1. Suppose we are given a set of n rectangular boxes, each specified by their height, width, and depth in centimeters. All three dimensions of each box lie strictly between 10cm and 20cm, and all $3n$ dimensions are distinct. As you might expect, one box can be nested inside another if the first box can be rotated so that is is smaller in every dimension than the second box. Boxes can be nested recursively, but two boxes cannot be nested side-by-side inside a third box. A box is *visible* if it is not nested inside another box.

   Describe and analyze an algorithm to nest the boxes, so that the number of visible boxes is as small as possible.

Solution: We need to ***assign*** as many boxes as possible to a unique larger box. To that end, we define a bipartite graph $G = (L \cup R, E)$, where L contains a vertex for each box, R contains a vertex for each box, and $uv \in E$ if and only if box u can nest inside box v. This graph has $2n$ vertices and $O(n^2)$ edges, and we can construct it in $O(n^2)$ time by brute force.

Now we compute a maximum matching in G, using the algorithm described in the notes (which is based on Ford-Fulkerson), in $O(VE) = O(n^3)$ time. Then for each edge uv in the matching, put box u inside box v. A box is visible if and only if the corresponding node in L is *not* adjacent to an edge in the matching. Thus, minimizing the number of visible boxes is equivalent to maximizing the number of edges in the matching.

Overall, the algorithm runs in $\boldsymbol{O(n^3)}$ ***time***. ■

Solution: We build a directed acyclic graph G whose vertices are the boxes, with an edge $u \to v$ if and only if box u fits inside box v. Then we find the smallest number of disjoint paths that cover G, using the algorithm in the textbook; each path corresponds to a nested sequence of boxes. The algorithm runs in $O(VE) = \boldsymbol{O(n^3)}$ ***time***. ■

Rubric: Standard graph-reduction rubric. No correctness proof required. These are not the only correct solutions.

2. Consider the following randomized version of mergesort. The input is an unsorted array $A[1..n]$ of distinct numbers. The MERGE subroutine takes two sorted arrays as input and returns a single sorted array, containing the elements of both input arrays, in linear time.

```
RandomizedMergeSort(A[1..n]):
  if n ≤ 1
      return A
  ℓ ← 0;  r ← 0
  for i ← 1 to n
      with probability 1/2
          ℓ ← ℓ + 1
          L[ℓ] ← A[i]
      else
          r ← r + 1
          R[r] ← A[i]
  L ← RandomizedMergeSort(L[1..ℓ])
  R ← RandomizedMergeSort(R[1..r])
  return Merge(L, R)
```

(a) Fix two arbitrary indices $i \neq j$. What is the probability that $A[i]$ and $A[j]$ appear in the same recursive subproblem (either L or R)?

Solution: $1/2$ ■

Rubric: 1 point.

(b) What is the probability that $A[i]$ and $A[j]$ appear in the same subproblem for more than k levels of recursion?

Solution: $1/2^k$ ■

Rubric: 1 point. No penalty for $1/2^{k-1}$ or $1/2^{k+1}$ (off-by-one errors in level counting), and similarly for later parts.

(c) What is the expected number of *pairs* of items that appear in the same subproblem for more than k levels of recursion?

Solution: $\binom{n}{2}/2^k$ ■

Rubric: 2 points.

(d) Give an upper bound on the probability that ***at least one*** pair of items appear in the same subproblem for more than k levels of recursion. Equivalently, upper bound the probability that the recursion tree of RandomizedMergeSort has depth greater than k.

Solution: $\binom{n}{2}/2^k$

Let X be the number of pairs that appear together in some level-k subproblem. Then $\Pr[X \geq 1] \leq \mathrm{E}[X]/1 = \binom{n}{2}/2^k$ by Markov's inequality. ■

Solution: $\binom{n}{2}/2^k$

Fix k. Let X_{ij} indicate whether $A[i]$ and $A[j]$ appear in the same level-k subproblem, let $X = \sum_{i<j} X_{ij}$ be the number of pairs that appear together in some level-k subproblem, and finally let $\mu = \mathrm{E}[X] = \binom{n}{2}/2^k$. The variables X_{ij} are pairwise indpendent, so we can apply Chebychev's inequality to bound $\Pr[X \geq 1]$ as follows:

$$\begin{aligned} \Pr[X \geq 1] &= \Pr[X \geq (1+\delta)\mu] && \text{where } \delta = \frac{1}{\mu} - 1 \\ &< \frac{1}{\delta^2 \mu} && \text{by Chebyshev} \\ &= \frac{\mu}{(1-\mu)^2} \end{aligned}$$

This upper bound is *slightly larger* than μ, but not enough to matter. ■

Solution (partial credit): $1-(1-1/2^k)^{\binom{n}{2}}$

This would be the exact answer if pairs were fully independent, but they are not. If $A[1]$ and $A[2]$ appear in the same level-k subproblem, and $A[1]$ and $A[3]$ also appear in the same level-k subproblem, then $A[2]$ and $A[3]$ appear in the same level-k subproblem with probability 1, not with probability 2^{-k}.

