Design and Analysis of Algorithms

Practice Problems

Stable matchings

Infinite loop.

A past DAA student proposed the following algorithm for the stable marriage problem. Begin with an arbitrary perfect matching $M = \{(b_1, g_1), (b_2, g_2), \ldots, (b_n, g_n)\}$ between boys and girls. Search for two distinct pairs $(b_i, g_i), (b_j, g_j) \in M$ such that b_i and g_j form a rogue couple. If such pairs are found, rematch the four participants by replacing (b_i, g_i) and (b_j, g_j) with (b_i, g_j) and (b_j, g_i). Repeat this process until no more rogue couples remain. This algorithm will certainly succeed if it halts. Prove that, unfortunately, it need not halt.

Worst-case execution.

Prove that the Gale-Shapley algorithm terminates within $(n-1)^2 + 1$ days, and that this bound is tight.

Good people, bad people.

Consider a town with n men and n women, each seeking to marry someone of the opposite gender. Each man has a preference list ranking all the women, and each woman has a preference list ranking all the men. The $2n$ people are divided into two categories: good and bad. Suppose that for some integer $1 \leqslant k \leqslant n-1$, there are k good men and k good women; thus, there are $n-k$ bad men and $n-k$ bad women. Everyone prefers marrying a good person over marrying a bad person. Formally, each preference list ranks all good people of the opposite gender higher than all bad people of the opposite gender. That is, the first k entries are the good people in some order, followed by the $n-k$ bad people in some order.

Show that in every stable matching, every good man is married to a good woman.

Soulmates.

Consider an instance of the stable marriage problem. We call a boy and a girl soulmates if they are matched with each other in every stable matching for this instance. Design a polynomial-time algorithm to determine whether any soulmates exist in this instance, and, if so, to identify them.

True or false.

Decide whether the following statements are true or false. If a statement is true, give a brief explanation. If it is false, provide a counterexample.

(1) Consider an instance of the stable marriage problem with at least two boys and two girls in which there exists a boy b and a girl g such that b is ranked first on the preference list of g, and g is ranked first on the preference list of b. Then, in every stable matching M for this instance, the pair (b, g) belongs to M.

(2) Consider an instance of the stable marriage problem with at least two boys and two girls in which there exists a boy b and a girl g such that b is ranked last on the preference list of g, and g is ranked last on the preference list of b. Then, in every stable matching M for this instance, the pair (b, g) does not belong to M.

Identical preferences.

Prove that a stable marriage instance in which all girls have the same preference list admits exactly one stable matching.

Combining stable matchings.

For a given stable marriage instance, let M_1 and M_2 be two distinct stable matchings. Prove that

- If each boy is assigned the better of his partners in M_1 and M_2, then the result is also a stable matching.
- If each boy is assigned the worse of his partners in M_1 and M_2, then the result is also a stable matching.

Preferences between matchings.

One can extend the notion of preference so that it applies to matchings as well as to individuals. A person x is said to prefer a matching M to a matching M' if x prefers their partner in M to their partner in M'.

Let M and M' be stable matchings, and suppose that b and g are partners in M but not in M'. Prove that one of b and g prefers M to M', while the other prefers M' to M.

Forbidden pairs.

We consider a generalization of the stable marriage problem in which certain boy-girl pairs are explicitly forbidden. Let B be a set of n boys, let G be a set of n girls, and let $F \subseteq B \times G$ be a set of forbidden pairs. Each boy b ranks all girls g such that $(b, g) \notin F$, and each girl g ranks all boys b such that $(b, g) \notin F$.

In this setting, a matching M is stable if it admits none of the following types of instability:

(i) There exist pairs (b, g) and (b', g') in M such that $(b, g') \notin F$, b prefers g' to g, and g' prefers b to b'. (This is the usual kind of instability.)

(ii) There exists a pair $(b, g) \in M$, and a boy b' who is unmatched such that $(b', g) \notin F$, and g prefers b' to b.

(iii) There exists a pair $(b, g) \in M$, and a girl g' who is unmatched such that $(b, g') \notin F$, and b prefers g' to g.

(iv) There exist a boy b and a girl g, both of whom are unmatched, such that $(b, g) \notin F$.

The definition assumes that everyone prefers being married to remaining single. Under these conditions, a stable matching need not be perfect. Now we can ask: For every set of preference lists and every set of forbidden pairs, does there always exist a stable matching?

All-or-nothing participation.

