Practice Problems 4

Problem 1. A set of n planes are ready to take off at the airport. Let t_i denote the time (duration) that it takes for plane-i to take off. You are to determine an order for them to take off. If a plane is jth in the take-off order, it must wait for the $j-1$ planes that come before. The *total wait time* is the sum of the wait times for all the planes. The objective is to compute a take-off order that *minimizes the total wait time*.

For example, in Fig. 1(a) we show an input with three planes. In Fig. 1(b) we show that the take-off order $(2, 3, 1)$ has a total wait time of 7 units. Plane-2 doesn't wait at all, plane-3 waits 2 units, and plane-1 waits $(2 + 3) = 5$ for the prior two planes.

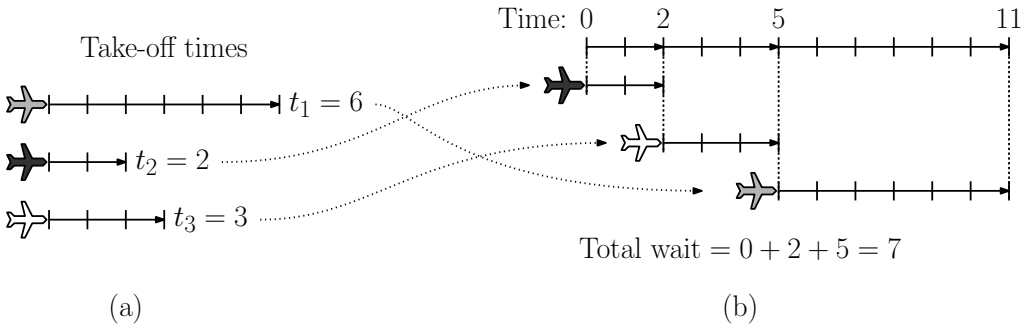

Figure 1: Scheduling planes for take-off.

(a) Design an algorithm, which given an array $t[1..n]$ of take-off times determines the optimum take-off order. Prove your algorithm's correctness and derive its running time. (Prove correctness from *first principles*, without using results from class or homework solutions.) **Hint:** Use a greedy approach and prove correctness by an exchange argument.

(b) Suppose that in addition to the take-off times $t[1..n]$ you are also given the number of passengers $p[1..n]$ on each plane. Delaying a plane with many passengers is bad. Define the *weighted total delay* to be the sum of delays for each plane weighted by the number of passengers on the plane.

For example, in Fig. 2(a) we show an input with three planes. In Fig. 2(b) we show that the take-off order $(2, 3, 1)$ has a total wait time of 360 units. Plane-2 doesn't wait at all, plane-3 has a total weight of $2 \cdot 100$, and plane-1 has a total weight of $32 \cdot (2 + 3) = 32 \cdot 5$ for the prior two planes.

Answer the same problem as in part (a), but for this weighted version of the problem. (Same hint applies. If there are common elements between the two algorithms and their analyses, you can simply explain the modifications needed for this version.)

Problem 2. You are given an integer n and two sequences of nonnegative integers $R = \langle r_1, \ldots, r_n \rangle$ and $C = \langle c_1, \ldots, c_n \rangle$, such that $0 \le r_i, c_j \le n$, and $\sum_i r_i = \sum_j c_i$.

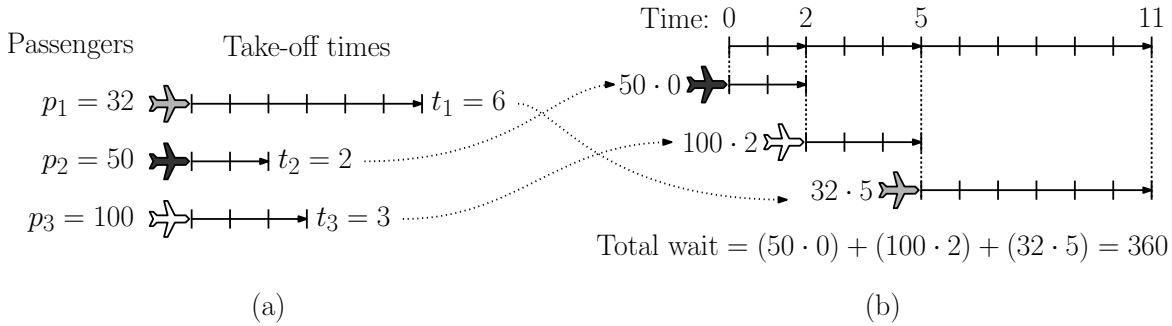

Figure 2: Weighted version of plane scheduling problem.

