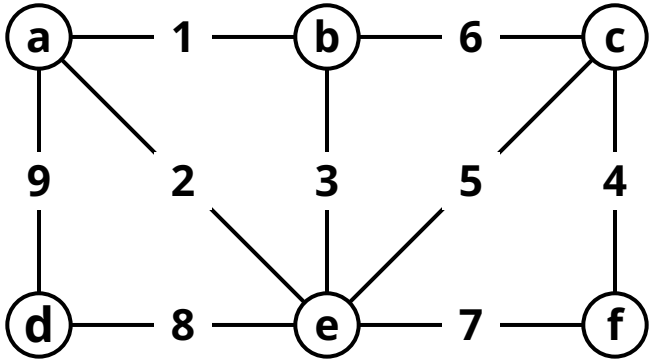
Give the order in which Kruskal's algorithm selects edges.



After considering the edge with weight **2**, circle the connected components above.

Kruskal's Code

```
DisjointSets<V> trees = new DSImpl<V>(graph.vertices());

List<Edge<V>> edges = graph.edges();
edges.sort(Double.comparingDouble(Edge::weight));

List<Edge<V>> result = new ArrayList<>();
int i = 0;
while (result.size() < graph.vertices().size() - 1) {
    Edge<V> e = edges.get(i);

    if (!trees.find(e.from).equals(trees.find(e.to))) {
            trees.union(e.from, e.to);
            result.add(e);
        }
        i += 1;
    }

    return result;
}
```

Kruskal's algorithm relies on two **disjoint set** operations: `find`, which returns the set representative (e.g. smallest value in set), and `union`, which combines two sets.

If we know that Kruskal's algorithm is in Θ(|E| log |E|) time overall, fill in the blanks with big-O bounds for the calls to `find` and `union`.

1. Merge sort the list of edges in the graph. **Θ(|E| log |E|)**
2. While the size of the MST < |V| - 1…
   a. Check if **`find(e.from).equals(find(e.to))`**. _____
   b. If false, **`union(e.from, e.to)`** and add **`e`** to result. _____

What is the runtime if we use a **`List<Set<Node>>`** to implement `find` and `union`?

1. Merge sort the list of edges in the graph. **Θ(|E| log |E|)**
2. While the size of the MST < |V| - 1…
   a. Check if **`find(e.from).equals(find(e.to))`**. _____
   b. If false, **`union(e.from, e.to)`** and add **`e`** to result. _____

Prim's algorithm is in $\Theta(|E| \log |V|)$ time overall. To get Kruskal's to match this, we need an approach that has $O(\log |V|)$ time for both `find` and `union`.

Quick Find. Optimizes for the `find` operation.

Quick Union. Optimizes for the `union` operation, but doesn't really succeed.

Weighted Quick Union. Addresses the worst-case height problem in quick union.

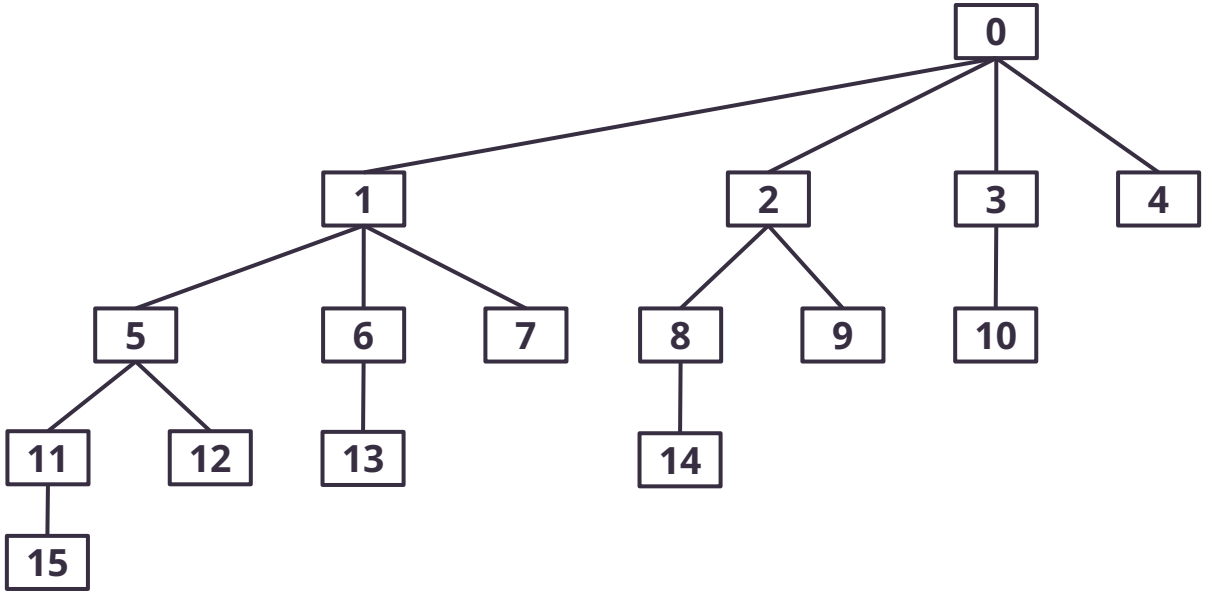
Let's speed up Kruskal's algorithm! If our edge weights are restricted to `int` values **bounded between 0 and 255**, fill in the blanks with your best possible runtimes.

1. _____ sort the list of edges in the graph. _____
2. While the size of the MST < |V| - 1...
   a. Check if **`find(e.from).equals(find(e.to))`**. _____
   b. If false, **`union(e.from, e.to)`** and add **`e`** to result. _____

What is the runtime if we have a faster disjoint sets implementation, **weighted quick union with path compression**, where `find` and `union` are in $\Theta(\alpha|V|)$?

1. _____ sort the list of edges in the graph. _____
2. While the size of the MST < |V| - 1...
   a. Check if **`find(e.from).equals(find(e.to))`**. _____
   b. If false, **`union(e.from, e.to)`** and add **`e`** to result. _____

Path compression reassigns the parent of each node visited by `find` to the root.



Analogous to balancing a search tree by rotations or color flips. Θ(α(N)) **amortized**.