1. This algorithm takes $O(m)$ to process the whole sequence. That is because the algorithm runs on the inputs which are the edges(m).
2. T

union (A, B)
union (A, C)
find (C)
Output: set 0
find (D)
Output: set 1
union (G, H)
union (F, G)
find (H)
Output: set 3
union (C, F)
find (H)
Output: set 0

3. This wouldn't work in this case because id2 needs to be stored outside of the for-loop. Or, the value of ID[i] will be compared to the new value of id2 instead of the original value of id2
4. R
5. 5

Finding the MST by "Eyeballing" the weighted graph:
0 →1→3→2→4→6

Finding another spanning tree that's not minimal:
0 →1→3→2→6

6. Because the MST is the most efficient path between two vertices has to use the least weighted edge to reach the end vertex. As a result, we get the most efficient path from two nodes with the least weight possible.
7.
8. For a sparse graph, you have less edges than the vertices, and for a dense graph you have much more edges. When forming an adjacency-matrix, the time complexity is O(v2) and for an adjacency-list it's O(E).
   - Dense: $O(ElogE) \to O(Elogv^*) \to O(2ElogE) \to O(ElogE)$
   - Sparse: $O(Elogv) \to O(ElogE^*) \to O(2Elogv) \to O(ElogE)$
9.
10. Algorithm: buildMatrixUsingPred(int[] predecessor)
    int[][] retMatrix = matrix for i = 0 to matrix.length
    for j = 0 to matrix[i].length
      If i != j AND i = predecessor[j]
      Return end-if

```
        else
        ReturnMatrix[i][j] = -1
      end-else end-for
    end-for
    return retMatrix
```

11.

Sorted linked list.
Worst-case of extractMin $O(1)$
Worst-case of decreaseKey $O(n)$

Unsorted array.
Worst-case of extractMin $O(n)$
Worst-case of decreaseKey is $O(1)$

12.
13. We can use Dijkstra's Algorithm which can use either while loops or recursion. It has to iterate through each node and find a path for every vertex.
    Running time estimate would be $O(V + ElogV)$.