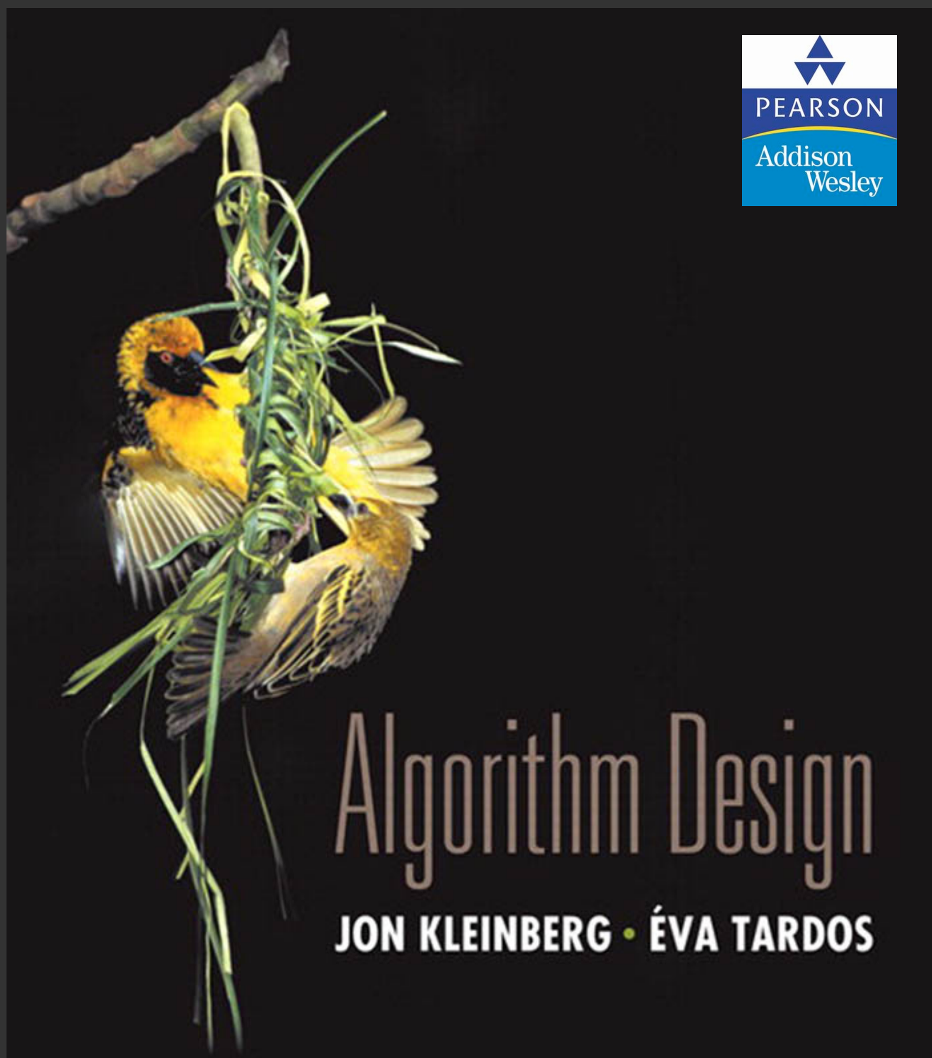


Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

UNION–FIND

- *naïve linking*
- *link-by-size*

Disjoint-sets data type

Goal. Support three operations on a collection of disjoint sets.

- MAKE-SET(x): create a new set containing only element x.
- FIND(x): return a canonical element in the set containing x.
- UNION(x, y): replace the sets containing x and y with their union.

Performance parameters.

- m = number of calls to MAKE-SET, FIND, and UNION.
- n = number of elements = number of calls to MAKE-SET.

Dynamic connectivity. Given an initially empty graph G, support three operations. ← disjoint sets = connected components

- ADD-NODE(u): add node u. ← 1 MAKE-SET operation
- ADD-EDGE(u, v): add an edge between nodes u and v. ← 1 UNION operation
- IS-CONNECTED(u, v): is there a path between u and v? ← 2 FIND operations

Disjoint-sets data type: applications

Original motivation. Compiling EQUIVALENCE, DIMENSION, and COMMON statements in Fortran.

An Improved Equivalence Algorithm

Bernard A. Galler and Michael J. Fisher
University of Michigan, Ann Arbor, Michigan