On the other hand, the World's Most Useful Approximation ($1 + x \approx e^x$ when $x \approx 0$) implies $1-(1-1/2^k)^{\binom{n}{2}} \approx 1 - e^{-\binom{n}{2}/2^k} \approx 1-(1-\binom{n}{2}/2^k) = \binom{n}{2}/2^k$, so this is actually pretty close, at least when $k \gg 2\lg n$. ■

Rubric: 2 points.

- 1 point for expression equivalent to $\binom{n}{2}/2^k$ after significant simplification

(e) For what value of k is the probability in part (d) at most $1/n$?

Solution: $k = 3 \lg n$

If $k = 3 \lg n$, then $\Pr[X \geq 1] \leq \binom{n}{2}/2^k = \binom{n}{2}/n^3 = n(n-1)/2n^3 < 1/n$. ■

Rubric: 2 points = 1 for "log n" + 1 for leading constant ≥ 3.

- 1½ for expression equivalent to $3 \lg n \pm O(1)$ after significant simplification (for example: $k = \lg(n\binom{n}{2})$.)
- 1 point for $\Theta(\log n)$. The leading constant must be at least 3.

(f) Prove that RandomizedMergeSort runs in $O(n \log n)$ time with probability at least $1 - 1/n$.

Solution: The total work at each level of the recursion tree is $O(n)$. We just proved that with probability at least $1 - 1/n$, the recursion tree has depth at most $3 \lg n$. We conclude that with probability at least $1 - 1/n$, RandomizedMergeSort runs in at most $3 \lg n \cdot O(n) = O(n \log n)$ time. ■

Rubric: 2 points. This is not the only correct proof.

3. Suppose we are given a set R of n *red* points, a set G of n *green* points, and a set B of n *blue* points; each point is given as a pair (x, y) of real numbers. We call these sets *separable* if there is a pair of parallel lines $y = ax + b$ and $y = ax + b'$ such that (1) all red points are below both lines, (2) all blue points are above both lines, and (3) all green points are between the lines.

   (a) Describe a linear program that is feasible if and only if the point sets G, B, R are separable.

   Solution: Our linear program has three variables a, b, and b', representing the pair of lines $y = ax + b$ and $y = ax + b'$. We only care about feasibility, so the objective function doesn't matter.

maximize	whatever	
subject to	$ax + b \geq y$	for each red point (x, y)
	$ax + b' \geq y$	for each red point (x, y)
	$ax + b \geq y$	for each green point (x, y)
	$ax + b' \leq y$	for each green point (x, y)
	$ax + b \leq y$	for each blue point (x, y)
	$ax + b' \leq y$	for each blue point (x, y)

   The green constraints imply $b \geq b'$, which means the first set of red constraints and the last set of blue constraints are redundant. ■

   (b) Describe a linear program whose solution describes a pair of parallel lines that separates G, B, R whose *vertical* distance is as small as possible. (Here you can assume that G, B, R are separable.)

   Solution: We use exactly the same constraints as part (a), with the objective

   $$\text{minimize } b - b'$$

   ■

4. Let $G = (V, E)$ be an arbitrary dag with a unique source s and a unique sink t. Suppose we compute a random walk from s to t, where at each node v (except t), we choose an outgoing edge $v \to w$ uniformly at random to determine the successor of v.

   (a) Describe and analyze an algorithm to compute, for every vertex v, the probability that the random walk visits v.

Solution: For any vertex v, let $Prob(v)$ denote the probability that our random walk visits v. This function satisfies the following recurrence, where $outdeg(u)$ denotes the number of edges leaving u.

$$Prob(v) = \begin{cases} 1 & \text{if } v = s \\ \displaystyle\sum_{u \to v} Prob(u)/outdeg(u) & \text{otherwise} \end{cases}$$

We need to compute this function for every vertex v.

We can memoize this function into the dag itself, adding a field $v.Prob$ to every node v, and we can evaluate the function for all v in *forward* topological order, in $\boldsymbol{O(V+E)}$ ***time***.

```
RandomWalkProbabilities(G):
  topologically sort G
  for all vertices v in topological order
      if v = s
          v.Prob ← 1
      else
          v.Prob ← 0
          for all edges u→v
              v.Prob ← v.Prob + u.Prob/u.outdeg
```

Equivalently, as usual, we can evaluate this function for all v by by performing a depth-first search of the reversed graph G^R starting at t.

```
RandomWalkProbabilities(G):
  for all vertices v
      v.Prob ← −1        ⟨⟨Sentinel value⟩⟩
  t.Prob ← RecRWP(t)
```

```
RecRWP(v):
  if v.Prob < 0
      if v = s
          v.Prob ← 1
      else
          v.Prob ← 0
          for all edges u→v
              v.Prob ← v.Prob + RecRWP(u)/u.outdeg
  return v.Prob
```

■

Rubric: 5 points: standard dynamic programming rubric.

