Problem 1

Here we have two cases.

The first is if $d_i \geq i-1$ for all i. Then we have that $k = 1 + \max_i \min\{d_i, i-1\} = 1 + n - 1 = n$. In this case, we can just cover each vertex a unique color. So clearly the graph is $k = n$ colorable.

The second case is if we have $d_i \geq i-1$ for some i and $d_i < i-1$ for other i. In this case, there is a crossover point $i = x$ such that when $i = x-1$, $d_{x-1} \geq x-2$ and when $i = x$, $d_x < x-1$. Notice that here,

$$i < x-1 \implies \min\{d_i, i-1\} = i-1$$

$$i > x \implies \min\{d_i, i-1\} = d_i$$

Given the nature of i and d_i (one increases monotonically as the other decreases monotonically), we then have that

$$\text{for } i < x-1,\ \max_i \min\{d_i, i-1\} = x-3$$

$$\text{for } i > x,\ \max_i \min\{d_i, i-1\} = d_{x+1}$$

Then we have that

$$\text{for } i = x-1,\ \max_i \min\{d_i, i-1\} = x-2$$

$$\text{for } i = x,\ \max_i \min\{d_i, i-1\} = d_x$$

We also have that

$$d_x < x-1 \implies d_x \leq x-2$$

So now we know the max of $\min\{d_i, i-1\}$ for all i is $x-2$. So overall, we can conclude that

$$k = 1 + \max_i \min\{d_i, i-1\} = 1 + x - 2 = x - 1$$

Now we must show that our graph is $k = x-1$ colorable. To do this, we can give a method to color G. We can start by coloring $\{v_1, \cdots, v_{x-1}\}$ each a different color. Then, for v_i where $i \geq x$, we have that $d_i < x-1$. So we have that we can color it a color such that it is not connected to any vertices of that same color, because at most v_i is connected to $x-2$ colors (so just color it the remaining color).

So we have covered both cases and proved the desired statement (G is k-colorable) for both.

Problem 2

We can see that adding an edge $e = (u, v)$ to a MST will always create at least one circle. This is because in any spanning tree there is exactly one path between any two vertices u and v. So then we have that adding $e = (u, v)$ will create a circle that includes the path from u to v in the spanning tree and the new edge e.

Now we must show that adding an edge $e = (u, v)$ to a MST will not create more than one circle. This is simple. We have that the only situation in which one can create more than one simple cycle in a graph by adding one edge is if there is already a cycle in the graph. But we have that spanning trees do not have any cycles. So we cannot possibly create more than one simple cycle by adding one edge to an MST. So our proof is complete.

Problem 3

Here is a description of our algorithm:

1. Run regular Prim's on the graph G to get the cost of a MST for G. Let that cost be called C_{MST}.
2. Let E' be the edges adjacent to the vertices V'.
3. Let $G' = (V - V', E - E')$. If G' is not connected, return false, as at least one vertex $v \in V'$ cannot be a leaf in a spanning tree for G.
4. Else, if G' is connected, then run Prim's on $G' = (V - V', E - E')$ to get a MST M'.

5. Let E'' be the subset of E' such that each $e \in E''$ is adjacent to a vertex in V and a vertex in V'. That is, no edge $e \in E''$ connects two vertices in V'. Then add V' and E'' back to G'. Call this new graph G''. Then, for each vertex $v \in V'$, add the edge e_m that connects it to M' with the least weight added. If there is no such edge that connects v to M', return false. Let the the new spanning tree be called M.

6. Compare the cost of M with C_{MST}. If they are equal, M is our desired MST. Else, it is impossible to find a MST of the type we want, so return false.