Prove that in a stable marriage instance with forbidden pairs, the boys and the girls are each partitioned into two sets: those who have partners in all stable matchings and those who have partners in none. Consequently, all stable matchings for a given instance have the same size.

Indifference.

The stable marriage problem traditionally assumes that all preference lists are strict. Here we consider a version in which ties are allowed. Let B be a set of n boys and let G be a set of n girls. Each person ranks all members of the opposite gender, but ties may occur. For example, a girl might rank b_1 first, tie b_2 and b_3 for second, and rank b_4 last. We say that g prefers b to b' if b is ranked strictly higher than b' on her preference list. With indifferences, there are two natural notions of instability:

(a) A strong instability in a perfect matching M consists of a boy b and a girl g such that both strictly prefer each other to their respective partner in M. Does there always exist a perfect matching with no strong instability?

(b) A weak instability in a perfect matching M consists of a boy b and a girl g whose partners in M are g' and b', respectively, such that one of the following holds:

- b prefers g to g', and g either prefers b to b' or is indifferent between them; or
- g prefers b to b', and b either prefers g to g' or is indifferent between them.

Does there always exist a perfect matching with no weak instability?

Weak-Pareto optimality.

Prove that for any stable marriage instance, there is no matching — stable or otherwise — in which every boy has a partner whom he strictly prefers to his partner in the boy-optimal stable matching produced by the Gale-Shapley algorithm.

Truthfulness.

We now explore the issue of truthfulness in the stable marriage problem, specifically in the Gale-Shapley algorithm. The basic question is whether a boy or a girl can benefit by misrepresenting their preferences. Prove that:

- A girl may be able to obtain a more desirable partner by falsely reordering her preference list.
- A boy cannot obtain a more desirable partner by falsely reordering his preference list.

Shipping company.

Peripatetic Shipping Lines owns n ships and provides service to n ports. Each ship has a schedule specifying, for each day of the month, which port it visits or whether it is at sea. Assume that the "month" has m days, with $m > n$. Each ship visits each port for exactly one day during the month. For safety reasons, the company enforces the following strict requirement: no two ships may be in the same port on the same day.

The company wishes to perform maintenance on all ships by truncating their schedules. For each ship S_i, there will be a day on which it arrives at a port and remains there for the rest of the month for maintenance. As a result, S_i will not visit any remaining ports on its original schedule.

Given the schedules of all ships, find truncation points such that the safety requirement continues to hold. For example, suppose we have two ships and two ports, and the "month" has four days. Suppose the first ship's schedule is (port P_1, at sea, port P_2, at sea) and the second ship's schedule is (at sea, port P_1, at sea, port P_2). Then the only way to choose truncations is to have the first ship remain in port P_2 starting on day 3, and to have the second ship remain in port P_1 starting on day 2.

Show that such truncations always exist, and give an efficient algorithm to find them.

Competition between television networks.

Suppose there are two television networks, A and B. There are n prime-time programming slots, and each network has n TV shows. Each network must assign each show to a distinct time slot. Each show has a unique rating. A network wins a time slot if its show assigned to that slot has a higher rating than the show assigned by the other network. Let S and T denote the schedules chosen by networks A and B, respectively. The pair (S, T) is called stable if neither network can unilaterally change its schedule to win more time slots. That is, there is no schedule S' such that network A wins more slots with (S', T) than with (S, T), and similarly there is no schedule T' such that network B wins more slots with (S, T') than with (S, T).

For every set of TV shows and ratings, does there always exist a stable pair of schedules? Resolve this question by either

(a) giving an efficient algorithm that produces a stable pair of schedules, or

(b) providing an example for which no stable pair exists.

Residents and hospitals.

Gale and Shapley published their paper on the stable matching problem in 1962, but a version of their algorithm had already been in use for a decade by the National Resident Matching Program to assign medical residents to hospitals. There are m hospitals, each with a certain number of available positions, and n graduating medical students, where n is at least as large as the total number of positions. Each hospital ranks the students, and each student ranks the hospitals. The goal is to assign each student to at most one hospital such that all hospital positions are filled. Some students may remain unmatched. An assignment is stable if neither of the following occurs:

- First type of instability: A student s is assigned a hospital h, another student s' is unassigned, and h prefers s' to s.
- Second type of instability: Students s and s' are assigned to hospitals h and h', respectively, such that h prefers s' to s, and s' prefers h to h'.

