P1) Let $G$ be a tree. Use induction to prove that the number of leaves of $G$ is at least the number of vertices of degree at least 3 in $G$. For example, the following tree has 3 leaves and 1 vertex of degree at least 3, and $3 \geq 1$.

\[
\begin{array}{c}
\circ \\
| \\
\circ \\
| \\
\circ \\
\end{array}
\]

**Solution:** Let $P(n)$ denote the statement “The number of leaves of any tree with $n$ vertices is at least the number of vertices of degree at least 3.”

**Base Case:** $P(1)$ and $P(2)$ holds obviously as there is no vertex of degree at least 3.

**IH:** Suppose $P(n-1)$ holds for some $n \geq 3$.

**IS:** We prove $P(n)$. Let $T$ be an arbitrary tree with $n$ nodes. Suppose that $T$ has $a$ leaves and $b$ nodes of degree at least three. We need to show that $a \geq b$. Since $T$ is a tree it has a leaf, say $x$. Let $T' = T - x$ denote the tree $T$ with the vertex $x$ and all its edges removed. As we prove in class when we remove a leaf from a tree the remaining graph, $T'$, is also tree. Suppose $T'$ has $a'$ leaves and $b'$ nodes of degree at least 3. By IH $a' \geq b'$.

Let $y$ be the unique neighbor of $x$ in $T$. Note that $\deg_{T'}(y) = \deg_T(y) - 1$.

**Case 1:** $\deg_{T}(y) = 2$: Then $a = a'$ because $y$ is a leaf in $T'$ which is no longer leaf in $T$ whereas we get a new leaf, $x$, in $T$. Also in this case, $b = b'$. Therefore, $a = a' \geq b' = b$ as desired.

**Case 2:** $\deg_T(y) \geq 3$: In this case, $a = a' + 1$, because $y$ is not a leaf in $T'$ so we have a new leaf, $x$, in $T$. And, obviously, $b \leq b' + 1$. Therefore, $b \leq b' + 1 \leq a' + 1 = a$ as desired.

Note that $\deg_T(y) = 1$ cannot happen because in such a case $T$ must have two nodes, i.e., $n = 2$.

P2) Let $G$ be a graph with $n$ vertices and at least $n$ edges. Show that $G$ has a cycle.

**Solution:** We prove by contradiction! Suppose $G$ has no cycle. Then,

**Case 1:** $G$ is connected. Then since $G$ has no cycles, $G$ is a tree with $n$ vertices. So it must have $n - 1$ edges. But we said it has $\geq n$. That is a contradiction.

**Case 2:** $G$ is disconnected. Suppose $G$ has $\ell$ connected components with number of vertices $n_1, n_2, \ldots, n_\ell$ and number of edges $m_1, m_2, \ldots, m_\ell$.

**Claim:** For some $i$ we must have $m_i \geq n_i$. **Pf:** For contradiction assume $m_i < n_i$ for all $i$. Summing up these inequalities we get $m = \sum_i m_i < \sum_i n_i = n$. But that contradicts the assumption that $m \geq n$. 

Section 2-1
So let $i$ be one of the indices for which $m_i \geq n_i$. But then the $i$-th component is connected and has no cycles. So similar to Case 1 we get a contradiction.

P3) Given a connected undirected graph $G = (V, E)$ with $n$ vertices and $m$ edges. Design an $O(m + n)$ time algorithm that outputs an edge $e$ of $G$ such that if we delete $e$, $G$ remains connected. If no such edge exists output “Impossible”. For example in the following graph if you delete the red edges the graph remains connected.

Solution: We run the following algorithm: We run BFS from an arbitrary vertex $s$. In the BFS code, when examining neighbors of $u$, say we find an already discovered vertex $x$ that is not the parent of $u$. Then we output the edge $(u, x)$ and we end the algorithm. Otherwise, if all edges have been examined without finding such a vertex, we output “Impossible”.

Correctness: Let $T$ be the BFS tree. Since $G$ is connected, all vertices are reachable from $s$; so $T$ has $n$ vertices and $n - 1$ edges.

We consider the following cases: If $G$ has no extra edges other than edges of $T$, i.e., $G$ has $n - 1$ edges. Then if we remove any edge of $G$ the remaining graph is disconnected. To see this, notice that $G - e$ has no cycles (since $G$ has no cycles) and if in addition it is connected then it must have $n - 1$ edges (not $n - 2$). In such a case since every edge of $G$ is in $T$ our code never finds an already discovered vertex and it outputs “Impossible”.

Otherwise, suppose $G$ has extra edges in addition to those contained in $T$. Then, the algorithm will eventually output some edge $e = (u, x)$ that is not in $T$ while inspecting vertex $u$. This means that $x$ was previously already marked as discovered, and therefore there is a path in the BFS tree $T$ that connects $x$ to $u$. Together with the edge $e = (u, x)$, this forms a cycle.

Running time: We are just adding one line to the BFS code, so the algorithm runs in the BFS time, i.e., $O(m + n)$.

We write the psuedo-code below, although the above description is already enough:
**Function BFS(s)**

**Initialize:** mark all vertices “undiscovered”
mark s “discovered”, set $P[s] = s$
queue = { s }
while queue not empty do
  $u = \text{remove\_first}(\text{queue})$
  for each edge $\{u, x\}$ do
    if $x$ is “undiscovered” then
      mark $x$ “discovered”
      Set Parent of $x$ to be $u$, $P[x] = u$. append $x$ on queue
    end
  else
    If $P[u] \neq x$, output $\{u, x\}$ and end the algorithm
  end
end
mark $u$ “fully-explored”
end
output “Impossible”

**Algorithm 1:** Algorithm for P3