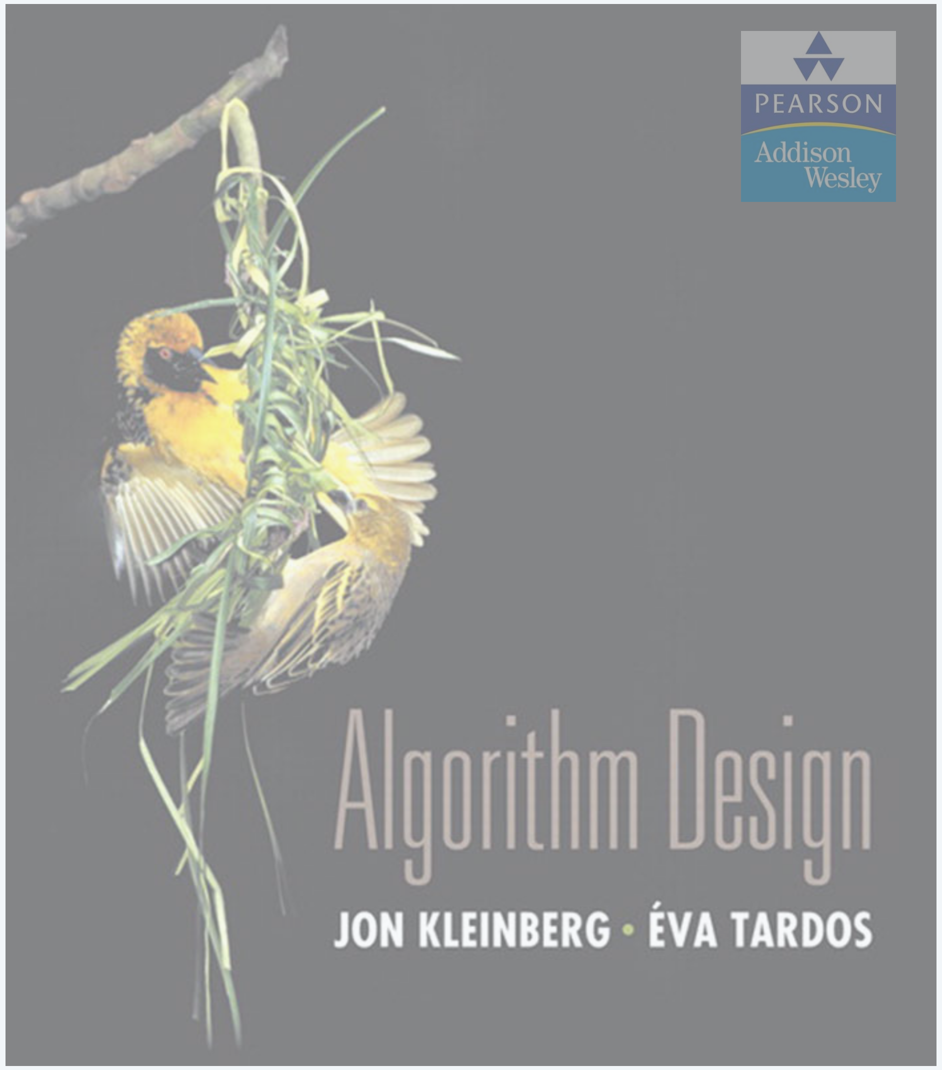
An algorithm for assigning storage on the basis of EQUIVALENCE, DIMENSION and COMMON declarations is presented. The algorithm is based on a tree structure, and has reduced computation time by 40 percent over a previously published algorithm by identifying all equivalence classes with one scan of the EQUIVALENCE declarations. The method is applicable in any problem in which it is necessary to identify equivalence classes, given the element pairs defining the equivalence relation.

Note. This 1964 paper also introduced key data structure for problem.

Disjoint-sets data type: applications

Applications.

- Percolation.
- Kruskal's algorithm.
- Connected components.
- Computing LCAs in trees.
- Computing dominators in digraphs.
- Equivalence of finite state automata.
- Checking flow graphs for reducibility.
- Hoshen–Kopelman algorithm in physics.
- Hinley–Milner polymorphic type inference.
- Morphological attribute openings and closings.
- Matlab's BW-LABEL function for image processing.
- Compiling EQUIVALENCE, DIMENSION and COMMON statements in Fortran.
- ...



