1. Suppose we are given two arrays $C[1..n]$ and $R[1..n]$ of positive integers. An $n \times n$ matrix of 0s and 1s *agrees with R and C* if, for every index i, the ith row contains $R[i]$ 1s, and the ith column contains $C[i]$ 1s. Describe and analyze an algorithm that either constructs a matrix that agrees with R and C, or correctly reports that no such matrix exists.

Solution: Let $G[1..n][1..n]$ denote the output matrix. (G stands for 'greedy'.) If $n = 0$, the algorithm returns the empty matrix G; so assume from now on that $n > 0$. The algorithm has several stages.

1. Sort the arrays $R[1..n]$ and $C[1..n]$ into non-decreasing order, while maintaining the original index of each entry $R[i]$ and $C[j]$ (for step 5).
2. If $R[n] < 0$ or $R[n] > n$, return NONE. Otherwise, fill the nth row of the output matrix with $n - R[n]$ 0's followed by $R[n]$ 1's.
3. If $C[n] < 0$ or $C[n] > n$, return NONE. Otherwise, fill the nth column of the output matrix with $n - C[n]$ 0's followed by $C[n]$ 1s.
4. Recursively fill the subarray $G[1..n-1][1..n-1]$ with a matrix that agrees with $R[1..n-1]$ and $C[1..n-1]$. If the recursive call returns NONE, return NONE.
5. Permute the rows and columns of A to reflect the original permutation of R and C.
6. Return $G[1..n][1..n]$.

The running time of the algorithm is dominated by the recursive call in step 4 and the $O(n^2)$-time permutation of the output matrix in step 5. Thus, the running time satisfies the recurrence $T(n) = T(n-1) + O(n^2)$, which means the algorithm runs in ***$O(n^3)$ time***.[1]

Correctness. It is clear that if the algorithm returns a matrix (and not NONE), that matrix agrees with R and C. It remains to prove that if there is a matrix that agrees with R and C, the algorithm returns such a matrix, and not NONE.

Assume without loss of generality that the arrays R and C are already sorted from smallest to largest. We claim that if any matrix that agrees with R and C, then there is such a matrix, such that all the 1's are to the right of all the 0's in the last row, and all the 1's are below all the 0's in the last column. In other words, if there is a valid matrix, there is a valid matrix that agrees with the output of our greedy algorithm in the last row and column.

Let A be an arbitrary matrix that agrees with R and C. There are three cases to consider.

- If all the 1's are to the right of all the 0's in the last row of A, and all the 1's are below all the 0's in the last column of A, we are done.
- Suppose $A[n][i] = 1$ and $A[n][j] = 0$ for some indices $i < j$. Because the array C is sorted, we must have $C[i] \le C[j]$. Thus, there must be a row index $r < n$ such that $A[r][i] = 0$ and $A[r][j] = 1$. If we modify the matrix A by setting $A[n][i] \leftarrow 1$, $A[n][j] \leftarrow 0$, $A[r][i] \leftarrow 1$, and $A[r][j] \leftarrow 1$, the resulting matrix still agrees with R and C, but now has fewer 1's to the left of 0's in the last row. Thus, by induction, we can modify A so that all 1's are the right of all 0's in the last row.
- Suppose all 1's are to the right of all 0's in the last row of A, but $A[i][n] = 1$ and $A[j][n] = 0$ for some indices $i < j < n$. (If $A[n][n] = 0$, then the entire matrix must be filled with zeros!) By a symmetric inductive argument, we can modify A so that it still agrees with R and C, but all 1s are below all 0's in the last column. None of the swaps in this case change the last row, so all 1's are still to the right of all 0's in the last row.

[1]With more effort, the running time can be reduced to $O(n^2)$.

Finally, we finish the proof by induction on n. The algorithm is trivially correct when $n = 0$, so suppose $n > 0$, and suppose there is a matrix that agrees with R and C. After sorting R and C, there is still a a matrix that agrees with R and C. The previous exchange argument implies that there is a matrix A that agrees with G in the last row and column. The first $n-1$ rows and $n-1$ columns of A agree with $R[1\,..\,n-1]$ and $C[1\,..\,n-1]$. Thus, the inductive hypothesis implies that the recursive call returns a matrix, and not NONE. We conclude that the algorithm actually returns a matrix, and not NONE, as required. ■

Rubric: max 10 points = 4 for correct algorithm + 3 for exchange argument + 2 for final induction + 1 for running time. No credit for a solution that does not include an attempted proof. (Thus, a correct algorithm with correct time analysis but a faulty proof of correctness is worth at most 5 points.) **+3 extra credit** for a correct algorithm that runs in $O(n^2)$ time; -3 for any slower correct algorithm. This is not the only correct algorithm, nor the only valid proof of correctness. This proof is more detailed than necessary for full credit.