Show that a stable assignment always exists, and give an efficient algorithm to find one.

Greedy algorithms

Reservation.

You have a resource — it may be a lecture room, a supercomputer, or an electron microscope — and many people request to use the resource for periods of time. A request takes the form: Can I reserve the resource starting at time s until time f? We will assume that the resource can be used by at most one person at a time. A scheduler wants to accept a subset of these requests, rejecting all others, so that the accepted requests do not overlap in time. The goal is to maximize the number of requests accepted.

More formally, there are n requests labeled $1, 2, \ldots, n$, with each request i specifying a start time s_i and a finish time f_i. Naturally, we have $s_i < f_i$ for all i. Two requests i and j are compatible if the requested intervals do not overlap: that is, either request i is for an earlier time interval than that of request j, i.e., $f_i \leqslant s_j$, or request j is for an earlier time interval than that of request i, i.e., $f_j \leqslant s_i$. More generally, a subset A of requests is compatible if all pairs of distinct requests $i, j \in A$ are compatible. Describe an $O(n \log n)$-time greedy algorithm for selecting a compatible subset of requests of maximum possible size.

Lifeguard shift.

A public swimming pool must be supervised continuously from time L to time R. There are n lifeguards available, and each lifeguard i is willing to work only during a specific time interval $[s_i, f_i]$, where $s_i < f_i$. Your task is to hire a minimum number of lifeguards such that every moment of time between L and R is covered by at least one lifeguard on duty.

Formally, you are given a set of n closed intervals on the real line,

$$I = \{[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]\},$$

where each interval $[s_i, f_i]$ satisfies $s_i < f_i$. You are also given a target interval $[L, R]$ with $L < R$. The goal is to select a minimum-cardinality subset $I' \subseteq I$

such that

$$[L, R] \subseteq \bigcup_{[s,f] \in I'} [s, f].$$

In other words, every point $x \in [L, R]$ must be contained in at least one selected interval. If no such subset exists, report that it is impossible to fully cover $[L, R]$.

Describe an $O(n \log n)$-time greedy algorithm to solve this problem.

Driving from Budapest to Barcelona.

We drive an automobile from Budapest to Barcelona. When full, our car's gas tank holds enough gas to travel m miles, and our map gives the distances between gas stations along the route. We wish to make as few gas stops as possible along the way. Give an efficient greedy method to determine at which gas stations we should stop, and prove that your strategy yields an optimal solution.

Exponential payoff.

Suppose you are given two sequences A and B, each containing n positive integers. You may reorder each sequence arbitrarily. After reordering, let a_i be the i^{th} element of A, and let b_i be the i^{th} element of B. You then receive a payoff of $a_1^{b_1} a_2^{b_2} \cdots a_n^{b_n}$. Give an $O(n \log n)$-time greedy algorithm that maximizes your payoff. Prove that your algorithm yields an optimal solution.

Dance school.

There are n boys and n girls in a dance school. Design efficient greedy algorithms, one for each of the following objectives: given the heights of the boys and the girls, pair each boy with a unique girl so that

(A) the maximum

(B) the sum

of the absolute differences between the heights of the boy and the girl in each pair is minimized.

Licenses.

Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain at most one licence per month. Each license is currently selling for a price of \$100. However, the prices are increasing according to exponential growth curves. In particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100 \cdot r_j^t$. We assume that all growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$, even though they start at the same price of \$100.

Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money spent is minimized? Describe an $O(n \log n)$-time greedy algorithm that takes the growth rates $r_1, r_2, \ldots, r_n$, and computes an optimal purchase order.

Homework assignments.

At the start of the semester, you are given n homework assignments $a_1, a_2, \ldots,$ a_n. You may complete the assignments in any order, but you must turn in exactly one assignment per week over the n weeks of the semester. Each assignment a_i has a difficulty d_i. You may assume the difficulty values are positive and distinct. If you turn in assignment a_i in week j, you receive $d_i(n-j+1)$ points. Describe an $O(n \log n)$-time greedy algorithm for finding an order of the assignments that maximizes your total score.

Average completion time.

Suppose you are given a set of n tasks $a_1, a_2, \ldots, a_n$, where task a_i requires a contiguous time interval of length p_i to process. You have one computer, and it can run only one task at a time. Each task must run non-preemptively, that is, once task a_i starts, it must run continuously until it is completed. Let c_i be the completion time of task a_i, that is, the time at which task a_i finishes processing. Your goal is to order the tasks so as to minimize the average completion time:

$$\frac{1}{n} \sum_{i=1}^{n} c_i.$$

For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5$, $c_1 = 5 + 3 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$.