Given these sequences, you are asked to determine whether it is possible to place pawns on an $n \times n$ chess board such for $1 \leq i, j \leq n$, row i has exactly r_i pawns and column j has exactly c_j pawns (see Fig. 3). If so, specify which squares of the board contain pawns. (There may be many valid solutions, and your algorithm can generate any one of them.)

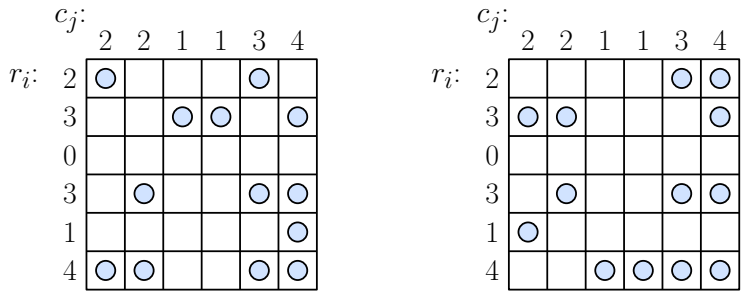

Figure 3: Two possible solutions to Challenge Problem 2 for the inputs $R = \langle 2, 3, 0, 3, 1, 4 \rangle$ and $C = \langle 2, 2, 1, 1, 3, 4 \rangle$.

Show that there exists an algorithm that solves this problem in $O(n^2)$ time. Prove that your algorithm is correct. (Hint: Fill rows one by one (in any order), using a greedy strategy to select which columns to fill.)

Problem 3. Gonzalez's algorithm (the greedy k-center heuristic) is run on a set P of $n = 100$ points in the plane. For $i \geq 1$, let C_i denote the set of centers after i iterations. Let Δ_i denote the maximum distance of any point of P to its closest center in C_i. Let Γ_i denote the minimum distance between any two centers of C_i. Which of the following statements necessarily holds? (Select all that apply.)

(i) $\Delta_4 \leq \Delta_3$

(ii) $\Gamma_4 \leq \Gamma_3$

(iii) $\Gamma_4 \leq \Delta_3$

(iv) $\Gamma_4 \geq \Delta_3$

Problem 4. An interesting feature of Gonzalez's algorithm is that it can be applied even to sets of infinite size. (Formally, we need the set to be closed and bounded, but let's not worry

about these formalities.) In this problem, we will explore an intriguing connection between Gonzalez's algorithm and the concept of fractal dimension.

Consider the sequence of geometric sets shown in Fig. 4 below. If this sequence is carried out in the limit to infinity, the result is an a set T of infinite cardinality, called the *Sierpiński triangle*. This is a famous example of a *fractal*, that is, a self-similar shape whose Hausdorff dimension is a fraction. (The Hausdorff dimension of the Sierpiński triangle is $\log 3/\log 2 \approx 1.585$.)

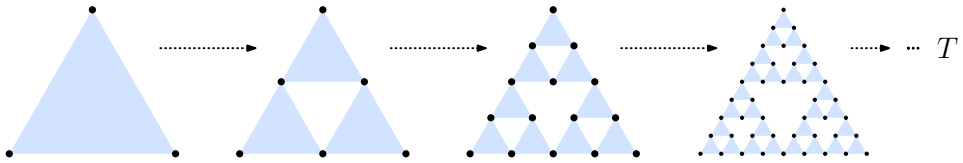

Figure 4: The limit of this process is the Sierpiński triangle.

Let's apply Gonzalez's algorithm to T. Let's assume that T has a side length of 1. Let $G = \{g_1, \ldots, g_k\}$ denote the first k Gonzalez points. Recall that for any $k \geq 0$, $\Delta(G_k)$ denotes the maximum distance of any point of T to its closest point in G_k. Let's start by placing g_1 at the lower-left vertex of T, yielding $\Delta(G_1) = 1$ (see Fig. 5(a)). The next two points will be placed at the other two vertices of the triangle, yielding $\Delta(G_3) = \frac{1}{2}$ (see Fig. 5(b)). The next three points will be placed at the midpoints of the triangle edges, yielding $\Delta(G_6) = \frac{1}{4}$ (see Fig. 5(c)). We can continue this process forever.