UNION–FIND

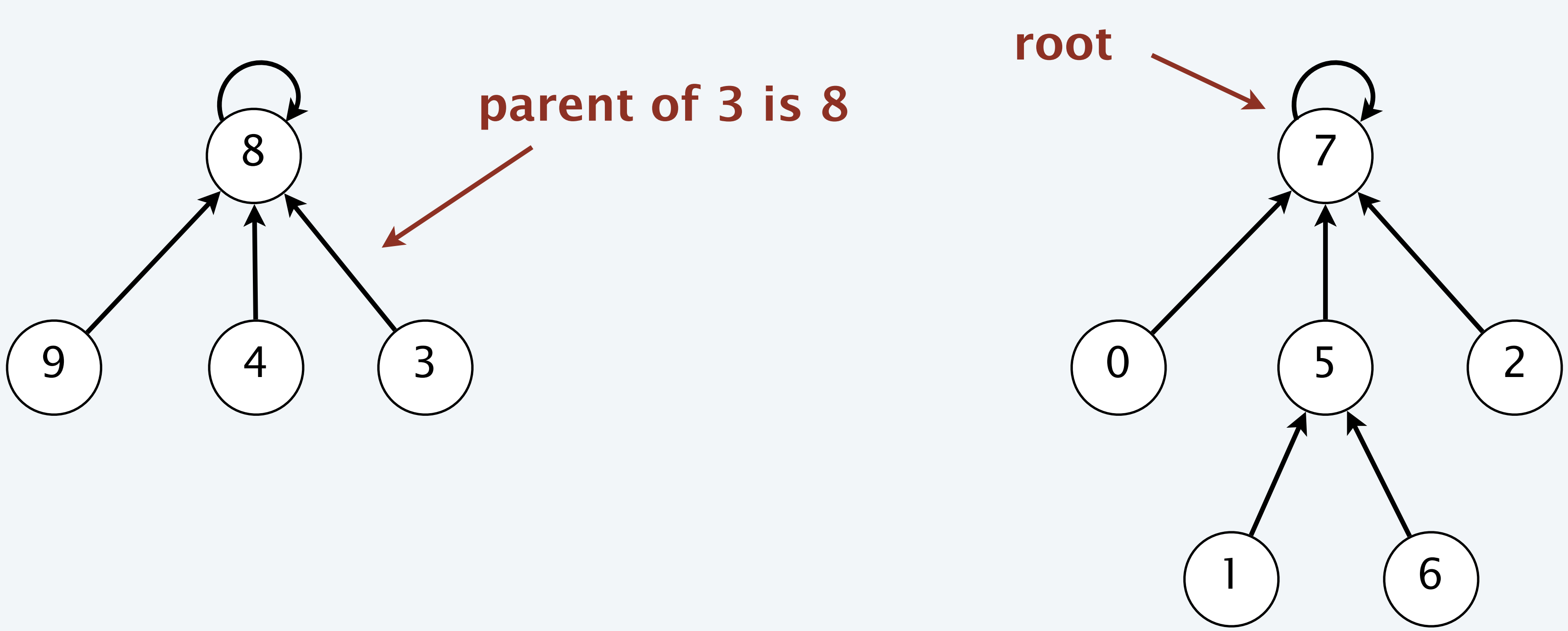
- *naïve linking*
- *link-by-size*

Disjoint-sets data structure

Parent-link representation. Represent each set as a tree of elements.

- Each element has an explicit parent pointer in the tree.
- The root serves as the canonical element (and points to itself).
- FIND(x): find the root of the tree containing x.
- UNION(x, y): merge trees containing x and y (by making one root point to the other root).

UNION(3, 5)

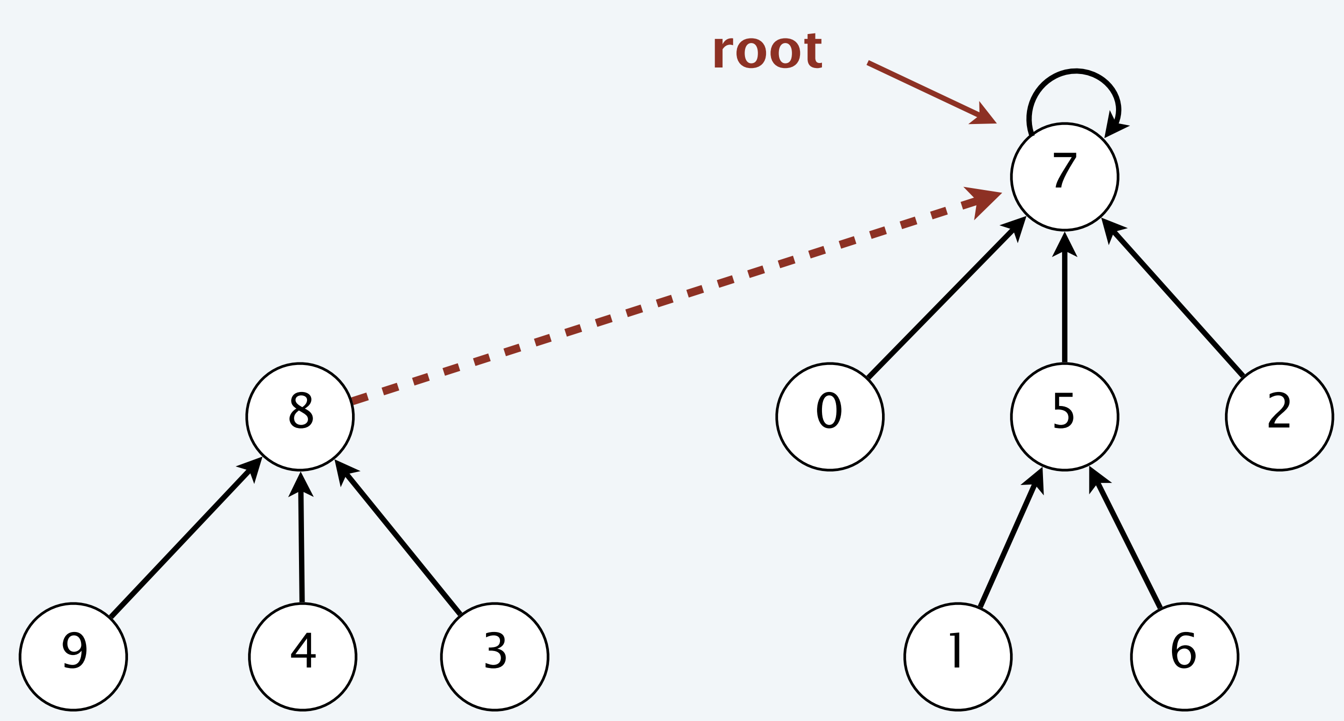


Disjoint-sets data structure

Parent-link representation. Represent each set as a tree of elements.

- Each element has an explicit parent pointer in the tree.
- The root serves as the canonical element (and points to itself).
- FIND(x): find the root of the tree containing x.
- UNION(x, y): merge trees containing x and y (by making one root point to the other root).

UNION(3, 5)