Give an $O(n \log n)$-time greedy algorithm that schedules the tasks to minimize the average completion time. Prove that your algorithm yields an optimal solution.

Photocopying service.

A small business — say, a photocopying service with a single large machine — faces the following scheduling problem. Each morning, they receive a set of jobs from customers and want to process them in an order that keeps customers happiest. Customer i's job requires a contiguous time interval of length t_i to process. Given a schedule, i.e., an ordering of the jobs, let C_i denote the finishing time of job i. For example, if job j is done first, then $C_j = t_j$; and if job j is done immediately after job i, then $C_j = C_i + t_j$. Each customer i also has a weight w_i that represents their importance to the business. The company wants to minimize the weighted sum of the completion times.

Formally, you are given a set of n jobs, each with a processing time t_i and a weight w_i. Design an $O(n \log n)$-time greedy algorithm to order the jobs so as to minimize

$$\sum_{i=1}^{n} w_i C_i.$$

Prove that your algorithm yields an optimal solution.

Greedy change-making.

In the change-making problem, you are given a set of coin denominations and asked to determine, for a given amount N, the minimum number of coins needed to pay N. For example, given coins of denominations $1, 2, 4, 8, 16$, the minimum representation of $N = 19$ requires 3 coins $(16 + 2 + 1)$. In some cases, the solution can be found by a greedy strategy that uses as many of the coins of the largest denomination as possible, then as many of the next largest as possible, and so on. Although this greedy strategy fails to be optimal in general, prove that it works for the set of coin denominations given above.

Divide and conquer

Significant inversions.

Recall the problem of finding the number of inversions: we are given a sequence of n numbers $a_1, a_2, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair of indices $i < j$ such that $a_i > a_j$. We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. We call a pair a significant inversion if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$-time divide and conquer algorithm to count the number of significant inversions in a sequence of n pairwise distinct numbers $a_1, a_2, \ldots, a_n$.

Maximum contiguous subarray.

Give an $O(n \log n)$-time divide and conquer algorithm to find a contiguous subarray within a one-dimensional array $A[1 : n]$ of real numbers that has the largest possible sum, i.e., find indices $1 \leqslant i \leqslant j \leqslant n$ such that

$$A[i] + A[i+1] + \cdots + A[j]$$

is maximized.

Fraud detection.

Suppose you are consulting for a bank that is concerned about fraud detection, and they present you with the following problem. They have a collection of n bank cards that they have confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we say that two bank cards are equivalent if they correspond to the same account. It's very difficult to read the account number directly from a bank card, but the bank has a high-tech "equivalence tester" that takes two bank

cards and, after performing some computations, determines whether they are equivalent.

Their question is the following: among the collection of n cards, is there a set of more than $n/2$ cards that are all equivalent to one another? Assume that the only feasible operation you can perform is to pick two cards and plug them into the equivalence tester. Show how to decide the answer to their question using only $O(n \log n)$ invocations of the equivalence tester.

Skyline.

You are given a set of n buildings in a city skyline. Each building is represented by a triple $B_i = (L_i, R_i, H_i)$ where L_i is the x-coordinate of the left edge, R_i is the x-coordinate of the right edge, and H_i is the height of the building with $L_i < R_i$ and $H_i > 0$.

The skyline is defined as the outer contour formed by the union of all buildings when viewed from a distance. It is represented as a sequence of critical points $(x_1, h_1), (x_2, h_2), \ldots, (x_k, h_k)$ where the height of the skyline changes at each x_i.

Design an $O(n \log n)$-time divide and conquer algorithm to compute the skyline of the set of n buildings.

Non-dominated points.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane, where each point $p_i = (x_i, y_i)$ has real-valued coordinates. We may assume that all x-coordinates are distinct and all y-coordinates are distinct. We say that a point $p_j = (x_j, y_j)$ dominates another point $p_k = (x_k, y_k)$ if $x_j > x_k$ and $y_j > y_k$.

Design an $O(n \log n)$-time divide and conquer algorithm to compute all non-dominated points in P, i.e., all points in P that are not dominated by any other point in P.

Convex hull.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane, where each point $p_i = (x_i, y_i)$ has real-valued coordinates. The convex hull of P is defined as the smallest convex polygon that contains all points in P.

