Reminders  Collaboration is permitted, but you must write the solutions by yourself without assistance, and be ready to explain them orally to a member of the course staff if asked. You must also identify your collaborators. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Exercises  These should not be handed in, but the material they cover may appear on exams.

• In the lectures of Monday and Wednesday, November 9 and 11, we discussed various strategies for the Interval Scheduling problem (called “Activity Selection” in CLRS).
  Consider the problem of implementing the greedy strategy which first selects the interval with the fewest conflicts. Given a set of intervals, give algorithms that find:
  – A time $t$ which is contained in the maximum number of intervals from the set. Your algorithm should run in $\Theta(n)$ time.
  – (Harder) The conflict numbers $c_1,\ldots,c_n$, where $c_i$ is the number of intervals in the set which overlap interval $i$, not including $i$ itself. (The straightforward algorithm here takes $\Theta(n^2)$ time but there is faster algorithm that runs in time $O(n \log n)$.)

• Recall that the binomial coefficient
  $${n \choose k} = \begin{cases} 
(n-1 \choose k-1) + (n-1 \choose k) & \text{if } 0 < k < n,
1 & \text{otherwise}.
\end{cases}$$
  Give a dynamic programming algorithm for calculating the binomial coefficient on input $n$ and $k$. Analyze time and space complexity of your algorithm. You may assume that addition can be performed in constant time.

• Many problems on DAGs (direct acyclic graphs) can be solved using dynamic programming. Consider an algorithm for the “critical path” problem from the most recent programming assignment. Let $OPT(u)$ denote the weight of the heaviest path beginning at node $u$. Show that $OPT(u)$ can be computed from the values $OPT(v)$ for the nodes $v$ adjacent to $u$.
  Give an algorithm for critical path that first sorts the DAG topologically, then computes the values $OPT(u)$ for all $u$. Finally, use this information to output the critical path.
Problems to be handed in. Please submit each problem on a separate sheet of paper.

1. (Hike Planning) You are planning a multi-day hike along a trail that is \( L \) miles long. You can hike up to \( d \) miles a day. Campgrounds are located at distances \( x_1, x_2, ..., x_n \) from the start of the trail. We say that a set of campgrounds is valid if the distance between each adjacent pair is at most \( d \) miles, the first one is at most \( d \) miles from the start of the trail, and the last one is at most \( d \) miles from the end of the trail (in other words, you can make it across the whole trail, stopping only at these campgrounds). Assume that the full set of campgrounds is valid.

(a) Give a greedy algorithm for finding a minimum valid set of campgrounds.
(b) Use the “greedy stays ahead” strategy to prove that your algorithm produces an optimal solution.
(c) Analyze the running time of your algorithm.

2. (Giving Change) Consider the “cashier’s problem”: give a specified amount of change using the smallest possible number of coins.

(a) Give pseudocode for the greedy algorithm given in class that takes an integer amount \( a \) and coin denominations \( d_1, ..., d_n \), and returns a set of coins whose value is \( a \).
(b) Suppose that coins come in denominations of 1, 5, 25 and 100. Use the “exchange” strategy to prove that the greedy algorithm finds the smallest possible set of coins.
(c) Now suppose that the coins come in denominations of 1, 5, 10 and 25 and 100 (as in the US). Prove that the greedy algorithm still returns the smallest possible set of coins (an exchange argument also works here but it is a touch more complicated).

3. (Billboards) You are managing the construction of billboards along a stretch of highway. The possible sites for the billboards are given by real numbers \( x_1, ..., x_n \), each of which specifies the position along the highway measured in miles from its western end. Assume that the highway is a straight line. If you place a billboard at location \( x_i \), your company will make a profit of \( r_i > 0 \) dollars.

Regulations require that every pair of billboards be at least 5 miles apart. You’d like to place billboards at a subset of the sites so as to maximize total profit, subject to this restriction. The input is given as a list of \( n \) pairs \( (x_1, r_1), ..., (x_n, r_n) \) where the \( x_i \)’s are sorted in increasing order.

(a) Give counter examples showing that the following greedy approaches do not always find the optimal solution:

i. *Next available location:* put a billboard at \( i = 1 \). From then on, put a billboard at the smallest index \( i \) which is more than five miles from your most recently placed billboard.

ii. *Most profitable first:* Put a billboard at the most profitable location. From then on, place a billboard at the most profitable location not ruled out by your current billboards.
(b) Give a dynamic programming algorithm for this problem. Analyze the space and time complexity of your algorithm.

To make it easier to present your answer clearly, try to follow the steps below (as with any design process, you may have to go back and forth a bit between these steps as you work on the problem):

i. Clearly define the subproblems that you will solve recursively (note: weighted interval scheduling should be a good source of inspiration here).

ii. Give a recursive formula for the solution to a given subproblem in terms of smaller subproblems. Explain why the formula is correct.

iii. Give pseudocode for an algorithm that calculates the profit of the optimal solution. Analyze time/space and explain why the algorithm is correct (based on previous parts).

iv. Give pseudocode for an algorithm that uses the information computed in the previous part to output an optimal solution. Analyze time/space and explain why the algorithm is correct.

4*. (Extra credit) We consider a variant of the scheduling task in Problem 3 from Homework 10. Suppose that you run a photocopy business with a single large machine. You’re faced with a list of n jobs to perform, and you want to order them so as to make your customers as happy as possible. Each job comes a duration \( t_i \) (the time it takes to complete the job) and a positive weight \( w_i \) representing the importance of the customer who ordered the job. Let \( f_i \) be the time at which job \( i \) is completed (so \( f_i \) is the sum of \( t_i \) together with the durations of all the jobs run before job \( i \)). You want to minimize the weighted sum of the finishing times,

\[
\sum_{i=1}^{n} w_i f_i.
\]

(a) (Warm-up, do not hand in) Show that ordering the jobs by either increasing duration or decreasing importance does not necessarily minimize the weighted sum.

(b) Give a greedy algorithm that outputs the ordering with the minimum weighted sum and prove that it is correct.