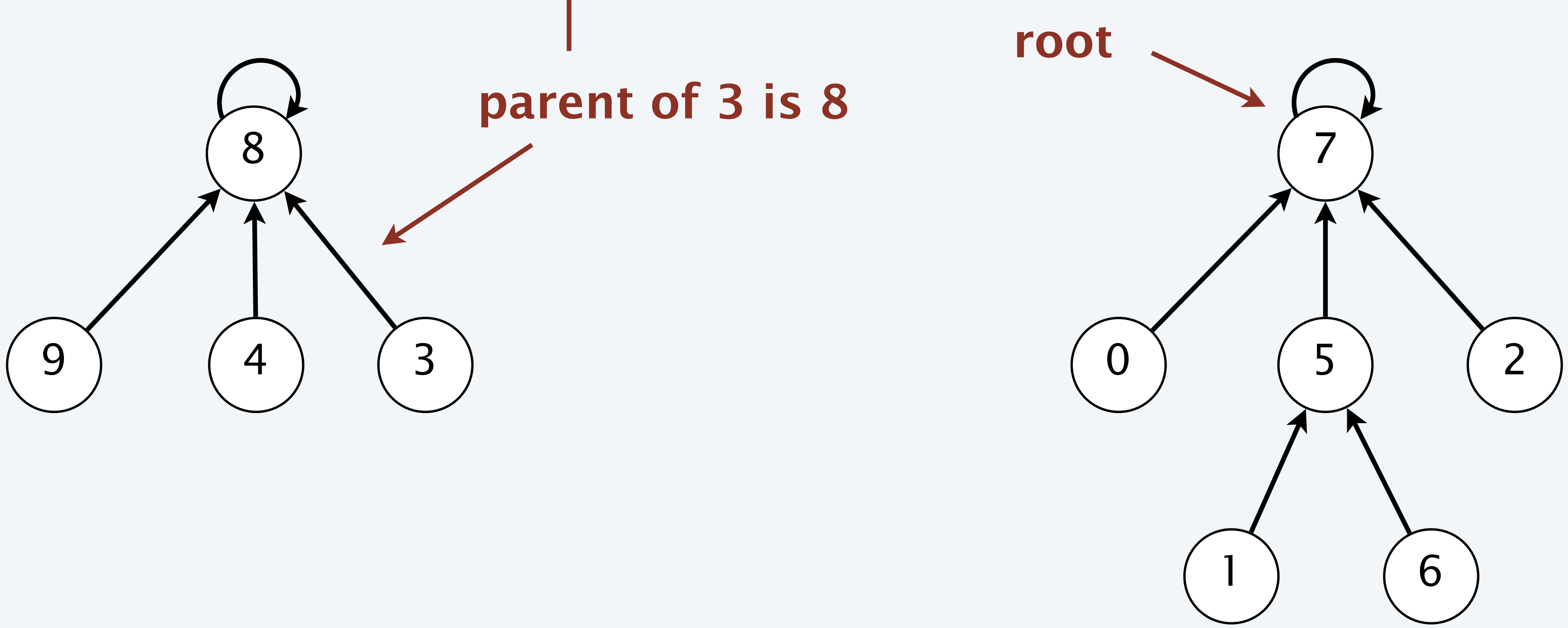


Disjoint-sets data structure

Array representation. Represent each set as a tree of elements.

- Allocate an array $parent[]$ of length n. ← must know number of elements n a priori
- $parent[i] = j$ means parent of element i is element j.

	0	1	2	3	4	5	6	7	8	9
parent[]	7	5	7	8	8	7	5	7	8	8

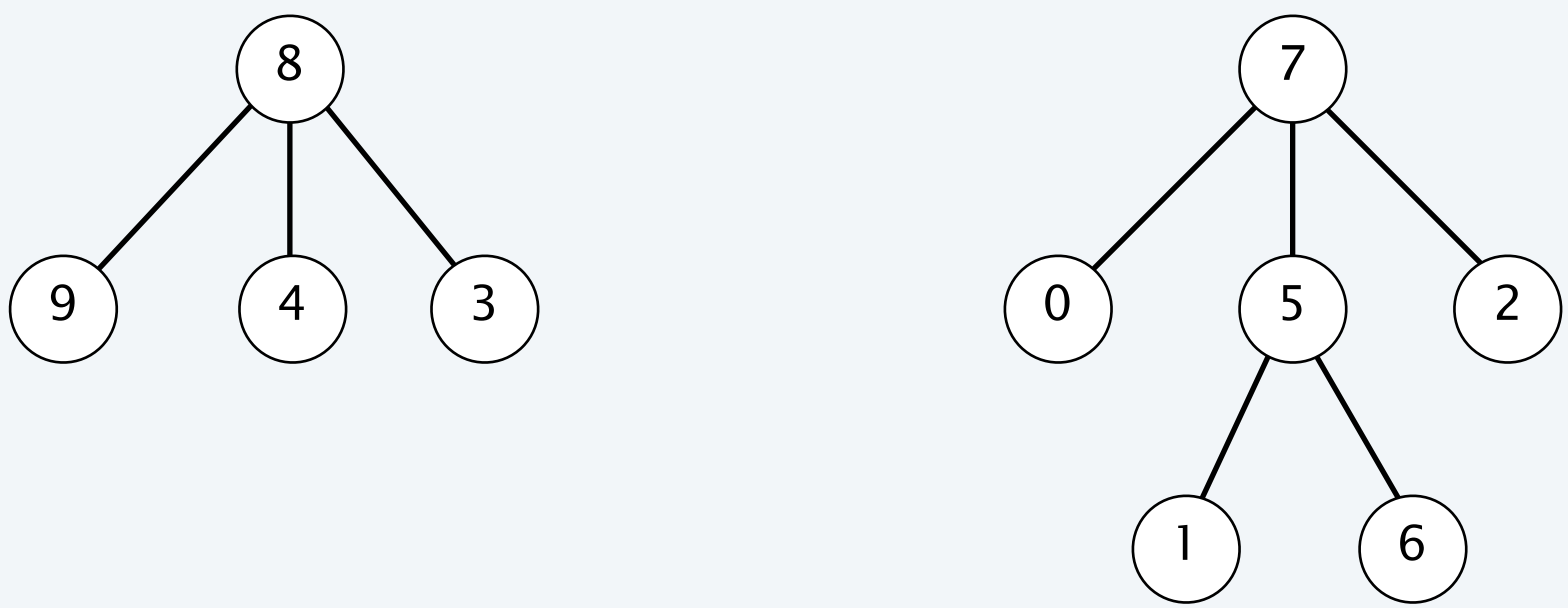


Note. For brevity, we suppress arrows and self-loops in figures.