Design an $O(n \log n)$-time divide and conquer algorithm to compute the convex hull of P. Your algorithm should output the vertices of the convex hull in counterclockwise order.

Hidden surface removal.

Hidden surface removal is a problem in computer graphics that hardly needs an introduction: when Bob is standing in front of Alice, you should be able to see Bob but not Alice; when Alice is standing in front of Bob, the opposite should occur. The magic of hidden surface removal is that one can often compute visibility faster than intuition might suggest.

Here is a clean geometric example. You are given n nonvertical lines $L_1, L_2, \ldots, L_n$ in the plane, where line L_i is specified by the equation $y = a_i x + b_i$. We may assume that no three lines intersect at a single point. We say that line L_i is uppermost at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0, i.e., $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say that line L_i is visible if there exists some x-coordinate at which it is uppermost — intuitively, some portion of it can be seen when looking down from "$y = \infty$". Give a divide and conquer algorithm that takes n lines as input and, in $O(n \log n)$ time, returns all the lines that are visible.

Missing element.

We are given a complete binary search tree with $n = 2^k - 1$ nodes for some positive integer k. The keys stored in the nodes are distinct elements of the set $\{1, 2, \ldots, 2^k\}$. Design an $O(\log n)$-time algorithm to determine the element of the set that is not present among the keys stored in the tree.

Local minimum in a tree.

Consider an n-node complete binary tree T, where $n = 2^k - 1$ for some positive integer k. Each node v of T is labeled with a real number x_v. You may assume that all labels are distinct. A node v of T is a local minimum if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge. You are given such a complete binary tree T, but the labeling is specified only implicitly: for each node v, you can determine the value x_v by probing node v. Show how to find a local minimum of T using only $O(\log n)$ probes.

Local minimum on a grid.

Consider an $n \times n$ grid graph G. An $n \times n$ grid graph is the adjacency graph of an $n \times n$ chessboard. More precisely, it is a graph whose node set consists of all ordered pairs of integers (i, j), where $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$. Two

nodes (i, j) and (k, l) are joined by an edge if and only if $|i-k|+|j-l| = 1$. Each node v is labeled by a real number x_v. You may assume that all labels are distinct. A node v of G is a local minimum if x_v is less than the label x_w for all nodes w that are joined to v by an edge. You are given such a grid graph G, but the labeling is specified only implicitly: for each node v, you can determine the value x_v by probing node v. Show how to find a local minimum of G using only $O(n)$ probes. (Note that G has n^2 nodes.)

Dynamic programming

Assembly lines.

This manufacturing problem asks for the fastest way through a factory. There are two assembly lines, each with n stations. The j^{th} station on line i is denoted $S_{i,j}$, and the assembly time at that station is $a_{i,j}$. An automobile chassis enters the factory and is placed on line i where $i \in \{1, 2\}$, taking e_i time. After passing through the j^{th} station on a line, the chassis proceeds to the $(j+1)^{\text{th}}$ station on either line. There is no transfer cost if it stays on the same line, but transferring to the other line after station $S_{i,j}$ takes $t_{i,j}$ time. After exiting the n^{th} station on a line, the completed automobile requires x_i time to leave the factory. The problem is to determine which stations to choose from line 1 and which to choose from line 2 in order to minimize the total time for one automobile to pass through the factory. Describe a dynamic programming algorithms to solve this problem in $O(n)$ time.

Checkerboard.

Suppose you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step, you may move the checker to one of three squares:

1. the square immediately above,
2. the square that is one up and one to the left (only if the checker is not already in the leftmost column),
3. the square that is one up and one to the right (only if the checker is not already in the rightmost column).

Each time you move from square x to square y, you receive $f(x, y)$ dollars. You are given $f(x, y)$ for all pairs (x, y) for which a move from x to y is legal. Give an $O(n^2)$-time dynamic programming algorithm that determines

a sequence of moves that takes the checker from some square on the bottom edge to some square on the top edge while collecting as many dollars as possible. Your algorithm may choose any starting square on the bottom edge and any destination square on the top edge to maximize the total reward.

Cutting rods.

An enterprise buys long steel rods and cuts them into shorter rods, which it then sells. Each cut is free. The management wants to determine the best way to cut up the rods.

The enterprise maintains a table that lists, for $i = 1, 2, \ldots$, the price p_i in dollars charged for a rod of length i inches. The length of each rod is always an integer.

