Answer four of these seven problems; the lowest three scores will be dropped.

1. Suppose we are given an array $A[1..n]$ with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a *local minimum* if it is less than or equal to both its neighbors, or more formally, if $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are five local minima in the following array:

9	7	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9

   We can obviously find a local minimum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array **must** have at least one local minimum. Why?]*

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 **and prove** a tight Θ-bound on the expected length of the random walk in this case. *[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 $n \geq 2$ vertices has at least two vertices with degree 1. Do not use the words "tree" of "leaf", or any well-known properties of trees; your proof should follow entirely from the definitions.

4. Consider the following sketch of a "reverse greedy" algorithm. The input is a connected undirected graph G with weighted edges, represented by an adjacency list.

```
ReverseGreedyMST(G):
  sort the edges E of G by weight
  for i ← 1 to |E|
      e ← ith heaviest edge in E
      if G \ e is connected
          remove e from G
```

   (a) **[4 pts]** What is the worst-case running time of this algorithm? (Answering this question will require fleshing out a few details.)

   (b) **[6 pts] Prove** that the algorithm transforms G into its minimum spanning tree.

5. SUBSETSUM and PARTITION are two closely related NP-hard problems.

   - SUBSETSUM: Given a set X of integers and an integer k, does X have a subset whose elements sum up to k?
   - PARTITION: Given a set X of integers, can X be partitioned into two subsets whose sums are equal?

   (a) **[2 pts]** Prove that PARTITION and SUBSETSUM are both in NP.

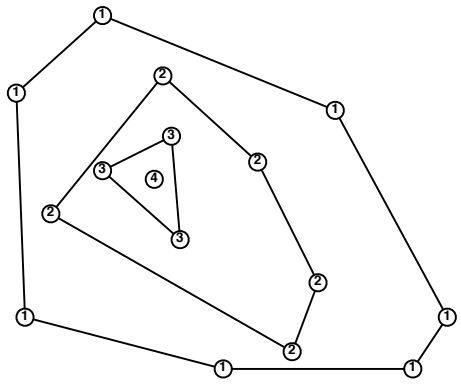
   (b) **[1 pt]** Suppose we knew that SUBSETSUM is NP-hard, and we wanted to prove that PARTITION is NP-hard. Which of the following arguments should we use?

   - Given a set X and an integer k, compute a set Y such that PARTITION(Y) is true if and only if SUBSETSUM(X, k) is true.
   - Given a set X, construct a set Y and an integer k such that PARTITION(X) is true if and only if SUBSETSUM(Y, k) is true.

   (c) **[3 pts]** Describe and analyze a polynomial-time reduction from PARTITION to SUBSETSUM. (See part (b).)

   (d) **[4 pts]** Describe and analyze a polynomial-time reduction from SUBSETSUM to PARTITION. (See part (b).)

6. Let P be a set of points in the plane. The *convex layers* of P are defined recursively as follows. If P is empty, it ha no convex layers. Otherwise, the first convex layer is the convex hull of P, and the remaining convex layers are the convex layers of P minus its convex hull.

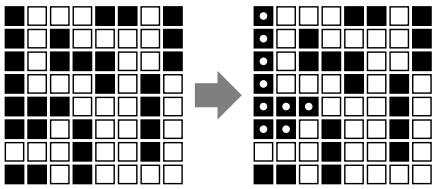


A set of points with 4 convex layers

Describe and analyze an algorithm to compute the number of convex layers of a point set P as quickly as possible. For example, given the points illustrated above, your algorithm would return the number 4.

7. (a) **[4 pts]** Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1..n, 1..n]$.

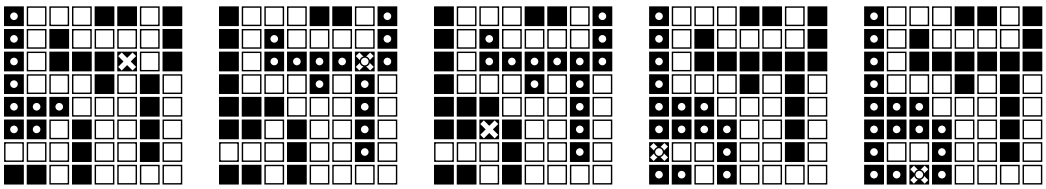
   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.



9

   (b) **[4 pts]** Design and analyze an algorithm BLACKEN(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the BLACKEN algorithm.



9 14 14 16 17

   (c) **[2 pts]** What is the *worst-case* running time of your BLACKEN algorithm?