3. O algoritmo funciona com base no fato de que os vértices das folhas sempre vão possuir excentricidade maior do que os outros vértices. Isso acontece já que, se um vértice é uma folha, quer dizer que existe um outro vértice (que se conecta com ele) que está mais perto do "resto do grafo" do que a folha. Já que temos certeza de que nenhuma folha será o centro do grafo, podemos retirá-las da fila de possíveis candidatos. Ao fazer isso, obtemos um novo subgrafo. O centro desse novo subgrafo é, por definição, o mesmo do grafo original, já que qualquer distância entre o centro e uma folha foi reduzida em 1, então todos os candidatos a menor caminho também reduziram em 1. Sabendo disso, podemos repetidamente retirar as folhas do grafo, camada por camada, até não podermos mais retirar folhas sem que apaguemos o mesmo. Já que estamos analisando uma árvore, é impossível esse último passo acabar com mais de três vértices (ou então haveria um ciclo), então podemos parar o algoritmo quando o tamanho do subgrafo for igual ou menor do que dois. O pseudocódigo pode ser feito assim:

```
Enquanto Tamanho_da_Arvore <= 2 faça
    Folhas ← Lista_Vazia()
    Para cada vértice em Arvore faça
        Se Grau_do_Vertice = 1 então
            Adicione Vértice a Folhas
    Para cada vértice em Folhas faça
        Remova Vértice de Arvore

Retorne Arvore
```

4. a) Na questão original da prova, eu respondi que Encontrar a MST, por qualquer algoritmo que seja, e depois disso retirar suas k-1 maiores arestas seria o suficiente para encontrar k conjuntos. Mesmo com essa frase ainda estando correta, é possível identificar um algoritmo que realize o processo mais eficientemente por pular alguns passos. O algoritmo de Kruskal para encontrar uma MST é visivelmente melhor nesse caso já que ele junta os vértices em ordem crescente de acordo com o tamanho das arestas, logo podemos pedir que ele pare o algoritmo quando já tiver formado esses k-1 conjuntos.
O pseudocodigo ficará assim:

```
K_Clusters(Pontos, K)
N ← número de pontos em Pontos
Se K ≥ N:
    Retorne uma lista de K clusters, cada um contendo um ponto de Pontos

Arestas ← uma lista vazia de arestas
Para i de 1 até N:
    Para j de i+1 até N:
        Calcule a distância entre Pontos[i] e Pontos[j] e adicione (i, j, distância) a
Arestas

Ordene Arestas em ordem crescente de distância
Conjuntos ← uma lista de N conjuntos iniciais, um para cada ponto em Pontos
Número_de_Clusters ← N
Clusters ← uma lista vazia de clusters
```

```
Para cada aresta (u, v, distância) em Arestas:
    Se Conjunto(u) ≠ Conjunto(v):
        Una Conjunto(u) e Conjunto(v)
        Reduza Número_de_Clusters em 1

    Se Número_de_Clusters = K:
        Pare

Para i de 1 até N:
    Adicione Pontos[i] a Clusters[Conjunto(i)]

Remova clusters vazios de Clusters

Retorne Clusters
```

4. b) Essa questão foi implementada em Sage e enviada juntamente desse pdf.