The problem is defined as follows. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces. If the price p_n for a rod of length n is sufficiently large, an optimal solution may require no cutting at all.

Design an $O(n^2)$-time dynamic programming algorithm to solve the problem.

Air freight scheduling.

Suppose you are consulting for a company that manufactures PC equipment and ships it to distributors all over the continent. For each of the next n weeks, the company has a projected supply s_i of equipment (measured in kilograms) that must be shipped by an air freight carrier. Each week's supply can be carried by one of two air freight companies, A or B:

- Company A charges a fixed rate r per kilogram, so it costs rs_i to ship a week's supply s_i.
- Company B charges a fixed amount c per week, independent of weight. However, contracts with company B must be made in blocks of four consecutive weeks.

A schedule is a choice of air freight company (A or B) for each of the n weeks, with the restriction that whenever company B is chosen, it must be chosen for a block of four contiguous weeks. The cost of the schedule is the total amount paid to both companies. Give an efficient dynamic programming algorithm that takes the sequence $s_1, s_2, \ldots, s_n$ and returns a schedule of minimum cost.

File replication.

Suppose we want to replicate a file across a collection of n servers labeled $S_1, S_2, \ldots, S_n$. Placing a copy of the file at server S_i incurs a placement cost of c_i, where $c_i > 0$ is an integer. If a user requests the file from server S_i and no copy is present at this server, then servers $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$ are searched in increasing order until a copy is found at some server S_j, where $j > i$. This results in an access cost of $j-i$. The access cost is 0 if S_i already holds a copy. We require that a copy of the file be placed at server S_n so that all searches eventually terminate. The goal is to place copies of the file at selected servers in order to minimize the total cost, defined as the sum of all placement costs plus the sum of all access costs over all n servers. Formally, a configuration is a choice, for each server S_i with $i = 1, 2, \ldots, n-1$, of whether to place a copy at S_i. (A copy is always placed at S_n.) Give an efficient dynamic programming algorithm to find a configuration of minimum total cost.

Rising trend.

Your friends have been studying the closing prices of technology stocks and are looking for interesting patterns. They define a rising trend as follows. Let the closing prices for a given stock over n consecutive days be $P[1], P[2], \ldots, P[n]$. A rising trend is a subsequence $P[i_1], P[i_2], \ldots, P[i_k]$ with $1 = i_1 < i_2 < \cdots < i_k \leqslant n$ and $P[i_j] < P[i_{j+1}]$ for each $j = 1, 2, \ldots, k-1$. Thus, a rising trend begins on the first day and is not necessarily contiguous, but the prices strictly increase over the chosen days. Give an efficient dynamic programming algorithm that takes $P[1], P[2], \ldots, P[n]$ and returns the length of the longest rising trend.

Longest increasing subsequence.

Give an $O(n^2)$-time dynamic programming algorithm to find a longest strictly increasing subsequence of a sequence of n numbers.

Low-stress jobs, high-stress jobs.

Suppose you are managing a consulting team of expert computer hackers, and each week you must choose a job for them to undertake. The available jobs are either low-stress or high-stress. If you select a low-stress job in week i, you earn a revenue of $\ell_i > 0$ dollars. If you select a high-stress job in week i, you earn a revenue of $h_i > 0$ dollars. However, in order to take a high-stress job in week i, the team must perform no job in week $i-1$; they

require a full week of preparation. Low-stress jobs may be taken even if a job of either type was performed in the previous week.

A plan specifies, for each week i, one of the choices "low-stress", "high-stress", or "none", with the constraint that if "high-stress" is chosen in week $i > 1$, then "none" must have been chosen in week $i - 1$. (It is permissible to choose a high-stress job in week 1.) The value of a plan is computed as follows: add ℓ_i if "low-stress" is chosen in week i, add h_i if "high-stress" is chosen in week i, and add 0 if "none" is chosen in week i. Give a polynomial time dynamic programming algorithm that takes values $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$ and returns a plan of maximum value. Analyze the running time of your algorithm.

Longest palindromic subsequence.

A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples include CIVIC, NOON, and AIBOHPHOBIA (fear of palindromes).

Give an $O(n^2)$-time dynamic programming algorithm that finds a longest palindromic subsequence of a given string of length n.

For example, given the string CHARACTER your algorithm should return the palindrome CARAC, since it is a subsequence of the given string and no longer palindromic subsequence exists.

