(a)

We know that $c_1 < c_2$, with $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ where $E_i = \{uv \in E : c(uv) = c_i\}$ and p_1 and p_2 connected components

This means that G_1 will include the vertices which have been assigned the lower value c_1, and G_2 will include the vertices which have been assigned the greater value. We are trying to prove $min - cost(G) = [c_1(n - p_1) + c_2(p_1 - 1)]$

By the MST cut property, for any cut of a graph, the minimum-weight edge crossing the cut must be included in every MST.

In our case, G_1 has p_1 connected components and only edges of cost c_1. Inside each component, all vertices can be connected using only c_1 edges. There are $n - p_1$ edges of cost c_1 in the MST because each of the p_1 components of G_1 with k_i vertices contributes $k_i - 1$ edges, i.e., $\sum_{i=1}^{p_1}(k_i - 1) = n - p_1$.

To connect the p_1 components into a single spanning tree, we need exactly $p_1 - 1$ edges. These edges cannot be from G_1 (since components are disconnected in G_1), so they must be from G_2 with cost c_2.

Therefore, the MST uses $n - p_1$ edges of cost c_1 and $p_1 - 1$ edges of cost c_2, giving total cost: $min\text{-}cost(G) = c_1(n - p_1) + c_2(p_1 - 1)$.

(b)

First, we create G_1 and G_2 as the problem states, with $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. Thus, we can separate the problem into finding trees with all edge values equal to c_1 and then connecting them with c_2 cost edges.

To obtain all the c_1 cost edges needed for the MST, we can run a DFS or BFS algorithm on all connected components of G_1. Since all costs are the same, any resulting tree is optimal for our MST. While doing this step, it is important to mark each component with an ID, in order to set up the next step using Union-Find.

We iterate through G_2's edges. For each $uv \in G_2$, if $find(u) \neq find(v)$ then we perform $union(u, v)$ and add uv to the MST. Exactly $p_1 - 1$ edges from G_2 are needed to connect all components.

- Creating G_1 and G_2 happens in $O(n + m)$.
- Iterating through G_1's connected components happens in $O(n)$.
- Iterating through G_2's edges and performing the Union-Find operations happens in $O(m \cdot \alpha(n))$ amortized time, with m as an upper bound for the size of E_2 and $\alpha(n)$ denoting the inverse Ackermann function.

Given that the inverse Ackermann function is upper bounded by 5, we can say that the suggested algorithm runs in $O(n + m)$ in practice.