Partitions with Union-Find Operations (§ 11.6)

- **makeSet(x)**: Create a singleton set containing the element $x$ and return the position storing $x$ in this set.
- **union(A, B)**: Return the set $A \cup B$, destroying the old $A$ and $B$.
- **find(p)**: Return the set containing the element in position $p$.

Tree-based Implementation (§ 11.6.3)
- Each element is stored in a node, which contains a pointer to a set name.
- A node $v$ whose set pointer points back to $v$ is also a set name.
- Each set is a tree, rooted at a node with a self-referencing set pointer.
- For example: The sets “1”, “2”, and “5”:

Union-Find Operations
- To do a union, simply make the root of one tree point to the root of the other.
- To do a find, follow set-name pointers from the starting node until reaching a node whose set-name pointer refers back to itself.

Union-Find Heuristic 1
- **Union by size**:
  - When performing a union, make the root of smaller tree point to the root of the larger.
  - Implies $O(n \log n)$ time for performing $n$ union-find operations:
    - Each time we follow a pointer, we are going to a subtree of size at least double the size of the previous subtree.
    - Thus, we will follow at most $O(\log n)$ pointers for any find.

Union-Find Heuristic 2
- **Path compression**:
  - After performing a find, compress all the pointers on the path just traversed so that they all point to the root.
  - Implies $O(n \log^* n)$ time for performing $n$ union-find operations:
    - Proof is somewhat involved... (and not in the book)