one. The greedy takes the long job first and is then blocked, returning 2; the optimum is k, so taking $k \geq 2c$ gives a ratio of at least c.

Where the unweighted arguments break. For earliest finish time the unweighted proof runs: if $g_1, \ldots, g_p$ are the greedy's jobs in order and $o_1, \ldots, o_q$ an optimal schedule in order, then $f(g_\rho) \leq f(o_\rho)$ for every ρ by induction, so the greedy never runs out of room before an optimum does, and $p \geq q$. Every step of that induction survives the introduction of weights. What fails is the conclusion: $p \geq q$ compares cardinalities, and cardinality is no longer the objective.

For heaviest first the failure is earlier. The exchange one would want — insert the heaviest job into an optimal schedule and remove what it conflicts with — may have to remove many jobs, and nothing bounds their combined weight; above, the heaviest job displaces k of them.

(b) The $O(n \log n)$ algorithm

Convention. The intervals are closed, so jobs i and j conflict exactly when $s_i \leq f_j$ and $s_j \leq f_i$; in particular two jobs that touch at a single coordinate ($f_i = s_j$) do conflict, as in Week 4. This enters in one place, the definition of the predecessor:

$$p(j) \;:=\; \max\{\, i : f_i < s_j \,\}, \qquad p(j) := 0 \text{ if there is no such } i,$$

with a *strict* inequality. Writing $f_i \leq s_j$ there would admit a touching pair.

```
sort the jobs so that f_1 <= f_2 <= ... <= f_n
for j = 1 to n: p(j) := binary search for the last f_i < s_j   // 0 if none
M[0] := 0
for j = 1 to n: M[j] := max(M[j-1], w_j + M[p(j)])
return M[n]
```

Correctness. We show by induction on j that $M[j]$ is the maximum weight of a schedule using only jobs of index at most j. The case $j = 0$ is clear. Let S be an optimal such schedule.

If $j \notin S$ then S lies inside $\{1, \ldots, j - 1\}$ and $w(S) \leq M[j - 1]$.

If $j \in S$, take any other $i \in S$. Then $i < j$, so $f_i \leq f_j$ by the sort order. Jobs i and j are disjoint and f_i is a point of job i, so f_i cannot lie in $[s_j, f_j]$; combined with $f_i \leq f_j$ this forces $f_i < s_j$. Since the jobs are sorted by finish time, $\{\, i : f_i < s_j \,\}$ is a prefix of the order, so $f_i < s_j$ gives $i \leq p(j)$. Hence every job of S other than j has index at most $p(j)$, and $w(S) \leq w_j + M[p(j)]$.

Both quantities are achievable — an optimal schedule inside $\{1, \ldots, j-1\}$ is one candidate, and an optimal schedule inside $\{1, \ldots, p(j)\}$ together with j is another, the latter feasible precisely because every job of index at most $p(j)$ finishes strictly before s_j. So $M[j]$ is the maximum of the two. □

Recovering the schedule. Walk back from $j = n$: if $j = 0$ stop; if $M[j] = M[j-1]$, job j was not needed, so move to $j - 1$; otherwise the maximum came from the other branch, so output j and move to $p(j)$. As with the LCS table in Week 5, the stored values are enough to tell the two branches apart, and the walk is $O(n)$.

Time. $\Theta(n \log n)$ to sort, $O(n \log n)$ for the n binary searches, $O(n)$ for the sweep and the walk.

(c) At most k jobs

Let $M[j][t]$ be the maximum weight of a schedule using at most t jobs, all of index at most j. Then $M[0][t] = M[j][0] = 0$ and

$$M[j][t] \;=\; \max\big(\, M[j-1][t],\;\; w_j + M[p(j)][t-1] \,\big).$$

Correctness. The case analysis of (b) is unchanged: an optimal schedule either omits j, and then lies inside $\{1, \ldots, j-1\}$ with at most t jobs, or contains j, and then its remaining at most $t - 1$ jobs lie inside $\{1, \ldots, p(j)\}$ by the same argument that every other member finishes strictly before s_j. Both branches are achievable. □

The budget has to sit in the state rather than be patched on afterwards. Two prefixes can have the same best weight while reaching it with different numbers of jobs, and which of those is preferable depends on how much budget is left, so the single number $M[j]$ of part (b) does not carry enough information to extend.

Time and space. $O(n \log n)$ for the sort and the predecessors, then $O(nk)$ for the table. If only the value is wanted, fill the table column by column in t: each column needs only the previous one, so two arrays of length $n + 1$ suffice and the space is $O(n)$.

The shortcut fails. Take $k = 1$ and the three jobs

$$\big([0,1],\, w = 2\big), \qquad \big([2,3],\, w = 2\big), \qquad \big([0,3],\, w = 3\big).$$

The unrestricted optimum is $\{[0,1],[2,3]\}$ with weight 4; discarding one of them leaves weight 2. But the best single job is $[0,3]$, with weight 3.

The shortcut assumes that the optimal schedule for budget k sits inside the optimal schedule for a larger budget, so that shrinking one gives the other. That is false: the optima for different budgets need not be nested, and here the $k = 1$ optimum $\{[0,3]\}$ shares no job at all with the unrestricted optimum. No rule for deleting from a larger solution can recover it, which is why the budget belongs in the state.