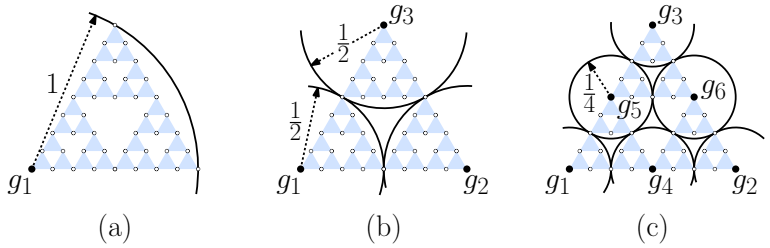

Figure 5: Running Gonzalez on the Sierpiński triangle.

(a) From the above example, it should be clear that after adding a sufficient number of points k, $\Delta(G_k)$ decreases to a power of $\frac{1}{2}$. How many points do we need for this to happen? For any $i \geq 0$, let $k(i)$ denote the minimum number of points such that $\Delta(G_{k(i)}) \leq 1/2^i$. (The example in the figure shows that $k(0) = 1$, $k(1) = 3$, and $k(2) = 6$.)

Give a formula for $k(i)$. (For full credit, your answer should be an exact closed-form formula. For partial credit, you can express $k(i)$ as a recurrence or a closed-form formula that is within a constant factor of the exact value.)

Justify your answer. (That is, explain how you derived it.)

(b) We can use Gonzalez's algorithm to bound the Hausdorff dimension of any fractal. Let's do this for the Sierpiński triangle. Here is the definition of Hausdorff dimension.

> Given a set T and real $r > 0$, define $N_T(r)$ to be the number of balls of radius at most r required to cover T completely. The *Hausdorff dimension* of T is the unique number d such that $N_T(r)$ grows as $1/r^d$, as r approaches zero.

Prove that the Hausdorff dimension of the Sierpiński triangle is $d = \log 3 / \log 2$. (Hint: Use your result from (a). Express the radius r and covering number $N_T(r)$ in terms of i and $k(i)$. Consider the limit as i grows to infinity.)

Problem 5. You are given a collection of intervals $I = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ along the real line (see Fig. 6(a)). You may assume that the interval endpoints are distinct. Your objective is to place the minimum number of pins $X = \{x_1, x_2, \ldots, x_k\}$ to stab all of these intervals (see Fig. 6(b)). A pin x is said to *stab* an interval $[a_i, b_i]$ if $a_i \leq x \leq b_i$. (Note that you can place a pin through either of the interval endpoints.)

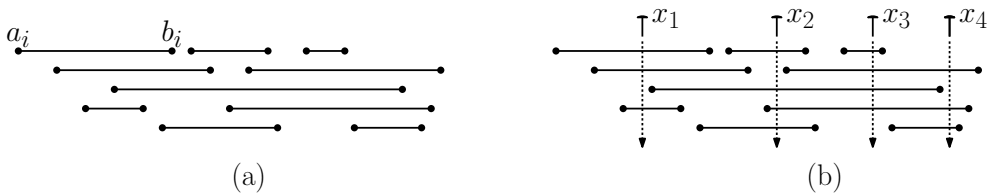

Figure 6: One-dimensional pinning.

(a) Here is an obvious *depth-based greedy heuristic*. Place a pin that stabs the maximum number of intervals, remove all the intervals that this pin stabs, and then repeat on the remaining set of intervals.

Present a counterexample that shows that this depth-based greedy heuristic is *not* optimal. It suffices to give a drawing and a brief explanation. (Hint: There is an example involving 6 intervals where the optimum pinning set consists of 2 pins, but greedy generates 3 pins. Of course, any valid counterexample is acceptable.)

(b) Given a set of n intervals I, let $\text{opt}(I)$ denote the size of the minimum stabbing set and let $\text{depth}(I)$ denote the number of pins generated by the above depth-based. Explain why the depth-based heuristic is essentially the same as the greedy set-cover heuristic. (What are the elements and what are the sets?) Given this, what can you infer about the approximation ratio $\text{depth}(I)/\text{opt}(I)$?

(c) Devise an optimal greedy algorithm for this problem. Justify your algorithm's correctness (both feasibility and optimality), and derive its running time. (The running time is not critical here, as long as it runs in polynomial time.)