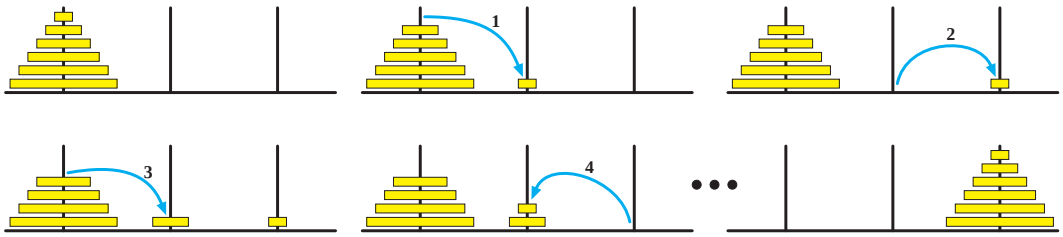
Write your answers in the separate answer booklet.

1. In the well-known *Tower of Hanoi* problem, we have three spikes, one of which has a tower of n disks of different sizes, stacked with smaller disks on top of larger ones. In a single step, we are allowed to take the top disk on any spike and move it to the top of another spike. We are never allowed to place a larger disk on top of a smaller one. Our goal is to move all the disks from one spike to another.

   Hmmm.... You've probably known how to solve this problem since CS 125, so make it more interesting, let's add another constraint: The three spikes are arranged in a row, and we are also forbidden to move a disk directly from the left spike to the right spike or vice versa. In other words, we *must* move a disk either to or from the middle spike at *every* step.



The first four steps required to move the disks from the left spike to the right spike.

   (a) **[4 pts]** Describe an algorithm that moves the stack of n disks from the left needle to the right needle in as few steps as possible.

   (b) **[6 pts] Exactly** how many steps does your algorithm take to move all n disks? A correct Θ-bound is worth 3 points. *[Hint: Set up and solve a recurrence.]*

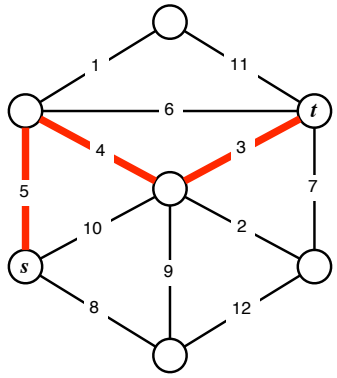
2. Consider a random walk on a path with vertices numbered $1, 2, \ldots, n$ from left to right. At each step, we flip a coin to decide which direction to walk, moving one step left or one step right with equal probability. The random walk ends when we fall off one end of the path, either by moving left from vertex 1 or by moving right from vertex n. In Midterm 2, you were asked to prove that if we start at vertex 1, the probability that the walk ends by falling off the *left* end of the path is exactly $n/(n+1)$.

   (a) **[6 pts] Prove** that if we start at vertex 1, the expected number of steps before the random walk ends is exactly n. *[Hint: Set up and solve a recurrence. Use the result from Midterm 2.]*

   (b) **[4 pts]** Suppose we start at vertex $n/2$ instead. State a tight Θ-bound on the expected length of the random walk in this case. **No proof is required.** *[Hint: Set up and solve a recurrence. Use part (a), even if you can't prove it.]*

3. **Prove** that any connected acyclic graph with n vertices has exactly $n-1$ edges. Do not use the word "tree" or any well-known properties of trees; your proof should follow entirely from the definitions.

4. Consider a path between two vertices s and t in an undirected weighted graph G. The *bottleneck length* of this path is the maximum weight of any edge in the path. The *bottleneck distance* between s and t is the minimum bottleneck length of any path from s to t. (If there are no paths from u to v, the bottleneck distance between s and t is ∞.)



The bottleneck distance between s and t is 5.

Describe and analyze an algorithm to compute the bottleneck distance between every pair of vertices in an arbitrary undirected weighted graph. Assume that no two edges have the same weight.

5. The 5COLOR asks, given a graph G, whether the vertices of a graph G can be colored with five colors so that no edge has two endpoints with the same color. You already know from class that this problem is NP-complete.