2. Suppose we have n skiers with heights given in an array $P[1..n]$, and n skis with heights given in an array $S[1..n]$. Describe an efficient algorithm to assign a ski to each skier, so that the average difference between the height of a skier and her assigned ski is as small as possible. The algorithm should compute a permutation σ such that the expression

$$\frac{1}{n}\sum_{i=1}^{n}\bigl|P[i]-S[\sigma(i)]\bigr|$$

is as small as possible.

Solution: We use the following greedy algorithm: For each index i from 1 to n, assign the ith tallest skis to the ith tallest skier. This algorithm is easy to implement in $\boldsymbol{O(n\log n)}$ ***time***; after sorting the skis and the skiers, the actual assignment takes only $O(n)$ time.

Proof of correctness. Without loss of generality, assume that the arrays P and S are already sorted downward. For any permutation σ, let $A(\sigma)=\sum_{i=1}^{n}|P[i]-S[\sigma(i)]|$. We need to prove that the identity permutation $\sigma(i)=i$ minimizes $A(\sigma)$. (We can clearly ignore the $1/n$ factor.)

First we claim that there is an optimal permutation σ such that $\sigma(1)=1$; that is, there is an optimal assignment where the tallest skier gets the tallest skis. Our claim is trivially true when $n\le 1$, so assume $n\ge 2$. Let σ be a arbitrary permutation such that $\sigma(1)=k\ne 1$, and let ℓ be the index such that $\sigma(\ell)=1$. Let σ' be the unique permutation that agrees with σ except that $\sigma'(1)=1$ and $\sigma'(\ell)=k$. Chasing definitions and canceling common terms, we find that

$$\Delta := A(\sigma)-A(\sigma') \;=\; |P[1]-S[k]|+|P[\ell]-S[1]|-|P[1]-S[1]|-|P[\ell]-S[k]|$$

We now show by exhaustive case analysis that $\Delta\ge 0$, which implies that $A(\sigma)\ge A(\sigma')$. Because $P[1]\ge P[\ell]$ and $S[1]\ge S[k]$ by assumption, there are exactly six permutations of $P[1]$, $S[1]$, $P[\ell]$, and $S[k]$ to consider.

- If $S[1]\ge S[k]\ge P[1]\ge P[\ell]$, then
$$\begin{aligned}\Delta &= \bigl(S[k]-P[1]\bigr)+\bigl(S[1]-P[\ell]\bigr)-\bigl(S[1]-P[1]\bigr)-\bigl(S[k]-P[\ell]\bigr)\\ &=0\end{aligned}$$

- If $S[1]\ge P[1]\ge S[k]\ge P[\ell]$, then
$$\begin{aligned}\Delta &= \bigl(P[1]-S[k]\bigr)+\bigl(S[1]-P[\ell]\bigr)-\bigl(S[1]-P[1]\bigr)-\bigl(S[k]-P[\ell]\bigr)\\ &=2(P[1]-S[k])\ge 0\end{aligned}$$

- If $S[1]\ge P[1]\ge P[\ell]\ge S[k]$, then
$$\begin{aligned}\Delta &= \bigl(P[1]-S[k]\bigr)+\bigl(S[1]-P[\ell]\bigr)-\bigl(S[1]-P[1]\bigr)-\bigl(P[\ell]-S[k]\bigr)\\ &=2(P[1]-P[\ell])\ge 0\end{aligned}$$

- If $P[1]\ge P[\ell]\ge S[1]\ge S[k]$, then
$$\begin{aligned}\Delta &= \bigl(P[1]-S[k]\bigr)+\bigl(P[\ell]-S[1]\bigr)-\bigl(P[1]-A[1]\bigr)-\bigl(P[\ell]-S[k]\bigr)\\ &=0\end{aligned}$$

- If $P[1]\ge S[1]\ge P[\ell]\ge S[k]$, then
$$\begin{aligned}\Delta &= \bigl(P[1]-S[k]\bigr)+\bigl(S[1]-P[\ell]\bigr)-\bigl(P[1]-S[1]\bigr)-\bigl(P[\ell]-S[k]\bigr)\\ &=2(S[1]-P[\ell])\ge 0\end{aligned}$$

- Finally, if $P[1] \geq S[1] \geq S[k] \geq P[\ell]$, then

$$\begin{aligned}\Delta &= \big(P[1]-S[k]\big) + \big(S[1]-P[\ell]\big) - \big(P[1]-S[1]\big) - \big(S[k]-P[\ell]\big)\\ &= 2(S[1]-S[k]) \geq 0\end{aligned}$$

In all cases, we conclude that $A(\sigma) \geq A(\sigma')$. In particular, if σ is an optimal permutation, then so is σ'. This completes the proof of our claim.

