1.5 Case-Study: Union-Find

*p* and *q* have **equivalence relation**:

* Reflexive: *p* is connected to *p*
* Symmetric: If *p* is connected to *q*, then *q* is connected to *p*.
* Transitive: If *p* is connected to *q* and *q* is connected to *r*, then *p* is connected to *r*.

Eg. Networks, variable-name equivalence, mathematical sets 🡪 **dynamic connectivity problems**

Union-find cost model: we count array accesses (the number of times an array entry is accessed, for read or write) 🡪 run time: within constant factors

**Quick-find**:

* maintain the invariant that p and q are connected iff id[p] is equal to id[q];
* quadratic-time process (N+3)(N-1) ~N­­2
* bad if there are a lot of data to union ()

**Quick-union**:

* + A link and common root
  + Forest-tree representation
    - Size of a tree is its number of nodes
    - Depth of a node in a tree is the number of links on the path from it to the root
    - Height of a tree is the maximum depth among its nodes
  + Union() is always linear
  + The number of array accesses used by *find()* in quick-union is 1 plus the twice the depth of the node corresponding to the given site. The number of array accesses used by *union()* and *connected()* is the cost of the two *find()* operations (plus 1 for *union()* if the given sites are in different trees).

**Weighted quick-union**:

* Keep track of the size of the tree and always connect the smaller to the larger (prevent worst cast in quick-union)
* The depth of any node in a forest built by weighted quick-union for N sites is at most lgN
* Worst-case order of growth of the cost of *find(),* *connected(),* and *union()* is logN

**Weighted quick-union with path compression:**

* Optimal but not quite constant-time per operation—amortized

\* study weighted quick-union more