WARNING: if you haven’t at least tried hard to solve the practice problems before reading these solutions, you are missing the point. If you can’t make any progress, talk to me or to the TAs before reading these solutions. Otherwise, you should come up with a solution of your own that you can compare to the one shown here.

1. Here is a greedy algorithm for this problem: place the first interval at \([x_1, x_1 + 1]\), remove all points in \([x_1, x_1 + 1]\), and then repeat this process on the remaining points. This algorithm is clearly \(O(n)\). We now prove it correct.

**Greedy Choice Property:** Let \(\Pi\) be any optimal solution, and suppose \(\Pi\) places its leftmost interval at \([x, x + 1]\). It must be that \(x \leq x_1\), since any feasible solution covers the leftmost point \(x_1\). Let \(\Pi^*\) be the solution obtained by replacing this first interval with \([x_1, x_1 + 1]\). The new interval still covers every point between \(x\) and \(x + 1\), since there are no points to the left of \(x_1\), and \(x_1 + 1 \geq x + 1\). Hence, the new solution \(\Pi^*\) remains feasible. Moreover, \(\Pi^*\) uses the same number of intervals as does \(\Pi\) and so is still optimal.

**Inductive Structure:** Let \(P\) be the original problem. After picking the interval \([x_1, x_1 + 1]\) and removing the points it covers, we are left with subproblem \(P'\): find an optimal solution that covers all points to the right of \(x_1 + 1\). Any solution to \(P'\) can be feasibly combined with the greedy choice because all points to the left of \(x_1 + 1\) are already covered.

**Optimal Substructure Property:** Let \(\Pi'\) be an optimal solution to \(P'\), and let \(\Pi = \Pi' \cup \{[x_1, x_1 + 1]\}\). Because we can decompose the objective as

\[
\text{cost}(\Pi) = \text{cost}(\Pi') + 1,
\]

we may apply the standard contradiction argument to show that solution \(\Pi\) is also optimal.

2. Here is a greedy algorithm for this problem: go as far as possible before stopping for gas. Here is some pseudocode:

\[
S \leftarrow \emptyset \\
\text{last} \leftarrow 0 \\
\text{for } i = 1 \text{ to } n \\
\quad \text{if } d_i - \text{last} > m \\
\quad \quad S \leftarrow S \cup \{i - 1\} \\
\quad \quad \text{last} \leftarrow d_{i-1}
\]

The above algorithm is clearly \(O(n)\); we now prove it correct.

**Greedy Choice Property:** Let \(S\) be any optimal solution, and let its stops be \(\{s_1 \ldots s_k\}\). Let \(g\) be the first stop chosen by the greedy algorithm, with distance \(d_g\). If \(s_1 = g\), then \(S\) contains the greedy choice, and we are done. Otherwise, let \(S'\) be a solution that replaces \(s_1\) by \(g\). We know that \(d_g \leq m\) and that \(d_g \geq d_1\); hence, the car does not run out of gas before \(g\) or between \(g\) and \(s_2\) (since
\[ d_2 - g \leq d_2 - d_1 \leq m \), and \( S' \) is still feasible. Moreover, \( S' \) contains the same number of stops as \( S \) and so is also optimal.

**Optimal Substructure Property:** Let \( P \) be the original problem. After stopping at station \( g \) at distance \( d_g \), the subproblem \( P' \) that remains is to get from \( g \) to Denver, with the subset of possible stops being all those after \( g \). Let \( S' \) be an optimal solution to \( P' \). Because cost(\( S \)) = cost(\( S' \)) + 1, the solution \( S \) is also optimal.

3. Here is a greedy algorithm: put as many songs as possible onto one CD without exceeding the \( m \)-minute limit. Then, close this CD, start a new one, and repeat the process for the remaining songs. The decision to close a CD can be made in constant time per song if we keep a running total of how much time we have used on the CD so far; hence, the algorithm takes \( O(n) \) total time. We now prove it correct.

**Greedy Choice Property:** Let \( S \) be an optimal solution in which the first CD holds the first \( k \) songs. Suppose the greedy algorithm puts the first \( g \) songs on this CD. If \( g = k \), we are done; otherwise, \( g > k \), and we modify \( S \) to produce a solution \( S' \) by moving songs from the second and later CD’s to the first CD until it contains \( g \) songs. If the greedy algorithm could put the first \( g \) songs on a CD, these songs must fit in \( m \) minutes, and no other CD’s total time increases; hence, the solution is still feasible. Moreover, the number of CD’s used by \( S' \) is at most as many as for \( S \), so the solution is still optimal.

**Inductive Structure:** Let \( P \) be the original problem. After we close CD 1, we have the subproblem \( P' \) of putting the remaining songs, from \( g + 1 \) to \( n \), on as few CD’s as possible. Any feasible solution to the subproblem is compatible with our choice for the first CD.

**Optimal Substructure Property:** Let \( S' \) be an optimal solution to \( P' \). Observe that cost(\( S \)) = cost(\( S' \)) + 1, and apply the standard contradiction argument.