We now prove by induction that the identity permutation $\sigma(i) = i$ minimizes $A(\sigma)$, starting with the vacuous base case $n = 0$. For any $n \geq 1$, we can write $A(\sigma) = |P[1] - S[1]| + A'(\sigma)$, where $A'(\sigma) = \sum_{i=2}^{n} |P[i] - S[\sigma(i)]|$. The preceding exchange argument implies that there is an optimal permutation with $\sigma(1) = 1$, and the inductive hypothesis implies that the identity permutation over the set $\{2, 3, \ldots, n\}$ minimizes $A'(\sigma)$. It follows that the identity permutation is optimal, as claimed. ■

Rubric: max 10 points = 4 for correct algorithm + 3 for exchange argument + 2 for final induction + 1 for running time. No credit for a solution that does not include an attempted proof. (Thus, a correct algorithm with correct time analysis but a faulty proof of correctness is worth at most 5 points.) −3 for any slower correct algorithm; scale partial credit. This is not the only correct algorithm, nor the only valid proof of correctness.

3. Alice wants to throw a party and she is trying to decide who to invite. She has n people to choose from, and she knows which pairs of these people know each other. She wants to pick as many people as possible, subject to two constraints:

   - For each guest, there should be at least five other guests that they already know.
   - For each guest, there should be at least five other guests that they *don't* already know.

   Describe and analyze an algorithm that computes the largest possible number of guests Alice can invite, given a list of n people and the list of pairs who know each other.

   Solution: We model the input as an undirected graph G, which has a vertex for each person and an edge (u, v) for each pair u and v that know each other. Let $\deg(v)$ denote the degree of vertex v in G. Our algorithm returns the error code NONE if there is no valid guest list.

```
GUESTLIST(G):
  $n \leftarrow |V(G)|$

  $z \leftarrow$ vertex in $G$ with maximum degree
  if $\deg(z) < 5$
      return NONE
  else if $\deg(z) > n - 6$
      return GUESTLIST($G - z$)

  $a \leftarrow$ vertex in $G$ with minimum degree
  if $\deg(a) < 5$
      return GUESTLIST($G - a$)

  return $n$
```

   We can convert the lists of n people and m acquaintance pairs into an adjacency list representation of the graph G in $O(n + m)$ time in a preprocessing phase. Using this representation, it is easy to find the vertices with minimum and maximum degree in $O(m + n)$ time, and to delete any vertex in $O(m+n)$ time. Thus, each call to GUESTLIST requires at most $O(m+n)$ time, not counting recursion. Since GUESTLIST makes at most n recursive calls before halting, the total running time is $\boldsymbol{O(n(n + m))}$.

   Proof of correctness. We prove that GUESTLIST(G) returns the largest possible number of guests by induction on n. Consider each line of the algorithm, in order

   If the maximum-degree vertex z has degree less than 5, then *every* vertex has degree less than 5, so there is no valid guest list. The algorithm correctly returns NONE in this case. In particular, if $n \leq 5$, the algorithm correctly returns NONE.

   If $\deg(z) > n - 6$, then z knows too many people to be invited. By the inductive hypothesis, GUESTLIST$(G - z)$ returns the maximum number of people *other* than z that can be invited, or *None* if there is no valid guest list that excludes z. Thus, the algorithm is correct in this case.

   If the minimum degree vertex a has degree less than 5, then a knows too few people to be invited. By the inductive hypothesis, GUESTLIST$(G - a)$ returns the maximum number of people *other* than a that can be invited, or *None* if there is no valid guest list that excludes a. Thus, the algorithm is correct in this case.

   Finally, if $\deg(z) \leq n - 5$ and $\deg(a) \geq 5$, then for every vertex v we have $5 \leq \deg(v) \leq n - 5$. Thus, we can legally invite all n guests, and this is clearly optimal. (This is the base case of our inductive proof.) ■

Rubric: max 10 points = 4 for correct algorithm + 4 for proof + 2 for running time. -1 for reporting the running time as $O(n^3)$. No credit for a solution that does not include an attempted proof. (Thus, a correct algorithm with correct time analysis but a faulty proof of correctness is worth at most 6 points.)

4. Consider the following heuristic for constructing a vertex cover of a connected graph G: return the set of non-leaf nodes in any depth-first spanning tree of G.

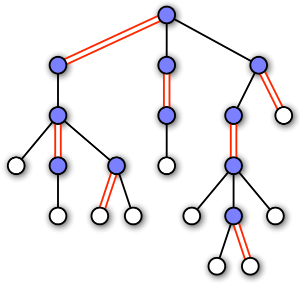
   (a) Prove that this heuristic returns a vertex cover of G.