Shortest palindromic supersequence.

A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples include CIVIC, NOON, and AIBOHPHOBIA (fear of palindromes).

Give an $O(n^2)$-time dynamic programming algorithm that finds a shortest palindromic supersequence containing a given string of length n as a subsequence.

For example, given the string TWENTYONE your algorithm might return the palindrome TWENTOYOTNEWT, since it contains the given string as a subsequence and no shorter palindrome satisfies this condition.

Optimal binary search trees.

You are given a sorted list of distinct keys $k_1, k_2, \ldots, k_n$ and a nonnegative search frequency p_i for each key k_i. Describe an $O(n^3)$-time dynamic programming algorithm for constructing a binary search tree T that contains

the keys $k_1, k_2, \ldots, k_n$ and minimizes the weighted search time

$$\sum_{i=1}^{n} p_i \cdot (\text{search time for } k_i \text{ in } T) = \sum_{i=1}^{n} p_i \cdot ((\text{depth of } k_i \text{ in } T) + 1).$$

Telescope scheduling.

Large, powerful telescopes are precious resources that are typically oversubscribed by astronomers who request times to use them. This is especially true for instruments such as the Hubble Space Telescope, which receives thousands of observation requests each month. In this simplified version of the telescope scheduling problem, we abstract away details such as telescope orientation and requester identity, while retaining the most important scheduling constraints.

The input is a list of observation requests, where each request R_i consists of:

- a requested start time s_i,
- a finish time f_i, and
- a positive benefit value b_i, representing the scientific gain from performing the observation.

Observation R_i must be performed continuously from time s_i to time f_i in order to obtain benefit b_i. Thus, two requests R_j and R_k conflict if the time intervals $[s_j, f_j]$ and $[s_k, f_k]$ overlap. Given a list of n observation requests, schedule a subset of nonconflicting observations to maximize the total benefit. Solve this problem in $O(n \log n)$ time.

Coins-in-a-line.

In the coins-in-a-line game, an even number n of coins — of various denominations — are placed in a row. Two players, Alice and Bob, take turns removing one coin from either end of the row, with Alice going first. The removed coin is added to the player's collection. The player whose collected coins have the larger total value wins. Determine an optimal strategy for Alice to maximize her total value, assuming Bob also plays optimally. Your algorithm should run in $O(n^2)$ time.

Houses-in-a-row.

This is a variant of the coins-in-a-line game. Two competing real estate developers, Alice and Bob, take turns dividing n houses arranged in a row, with Alice going first. On each turn, a player must choose one or more consecutive houses from either the left end or the right end of the remaining row. For example, in a row of houses numbered 1 to 100, Alice might choose houses 1, 2, and 3 in her first turn, after which Bob could choose houses 100 and 99, and so on until all houses are chosen. There is no limit on how many houses may be chosen in a turn, but house values may be positive or negative. Describe an efficient dynamic programming algorithm to determine how Alice can maximize the total net value of the houses she selects, assuming Bob also plays optimally. Analyze the running time of your algorithm.

Piano fingering.

A piano piece consists of a sequence of notes $h_1, h_2, \ldots, h_n$ to be played with one hand. The hand has five fingers, numbered 1 (thumb) through 5 (pinky). For each note h_i, you must choose a finger $f_i \in \{1, 2, 3, 4, 5\}$ to play that note. Certain transitions between fingers are more difficult than others, depending on both the notes being played and the fingers used. You are given a function $c(h_{i-1}, f_{i-1}, h_i, f_i)$ which returns a nonnegative integer representing the difficulty of playing note h_i with finger f_i immediately after playing note h_{i-1} with finger f_{i-1}. Your goal is to minimize the total difficulty of playing the entire piece.

Describe an $O(n)$-time dynamic programming algorithm for choosing a finger sequence $f_1, f_2, \ldots, f_n$ that minimizes

$$\sum_{i=2}^{n} c(h_{i-1}, f_{i-1}, h_i, f_i).$$

Quiz.

Consider a quiz with n questions. For each $i = 1, 2, \ldots, n$, question Q_i has an integral point value $v_i > 0$ and requires $m_i > 0$ minutes to solve. No partial credit is awarded. Design a dynamic programming algorithm that, given $v_1, v_2, \ldots, v_n$, $m_1, m_2, \ldots, m_n$, and V, computes the minimum number of minutes required to earn at least V points on the quiz. Analyze the running time of your algorithm.