Now we will explain why our algorithm is correct. First we will explain step 3. We have that if we remove V' and the graph G' is no longer connected, we return false. This because this implies that in any spanning tree, one $v \in V'$ must have at least two edges adjacent to it. If this were not true, then the spanning tree would not be connected (and thus would not actually be a spanning tree). Then we have that we get a spanning tree M' for G', then connect the vertices in V' back to this M' via edges with the least weight (step 5). This gives us a spanning tree because we clearly span all the vertices, and we don't create any cycles because we are not connecting any of the vertices in V' with other vertices in V'. We also have that it enforces that all the vertices in V' are leaves, since for each vertex $v \in V'$ we just add one edge that connects it to M'. If we cannot find an edge that connects it to M', we return false, because this means that not all vertices in V' can be leaves. Then we check if the spanning tree generated by connecting the vertices in V', M, is actually a *minimum* spanning tree. Now we must explain why this algorithm will always find such a MST if it does exist. This is fairly simple; we have that any MST for any graph $G = (V, E)$ such that all vertices in $V' \subset V$ are leaves will be of the same form as the spanning trees we generate with our algorithm. That is, each $v \in V'$ will be connected to one vertex v' in some MST for the rest of G, since the MST will cover the rest of the vertices by definition. And it doesn't matter if there are multiple MSTs for the rest of the vertices, as they will all span all those vertices and will all have the same cost. So it doesn't matter which MST M' we find in our algorithm. Another way to explain why any MST for any graph G such that all vertices $V' \subset V$ are leaves will be of the same form as the spanning trees we generate with our algorithm is the following. Such MSTs are going to have to have all vertices $V - V'$ connected minimimally, which is why we run Prim's on those vertices in our algorithm. And then all vertices in V' need to be leaves and need to be connected with minimum edge weights as well, which is exactly what we do in our algorithm. Thus we can conclude that our algorithm is correct.

Problem 4

We will prove this is true by induction. We will induct on k, the lower bound of the edge weights we remove. Our base case will be the lowest edge weight in the graph. We can see that in this case, we will remove every edge from T_1 and T_2. So clearly, T_1' and T_2' have the same connected components, since both graphs are exactly the same. Now comes our inductive assumption. So, assume that for $k = n$, T_1' and T_2' have the same connected components. Now we must show that for $k = n+1$, T_1'' and T_2'' have the same connected components, where T_1'' and T_2'' are the trees that result from adding back edges of weight $n \leq w < n+1$ from T_1 or T_2 to T_1' and T_2', respectively (in other words, T_1'' and T_2'' are the graphs that result from throwing from T_1 and T_2 every edge weight of at least $n+1$). To show this, we will consider three cases. The first case is if going from $k = n$ to $k = n+1$ results in no edges being added. That is, $T_1'' = T_1'$ and $T_2'' = T_2'$. This case is trivial; before we had the same connected components, so afterwards we do as well. The second case is if going from $k = n$ to $k = n+1$ results in the same set of edges being added to T_1' and T_2'. In this case, it is also clear that T_1'' and T_2'' have the same connected components, since we added the same set of edges to each. The third case is if going from $k = n$ to $k = n+1$ results in a different set of edges being added to T_1' and T_2' where those edges end up connecting the same connected components. That is, we add a set of edges S_1 to T_1', and we add a set of edges S_2 to T_2', where S_1 can differ from S_2, but both sets of edges end up connecting the same connected components. Since we connect the same connected components, then clearly in this case T_1'' and T_2'' have the same connected components. The fourth case is if going from $k = n$ to $k = n+1$ results in T_1'' and T_2'' having different connected components. But we have that this is not actually a case, because it cannot actually happen. To see this, assume to the contrary that T_1'' and T_2'' have different connected components. Then, WLOG, we have that there are connected components X and Y of T_1' that are connected in T_1'' that are not connected in T_2''. But we are going to have to connect those components in T_2 eventually, and when we do so, we are going to use an edge of weight at least $n+1$ (since we will add it for a greater k).

But this is a contradiction, since this means that T_2 is not a MST (since X and Y were connected with a weight of $n \leq w < n+1$ in T_1). Thus we can conclude by induction that T_1' and T_2' have the same connected components for all k.

Problem 5

(a) We will disprove this statement in the following way. We have that perfect matchings have $|V|/2$ edges and that spanning trees have $|V|-1$ edges. So we have that any two perfect matchings for a spanning tree will share at least 1 edge. Thus we can conclude that no spanning tree contains two edge-disjoint perfect matchings.

(b) Let M be a maximal matching, M^* a maximum matching, V the vertices in M, and V^* the vertices in M^*. We wish to prove that $|M| \geq \frac{1}{2}|M^*| \implies 2 \cdot |M| \geq |M^*|$. So, assume to the contrary that $2 \cdot |M| < |M^*|$. Notice that $2 \cdot |M| = |V|$. Also notice that each $v \in V$ can be adjacent to at most one edge in M^*, since M^* is a matching. So we have that, in total, all $|V|$ vertices in V are adjacent to at most $2 \cdot |M|$ edges in M^*. But since $|M^*| > 2 \cdot |M|$, this means there is at least one edge in M^* that is not incident to any vertex in M but is in the graph (since it is in M^*). So we can add this edge to M to enlarge it. But clearly, this is a contradiction. So we can conclude that $|M| \geq \frac{1}{2}|M^*|$ as desired.