4. An optimal greedy algorithm is the so-called STCF method, short for “Shortest-Time-to-Completion First.” At each point, we schedule the available job with the shortest remaining processing time. We may implement this algorithm as follows. Let \( PQ \) be a priority queue.

Create a list \( L \) of all jobs sorted by start times
Let \( t \) be the minimum start time of all jobs
While \( L \) is not empty
   Remove all jobs with minimum start time from \( L \)
   Insert these jobs into \( PQ \) with a key of the processing time
   Let \( p = PQ\text{-min()} \)
   Let job \( i \) be a job in \( PQ \) with a minimum key (remaining processing time)
   Let \( s \) be the minimum start time of jobs remaining in \( L \)
   Run job \( i \) from time \( t \) to \( t' = \min(s, p) \)
   Decrease the key of job \( i \) by \( t' - t \)
   If key for job \( i \) is 0 then remove job \( i \) from \( PQ \)
   Let \( t = t' \)

The time to create \( L \) is \( O(n \log n) \). The body of the while loop takes \( O(\log n) \) time. The number of iterations of the loop is bounded by \( 2n \), since a change in \( t \) is made only when a job completes or a new job is released. Hence, the main loop has a total execution time of \( O(n \log n) \), giving an overall time complexity of \( O(n \log n) \).
We now prove that this greedy algorithm is optimal. We first argue that there exists an optimal solution that schedules some job at earliest release time \( t \). Take an arbitrary optimal solution \( \hat{P}_i \) in which the first job is scheduled at time \( t' > t \). Let job \( i \) be a job with release time \( t \). Job \( i \) must be run somewhere in \( \hat{P}_i \). By moving a portion of \( i \) to run from \( t \) to \( t' \) (or until job \( i \) completes) we obtain a solution \( P \) which has cost at most that of \( \hat{P}_i \).

**Greedy Choice Property:** Let \( J_g \) be the first job scheduled in the greedy solution, and assume it is scheduled from \( t \) to \( t_g \) (at which time it is either completed or preempted). As argued above, there must be some optimal solution \( \Pi \) which schedules a job at time \( t \). Let \( J_s \) be the first job scheduled in optimal solution \( \Pi \), and suppose this job runs from time \( t \) to \( t_s \). We now argue that there is an optimal solution \( \Pi^* \) in which job \( J_g \) runs from time \( t \) to \( t' = \text{min}(t_g, t_s) \).

Suppose not; that is, assume \( J_g \neq J_s \). Let time \( t'' \) be the time in \( \Pi \) when job \( J_g \) is completed. We proceed using a proof by cases.

**Case 1:** Job \( J_s \) completes after time \( t'' \). By definition of \( t'' \), job \( J_g \) completes by then. Hence, at least \( t' - t \) units of \( J_g \) must be run in \( \Pi \) by time \( t'' \). We obtain \( \Pi^* \) from \( \Pi \) by interchanging the last \( t' - t \) units of \( J_g \) with the units of \( J_s \) run from \( t \) to \( t' \). The completion time of \( J_g \) is only improved by this. Furthermore, since \( J_s \) completes after \( t'' \), its completion time is unchanged. Hence cost(\( \Pi^* \)) \( \leq \) cost(\( \Pi \)), and so \( \Pi^* \) is an optimal schedule that makes the greedy choice (i.e. that schedules job \( J_g \) from \( t \) to \( t' \)).

**Case 2:** Job \( J_s \) completes before time \( t'' \). By the greedy choice property from \( t \) to \( t'' \), \( J_s \) must run at least as long as \( J_g \); otherwise, \( J_s \) would have been run instead of \( J_g \) in the greedy algorithm. Hence, we can create schedule \( \Pi^* \) from \( \Pi \) by interchanging the portion of the schedule when either \( J_g \) or \( J_s \) runs in \( \Pi \) to first run all of \( J_g \) and then run all of \( J_s \). The completion time for \( J_g \) in \( \Pi^* \) is at most the completion time for \( J_s \) in \( \Pi \). Furthermore, the completion time for \( J_s \) in \( \Pi^* \) must be equal to the completion time of \( J_g \) in \( \Pi \). Hence cost(\( \Pi^* \)) \( \leq \) cost(\( \Pi \)).

**Inductive Structure:** Let \( P \) be the original problem. Subproblem \( P' \) is obtained by reducing the processing time of \( J_g \) by \( t' - t \) and changing the start time of all jobs that are less than \( t' \) to be \( t' \). (When the processing time of a job is 0, it is removed from the subproblem.)

**Optimal Substructure Property:** Let \( \Pi' \) be an optimal solution to \( P' \). We first consider when \( J_g \) completes by time \( t' \). In this case

\[
\text{cost}(\Pi) = \text{cost}(\Pi') + t'.
\]

The other possibility is that \( J_g \) does not complete by time \( t' \). In this case

\[
\text{cost}(\Pi) = \text{cost}(\Pi').
\]

In both cases, for a solution \( \Pi \) which minimizes cost(\( \Pi \)), \( \Pi' \) contained within \( \Pi \) must minimize cost(\( \Pi' \)), and so the optimal substructure property follows.