(b) Describe and analyze an algorithm to compute the expected number of edges in the random walk.

Solution: Linearity of expectation implies that the expected number of *vertices* in the random walk is exactly $\sum_v Prob(v)$. So the following algorithm computes the expected number of *edges* in the random walk in $\boldsymbol{O(V+E)}$ ***time***.

```
RandomWalkExpectedLength(G):
  RandomWalkProbabilities(G)
  ℓ ← 0
  for all vertices v
      ℓ ← ℓ + v.Prob
  return ℓ − 1
```

■

Rubric: 5 points = 4 for algorithm + 1 for running time. Standard DP rubric for solutions that use DP directly.

5. You are managing a company with a large number of project teams. Each project team contains exactly three people. A single employee could belong to any number of project teams, but all of the project teams are different.

   For each employee x, you know a positive real number $\mathit{trouble}(x)$, which roughly estimates the probability that x will cause trouble. Your task is to choose a subset X of the employees that includes at least one member of each project team, such that $\sum_{x \in X} \mathit{trouble}(x)$ is as small as possible.

   (a) Write a linear programming relaxation for this problem.

   Solution: Identify the employees with the integers 1 through n. The linear program has a variable x_i, which intuitively indicates whether employee i is invited to the meeting.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} x_i \cdot \mathit{trouble}_i \\ \text{subject to} \quad & x_i + x_j + x_k \geq 1 \qquad \text{for each team } \{i,j,k\} \\ & x_i \geq 0 \qquad \text{for each employee } i \end{aligned}$$

   (To formulate the problem exactly using an integer linear program, replace each constraint $x_i \geq 0$ with $x_i \in \{0,1\}$.) ■

   (b) Describe an efficient **3-*approximation*** algorithm for this problem. *[Hint: Round the solution to your linear program from part (a).]*

   Solution: Let OPT denote the value of the optimal *integer* solution to our linear program. Let $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$ be the optimal *fractional* solution to the linear program in part (a), and let $OPT^* = \sum_i x_i^* \cdot \mathit{trouble}_i$ denote its objective value. We immediately have $OPT^* \leq OPT$.

   Define a new vector $x' = (x_1', x_2', \ldots, x_n')$ by setting

$$x_i' = \begin{cases} 1 & \text{if } x_i^* \geq 1/3 \\ 0 & \text{otherwise} \end{cases}$$

   and let $OPT' = \sum_i x_i' \cdot \mathit{trouble}_i$. We invite employee i to the meeting if and only if $x_i' = 1$.

   For each team $\{i,j,k\}$, the constraint $x_i^* + x_j^* + x_k^* \geq 1$ implies $\max\{x_i^*, x_j^*, x_k^*\} \geq 1/3$, so at least one of the variables x_i', x_j', x_k' is equal to 1. Thus, at least one member of each team is invited to the meeting.

   Finally, we have $x_i' \leq 3x_i^*$ for each employee i, which implies $OPT' \leq 3 \cdot OPT^* \leq 3 \cdot OPT$. ■

   A 3-approximation algorithm that is correct when $\mathit{trouble}(x) = 1$ for every employee x is worth half credit.

Solution: In this special case, we are trying to minimize the number of invited employees. The input to the following algorithm is the set T of all project teams; the output is the set of employees to invite.

```
DumbInvite(T):
  X ← ∅
  for all teams {i, j, k} ∈ T
      if i ∉ X and j ∉ X and k ∉ X
          X ← X ∪ {i, j, k}                (*)
  return X
```

Let OPT denote the smallest set of employees that we can invite, and suppose DumbInvite executes line $(*)$ exactly K times. OPT must contain at least one employee from each team $\{i, j, k\}$ that invokes line $(*)$. Thus, $|OPT| \geq K$. On the other hand, $|X| = 3K$. We conclude that $|X| \leq 3|OPT|$. ■

6. Suppose we need to distribute a message to all the nodes in a given binary tree. Initially, only the root node knows the message. In a single round, each node that knows the message is allowed (but not required) to forward it to at most one of its children. Describe and analyze an algorithm to compute the minimum number of rounds required for the message to be delivered to all nodes in the tree.

Solution: Let $MinR(v)$ denote the minimum number of rounds need to spread a message from v to all its descendants. We need to compute $MinR(root)$. This function obeys the following recurrence:

$$MinR(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + MinR(w) & \text{if } w \text{ is only child of } v \\ \min \begin{Bmatrix} \max\{1 + MinR(v.left),\ 2 + MinR(v.right)\} \\ \max\{2 + MinR(v.left),\ 1 + MinR(v.right)\} \end{Bmatrix} & \text{otherwise} \end{cases}$$

We can memoize this function into the tree itself. Since the value at every node depends only on the values at its children, we can evaluate this function at all nodes by a postorder traversal of the tree. The resulting algorithm runs in $O(n)$ *time*. ■

Rubric: Standard dynamic programming rubric.