Naïve linking

Naïve linking. Link root of first tree to root of second tree.

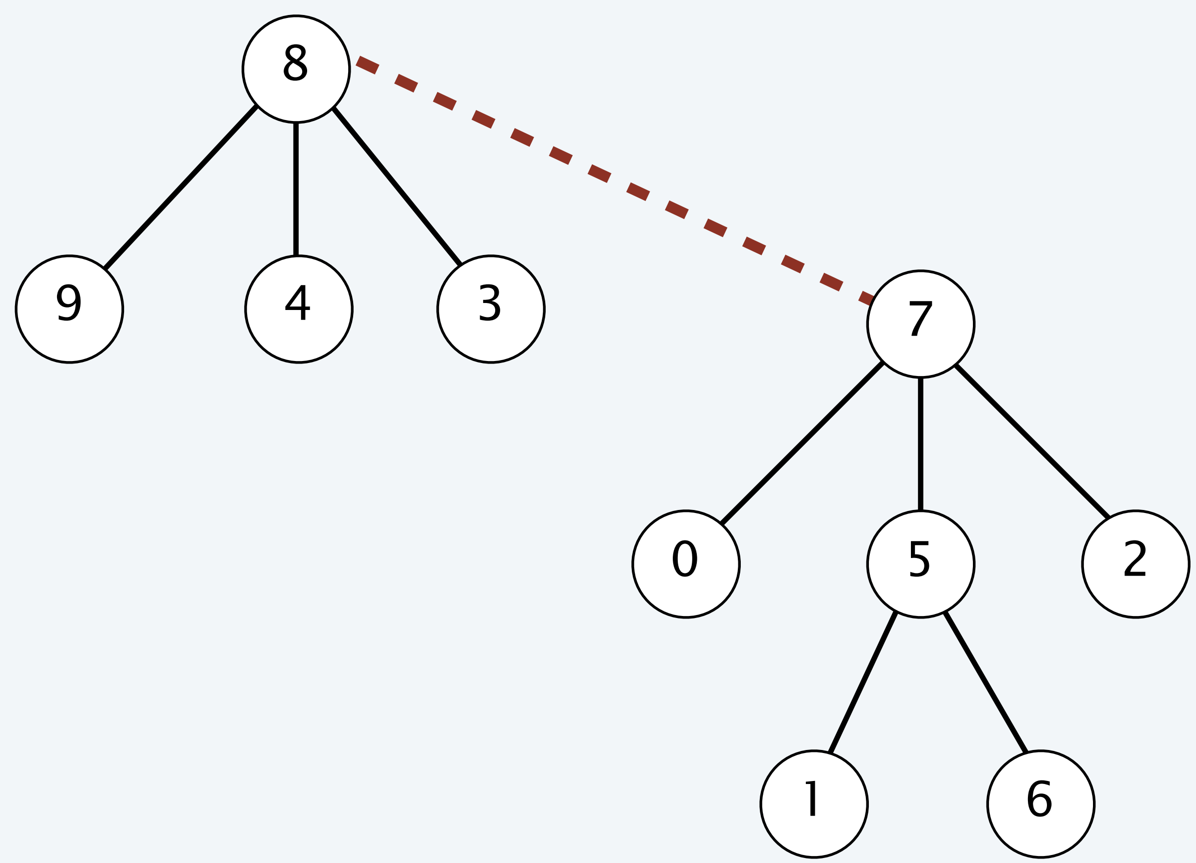
UNION(5, 3)



Naïve linking

Naïve linking. Link root of first tree to root of second tree.

UNION(5, 3)



Naïve linking

Naïve linking. Link root of first tree to root of second tree.

MAKE-SET(x)

$parent[x] \leftarrow x.$

FIND(x)

WHILE $(x \neq parent[x])$

$\quad x \leftarrow parent[x].$

RETURN x.

UNION(x, y)

$r \leftarrow$ FIND(x).

$s \leftarrow$ FIND(y).