   Solution: Let T be a depth-first spanning tree of G, and let N be the set of nodes in G that are not leaves in T. A vertex v is a leaf in T if and only if the depth-first search visits every neighbor of v before it visits v. Thus, two leaves in T cannot be adjacent in G. Conversely, for any edge in G, at least one of its endpoints must be in N. We conclude that N is a vertex cover of G. ■

   Rubric: 3 points max. −1½ points for one minor error.

   (b) Prove that this heuristic returns a 2-approximation to the minimum vertex cover of G.

   Solution: Let T be a depth-first spanning tree of G. We construct a *matching* M in T—a subgraph of T in which every node has degree 1—using the following recursive algorithm. Perform a preorder traversal of T, visiting the children of each node in arbitrary order; initially M is empty and every node is unmarked. Whenever the traversal visits an unmarked node v whose parent u is also unmarked, add edge uv to M and mark both u and v.



The number of edges in the matching is at least half the number of non-leaf nodes in the tree.

   By construction, every non-leaf node in T is marked and therefore adjacent to exactly one edge in M. (Some leaves are also marked, but that only helps us.) Thus, if T has k non-leaf nodes, then M contains at least $k/2$ edges. Every vertex cover contains at least one node incident to each edge in M; thus, the minimum vertex cover has at least $k/2$ nodes. We conclude that our algorithm computes a vertex cover that is at most twice as large as the minimum vertex cover. ■

   Rubric: 4 points max. −2 points for one minor error.

   (c) Describe an infinite family of graphs for which this heuristic returns a vertex cover of size $2 \cdot OPT$.

   Solution: Suppose the input graph G is a path of length $2k$; let $v_1, v_2, \ldots, v_{2k+1}$ be the vertices of the path. If we start our depth-first search at v_1, then v_{2k+1} is the only leaf in the depth-first spanning tree, so we get a vertex cover of size $2k$. On the other hand, the even-index vertices $v_2, v_4, \ldots, v_{2k}$ define a vertex cover of size k. ■

   Rubric: 3 points max = 2 for family of graphs + 1 for proof.

5. Suppose we want to route a set of N calls on a telecommunications network that consist of a cycle on n nodes, indexed in order from 0 to $n-1$. Each call has a source node and a destination node, and can be routed either clockwise or counterclockwise around the cycle. Our goal is to route the calls so as to minimize the overall load on the network. The load L_i on any edge $(i, (i+1) \bmod n)$ is the number of calls routed through that edge, and the overall load is $\max_i L_i$. Describe and analyze an efficient 2-approximation algorithm for this problem.

Solution: We use a straightforward greedy algorithm: Route each call along the shorter of the two paths connecting its source and destination nodes. The algorithm clearly runs in ***O(N) time***.

Let Alg be the routing computed by this greedy algorithm, and let Opt be any optimal routing. For any integer i, let e_i denote the edge $(i \bmod n, (i+1) \bmod n)$, and let $\text{Alg}(i)$ and $\text{Opt}(i)$ denote the loads on each e_i in routings Alg and Opt, respectively.

In the greedy routing Alg, each call uses at most $\lfloor n/2 \rfloor$ edges. In particular, no call in Alg uses both an edge e_i and the antipodal edge $e_{i+\lfloor n/2 \rfloor}$. Thus, any call that uses edge e_i in Alg must use either e_i or the diametrically opposite edge $e_{i+\lfloor n/2 \rfloor}$ in *any* other routing, including Opt. It follows that for every index i, we have

$$\begin{aligned}\text{Alg}(i) &\le \text{Opt}(i) + \text{Opt}(i + \lfloor n/2 \rfloor) \\ &\le 2 \cdot \max\left\{\text{Opt}(i),\ \text{Opt}(i + \lfloor n/2 \rfloor)\right\}.\end{aligned}$$

Therefore,

$$\begin{aligned}\max_i \text{Alg}(i) &\le \max_i \left(2 \cdot \max\left\{\text{Opt}(i),\ \text{Opt}(i + \lfloor n/2 \rfloor)\right\}\right) \\ &\le 2 \cdot \max_i \text{Opt}(i).\end{aligned}$$

We conclude that our algorithm computes a 2-approximation of the optimum load, as required. ■

Rubric: 10 points max = 4 points for algorithm + 2 points for time analysis + 4 points for approximation ratio analysis. This is not the only correct algorithm, nor the only correct proof. Max 7 points for an algorithm that runs in super-linear time, or has a constant approximation ratio larger than 2.