   Now consider the related problem 5COLOR±1: Given a graph G, can we color each vertex with an integer from the set $\{0, 1, 2, 3, 4\}$, so that for every edge, the colors of the two endpoints differ by exactly 1 modulo 5? (For example, a vertex with color 4 can only be adjacent to vertices colored 0 or 3.) We would like to show that 5COLOR±1 is NP-complete as well.

   (a) **[2 pts]** Show that 5COLOR±1 is in NP.

   (b) **[1 pt]** To prove that 5COLOR±1 is NP-hard (and therefore NP-complete), we must describe a polynomial time algorithm for *one* of the following problems. Which one?

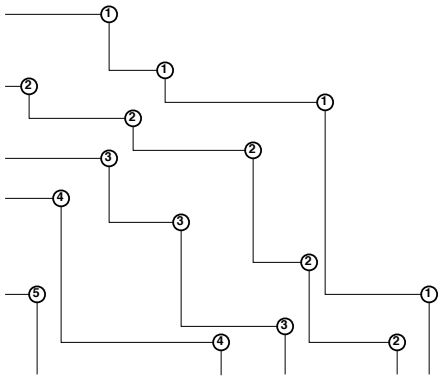
   - Given an arbitrary graph G, compute a graph H such that 5COLOR(G) is true if and only if 5COLOR±1(H) is true.
   - Given an arbitrary graph G, compute a graph H such that 5COLOR±1(G) is true if and only if 5COLOR(H) is true.

   (c) **[1 pt]** Explain briefly why the following argument is not correct.

   > For any graph G, if 5COLOR±1(G) is true, then 5COLOR(G) is true (using the same coloring). Therefore if we could solve 5COLOR±1 quickly, we could also solve 5COLOR quickly. In other words, 5COLOR±1 is at least as hard as 5COLOR. We know that 5COLOR is NP-hard, so 5COLOR±1 must also be NP-hard!

   (d) **[6 pts]** Prove that 5COLOR±1 is NP-hard. *[Hint: Look at some small examples. Replace the edges of G with a simple gadget, so that the resulting graph H has the desired property from part (b).]*

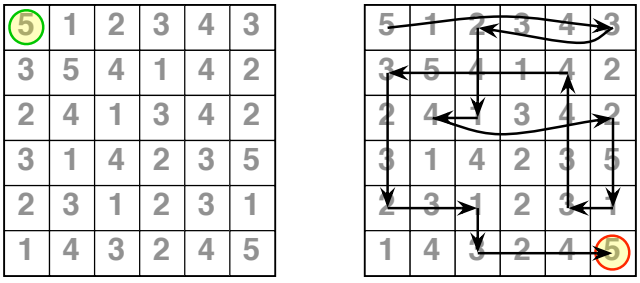
6. Let P be a set of points in the plane. Recall that a point $p \in P$ is *Pareto-optimal* if no other points in P are both above and to the right of p. Intuitively, the sequence of Pareto-optimal points forms a *staircase* with all the other points in P below and to the left. The *staircase layers* of P are defined recursively as follows. The empty set has no staircase layers. Otherwise, the first staircase layer contains all the Pareto-optimal points in P, and the remaining layers are the staircase layers of P minus the first layer.



A set of points with 5 staircase layers

Describe and analyze an algorithm to compute the number of staircase layers of a point set P as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 5.

7. Consider the following puzzle played on an $n \times n$ square grid, where each square is labeled with a positive integer. A token is placed on one of the squares. At each turn, you may move the token left, right, up, or down; the distance you move the token must be equal to the number on the current square. For example, if the token is on a square labeled "3", you are allowed more the token three squares down, three square left, three squares up, or three squares right. You are never allowed to move the token off the board.



A sequence of legal moves from the top left corner to the bottom right corner.

(a) **[4 pts]** Describe and analyze an algorithm to determine, given an $n \times n$ array of labels and two squares s and t, whether there is a sequence of legal moves that takes the token from s to t.

(b) **[6 pts]** Suppose you are only given the $n \times n$ array of labels. Describe how to preprocess these values, so that afterwards, given any two squares s and t, you can determine in $O(1)$ time whether there is a sequence of legal moves from s to t.