$parent[r] \leftarrow s.$

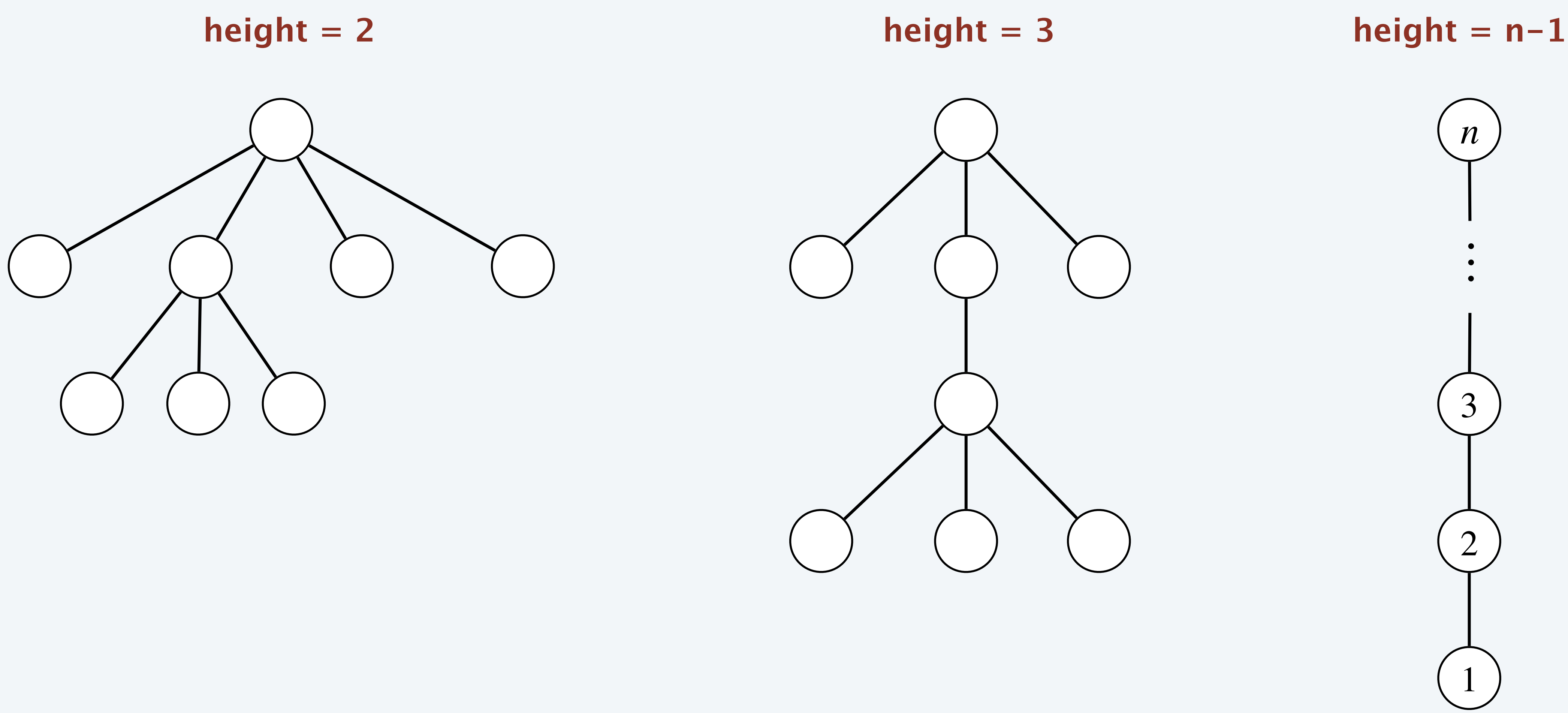
Naïve linking: analysis

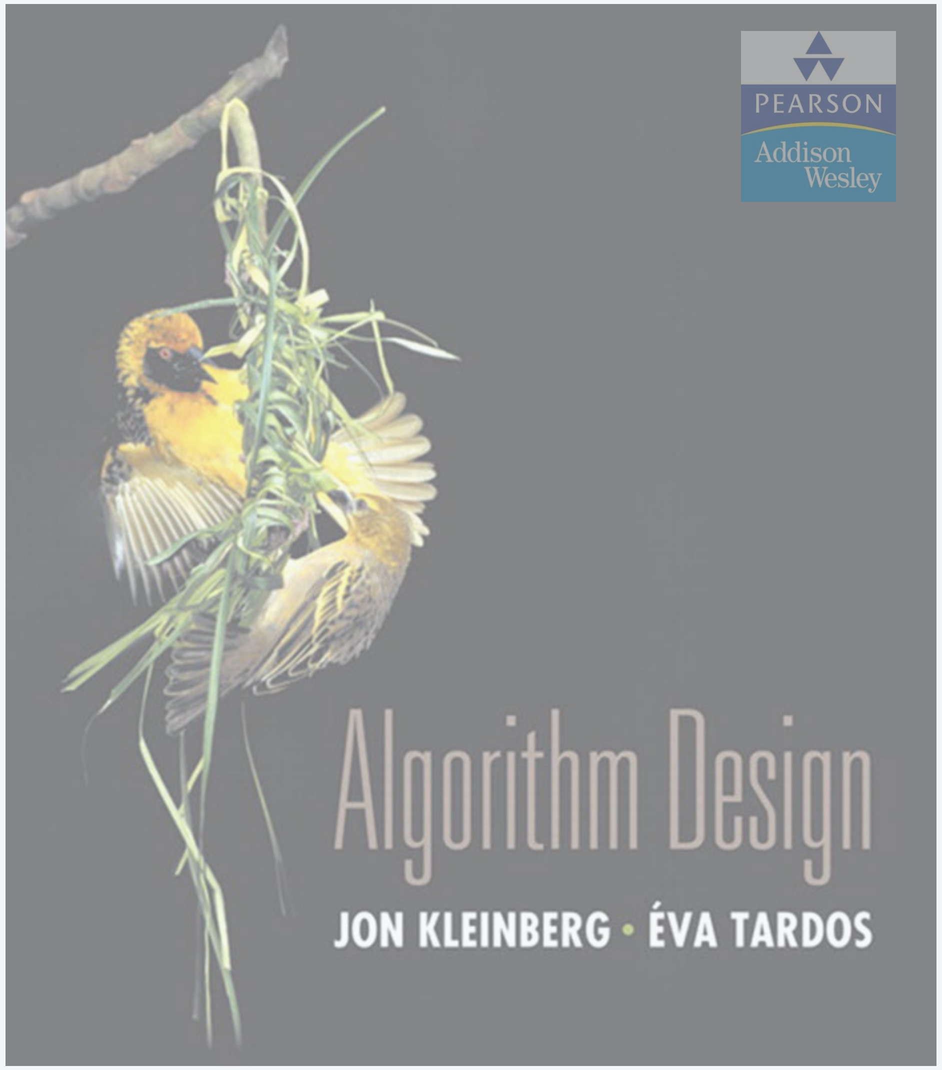
Theorem. Using naïve linking, a UNION or FIND operation can take $\Theta(n)$ time in the worst case, where n is the number of elements.

max number of links on any path from root to leaf node

Pf.

