1. Problem: interval scheduling (scheduling access to a resource that cannot be shared)
   (a) A first example of an optimization problem
   (b) Given as input a set of time intervals \( \{[s_i, f_i] : i = 1, \ldots, n \} \)
   (c) Candidate solutions: subsets of intervals
   (d) Constraints: intervals may not overlap ("conflict")
   (e) Feasible solutions: a subset of intervals that do not overlap with one another
   (f) Objective: maximize the size of the subset (with no conflicts)
   (g) An algorithm for interval scheduling is considered to be correct if it returns conflict-free subsets of maximum size

2. Algorithm style: “Greedy”
   (a) Roughly: build up a solution via local, myopic choices that maximize some simpler (easy-to-optimize) objective
   (b) Possible greedy algorithms for interval scheduling: earliest start, earliest finish, fewest conflicts, etc.
   (c) Most of these are not correct, i.e., do not obtain maximum size subsets.

3. “Stays-ahead” analysis of earliest finish
   (a) Earliest-finish is a correct greedy algorithm, which can be made to run in time \( O(n \log n) \).
   (b) Prove by induction that for every other possible feasible solution \( O \), the \( r \)th interval (in order of increasing finish time) in \( O \) finishes at or after the finish time of the solution \( A \) returned by earliest-finish.
   (c) Therefore, if \( O \) has an \( r + 1 \)th interval, the earliest-finish algorithm can add another interval.

4. Problem: weighted interval scheduling
   (a) Intervals now have a value attached, and the objective is to maximize the total value of the chosen subset.
   (b) Earliest-finish is not correct for weighted interval scheduling, and I don’t know of any correct greedy algorithm for this problem.
   (c) A more powerful style of algorithm will work: “dynamic programming” – roughly:
      i. write down an obviously correct but inefficient recursive algorithm
      ii. Record/“memoize” the solutions to subproblems to obtain a polynomial running time.