(a) Prove that every tournament $G = (V, E)$ contains a Hamiltonian path.

**Solution:** For simplicity, we'll slightly abuse the definition of path to include a length 0 sequence of edges from a vertex $u$ to itself.

Let $v$ be an arbitrary vertex of $V$. Per the hint, let $V^-$ denote the predecessors of $V$, and let $V^+$ denote the successors of $v$. Let $G^- = (V^-, E^-)$ denote the induced subgraph of $V^-$ which contains every edge of $G$ between a pair of vertices in $V^-$. Let $G^+ = (V^+, E^+)$ denote the induced subgraph of $V^+$.

Both $G^-$ and $G^+$ are tournaments. Consider the following path $P$ in $G$. Suppose $|V^-| \geq 1$. In this case, $G^-$ contains a Hamiltonian path $P^-$ by induction on the number of vertices. Let $u \in V^-$ be the last vertex of $P^-$. By definition of $V^-$, vertex $u$ is an in-neighbor of $v$. We have $P$ begin with $P^-$ followed by edge $u \rightarrow v$. If $|V^-| = 0$, path $P$ begins with $v$ instead. Now, suppose $|V^+| \geq 1$. In this case, $G^+$ contains a Hamiltonian path $P^+$ by induction on the number of vertices. Let $w \in V^+$ be the first vertex of $P^+$. By definition of $V^+$, vertex $w$ is an out-neighbor of $v$. We have $P$ end with $v \rightarrow w$ followed by $P^+$. If $|V^+| = 0$, path $P$ ends with $v$ instead. Our path $P$ includes every vertex of $G$, so it is Hamiltonian.

**Rubric:** 4 points total. —1 point if their argument excludes some base cases.

(b) Describe and analyze an algorithm to compute a Hamiltonian path in a tournament $G$. Any output that reasonably describes the path or the order of the vertices along the path is fine.

**Solution:** The inductive proof in part (a) suggests a recursive strategy to find a Hamiltonian path, but it is still not clear how to implement it efficiently. The most natural approach is to explicitly create the tournaments $G^-$ and $G^+$, but that leads to spending $O(V^2)$ time outside recursive calls. If the calls are very unbalanced, the total running time could go to $O(V^3)$.

Instead, we'll pass along a boolean array of length $V$ that indicates which vertices are present in a recursive subproblem. We can build the arrays for recursive calls in $O(V)$ time by looking over the adjacency list for our arbitrarily selected vertex $v$. Doing so will let us spend only $O(V)$ work outside recursive calls. For our output, we'll print a list of vertices in the order they appear in $G$'s Hamiltonian path. Assume the vertices are numbered arbitrarily from 1 to $|V|$. The procedure `RecPath(verts[1..|V|])` takes as input a boolean array `vertex[1..|V|]` where `verts[v]` is `TRUE` if and only if vertex $v$ appears in the subset of vertices for which we're trying to find a Hamiltonian path. We'll assume at least one entry is `TRUE` in each recursive call. We initially pass an array with every entry equal to `TRUE`. 


RecPath(verts[1 .. |V|]):
    v ← 1  ⟨Pick vertex v from the given subset.⟩
    while not verts[v]
        v ← v + 1
        for w ← 1 to |V|
            mcount ← 0
            minus[w] ← FALSE
            pcount ← 0
            plus[w] ← FALSE
            for each edge v→w ⟨Build V⁺.⟩
                if verts[w]
                    pcount ← pcount + 1
                    plus[w] ← TRUE
            for w ← 1 to |V| ⟨Build V⁻.⟩
                if w ≠ v and verts[w] and not plus[w]
                    mcount ← mcount + 1
                    minus[w] ← TRUE
                if mcount ≥ 1
                    RecPaths(minus[1 .. |V|])
            print v
            if pcount ≥ 1
                RecPaths(plus[1 .. |V|])

Correctness follows immediately from part (a). Each recursive subproblem does \(O(V)\) work outside its recursive calls. Each subproblem also involves a different choice of the vertex \(v\), so there are \(|V|\) subproblems. The algorithm runs in \(O(V^2)\) time. ■

Rubric: 6 points total: 4 points for the algorithm; 2 points for running time analysis.
−1 point if correctness is not obvious but no argument is given. A correct \(O(V^3)\) time algorithm is worth 4 points.
Describe and analyze an efficient algorithm that correctly determines if the puzzle has a solution.

**Solution:** Per the hint, we’ll build a directed configuration graph $G = (V, E)$ for this puzzle. We add a vertex $(i_r, j_r, i_b, j_b)$ for each choice of four integer coordinates between 1 and $n$ where $(i_r, j_r) \neq (i_b, j_b)$. Each such vertex represents a configuration where the red token lies at $(i_r, j_r)$ and the blue token lies at $(i_b, j_b)$. We’ll use the convention that position $(i, j)$ is in the $i$th row from the top and the $j$th column from the left. We add an edge $u \rightarrow v$ for each pair vertices $u$ and $v$ where a token can move us from configuration $u$ to configuration $v$ in a single step. Determining each outgoing edge of a vertex is easily done in constant time by computing the new coordinates after each of the possible moves.

We do a breadth-first search from $(1, 1, n, n)$ and return TRUE if and only if $(n, n, 1, 1)$ gets marked. Indeed, a path in $G$ represents a sequence of legal moves from one configuration to another, so there will be a sequence of moves from the initial configuration $(1, 1, n, n)$ to $(n, n, 1, 1)$ if and only if $(n, n, 1, 1)$ is reachable from $(1, 1, n, n)$ in $G$.

The breadth-first search takes $O(V + E)$ time. There are $n^2(n^2 - 1) = O(n^4)$ vertices in $G$. There are at most eight legal moves we can perform from any position (moving one of the two token in each of up to four directions), so the number of edges is $O(n^4)$ as well. The algorithm runs in $O(n^4)$ time.

**Rubric:** 10 points total: 7 points for the algorithm; 3 points for the running time analysis. 8 points total for a correct algorithm that claims $O(n^8)$ running time. −2 points if the algorithm appears correct but it’s not explicitly clear what the vertices and edges represent.