- In the worst case, FIND takes time proportional to the height of the tree.
- Height of the tree is $n - 1$ after the sequence of union operations: UNION$(1, 2)$, UNION$(2, 3)$, …, UNION$(n - 1, n)$.





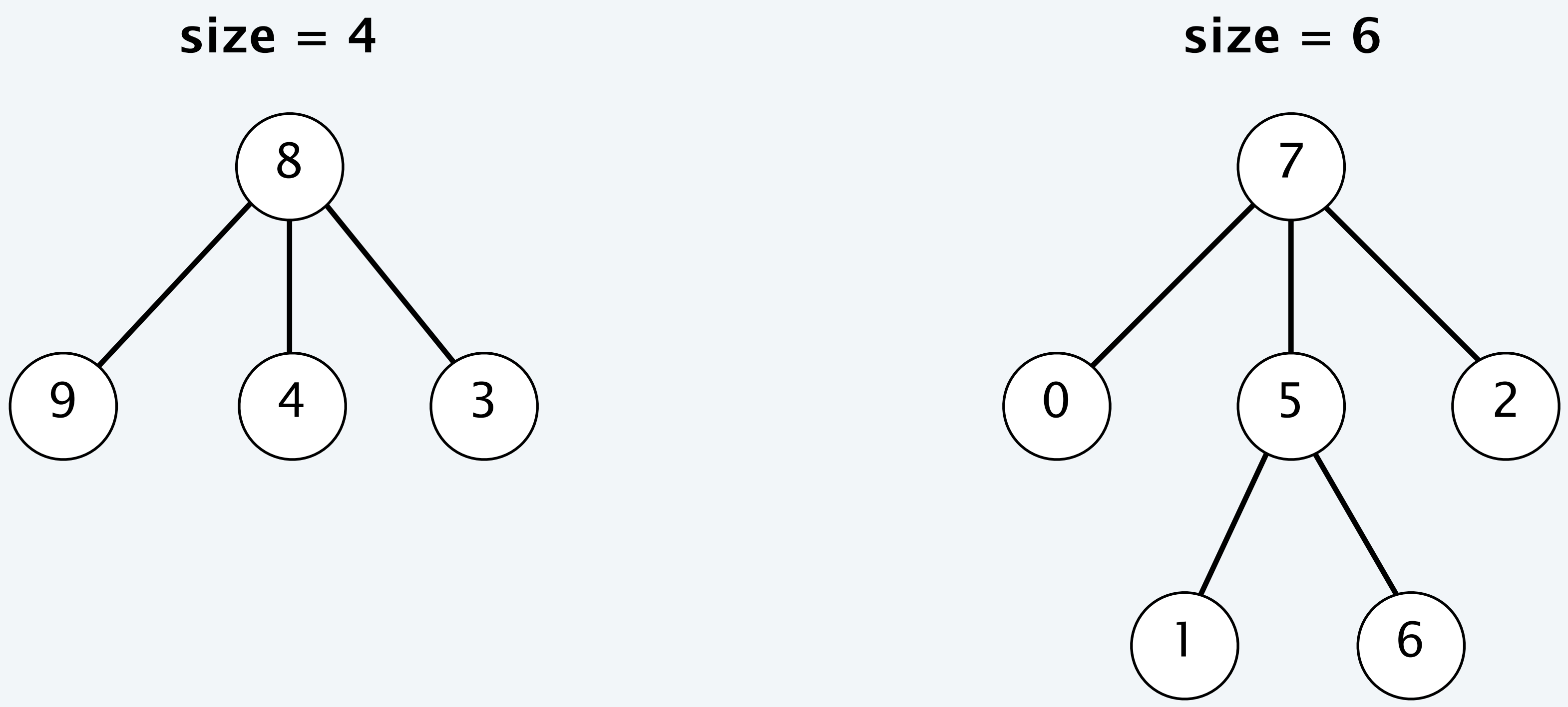
UNION–FIND

- *naïve linking*
- *link-by-size*

Link-by-size

Link-by-size. Maintain a tree size (number of nodes) for each root node.
Link root of smaller tree to root of larger tree (breaking ties arbitrarily).

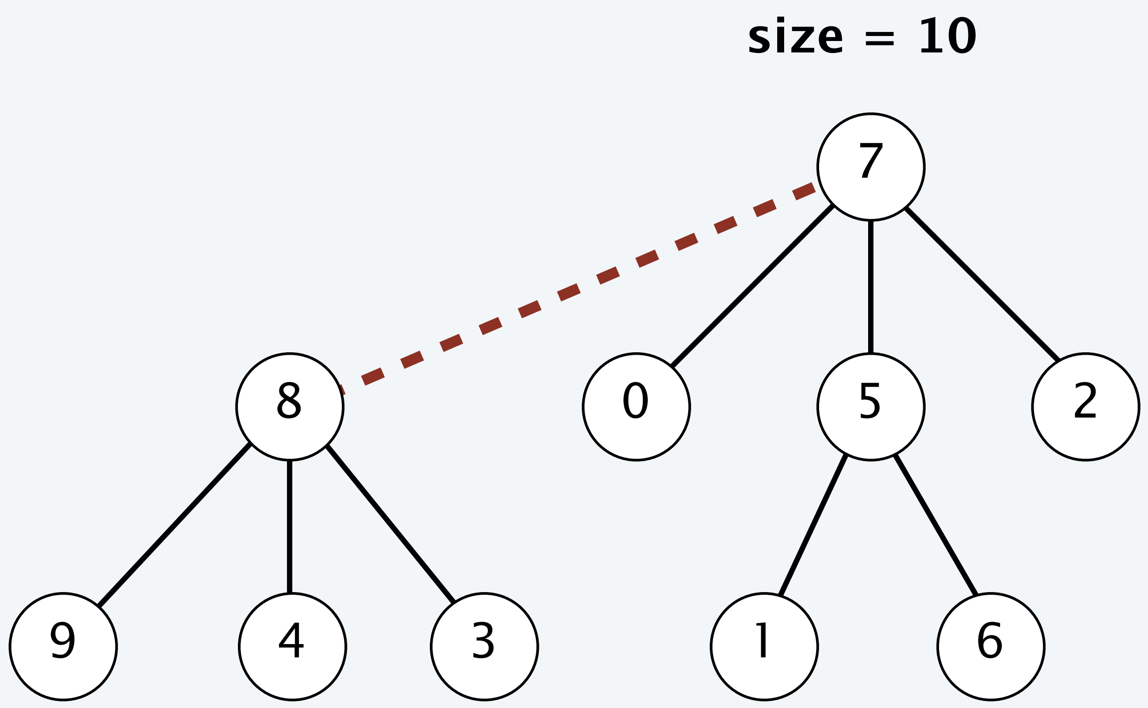
UNION(5, 3)



Link-by-size

Link-by-size. Maintain a tree size (number of nodes) for each root node. Link root of smaller tree to root of larger tree (breaking ties arbitrarily).

UNION(5, 3)



Link-by-size

Link-by-size. Maintain a tree size (number of nodes) for each root node. Link root of smaller tree to root of larger tree (breaking ties arbitrarily).

MAKE-SET(x)

$parent[x] \leftarrow x$.

$size[x] \leftarrow 1$.

FIND(x)

WHILE ($x \neq parent[x]$)

  $x \leftarrow parent[x]$.

RETURN x.

UNION(x, y)

$r \leftarrow$ FIND(x).

$s \leftarrow$ FIND(y).

IF ($r = s$) RETURN.

ELSE IF ($size[r] > size[s]$)

  $parent[s] \leftarrow r$.

  $size[r] \leftarrow size[r] + size[s]$.

ELSE

  $parent[r] \leftarrow s$.

  $size[s] \leftarrow size[r] + size[s]$.

link-by-size

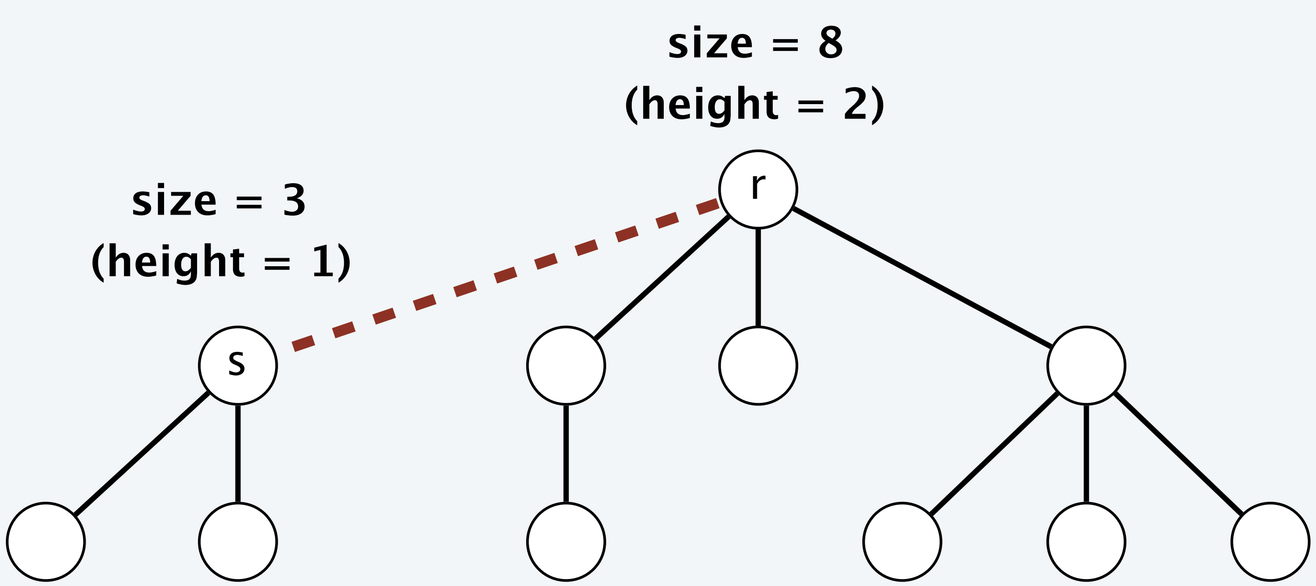
Link-by-size: analysis

Property. Using link-by-size, for every root node r : $size[r] \geq 2^{height(r)}$.

Pf. [by induction on number of links]

- Base case: singleton tree has size 1 and height 0.
- Inductive hypothesis: assume true after first i links.
- Tree rooted at r changes only when a smaller (or equal) size tree rooted at s is linked into r.
- Case 1. [$height(r) > height(s)$]

$$\begin{aligned} size'[r] &> size[r] \\ &\geq 2^{height(r)} \quad \longleftarrow \text{inductive hypothesis} \\ &= 2^{height'(r)}. \end{aligned}$$



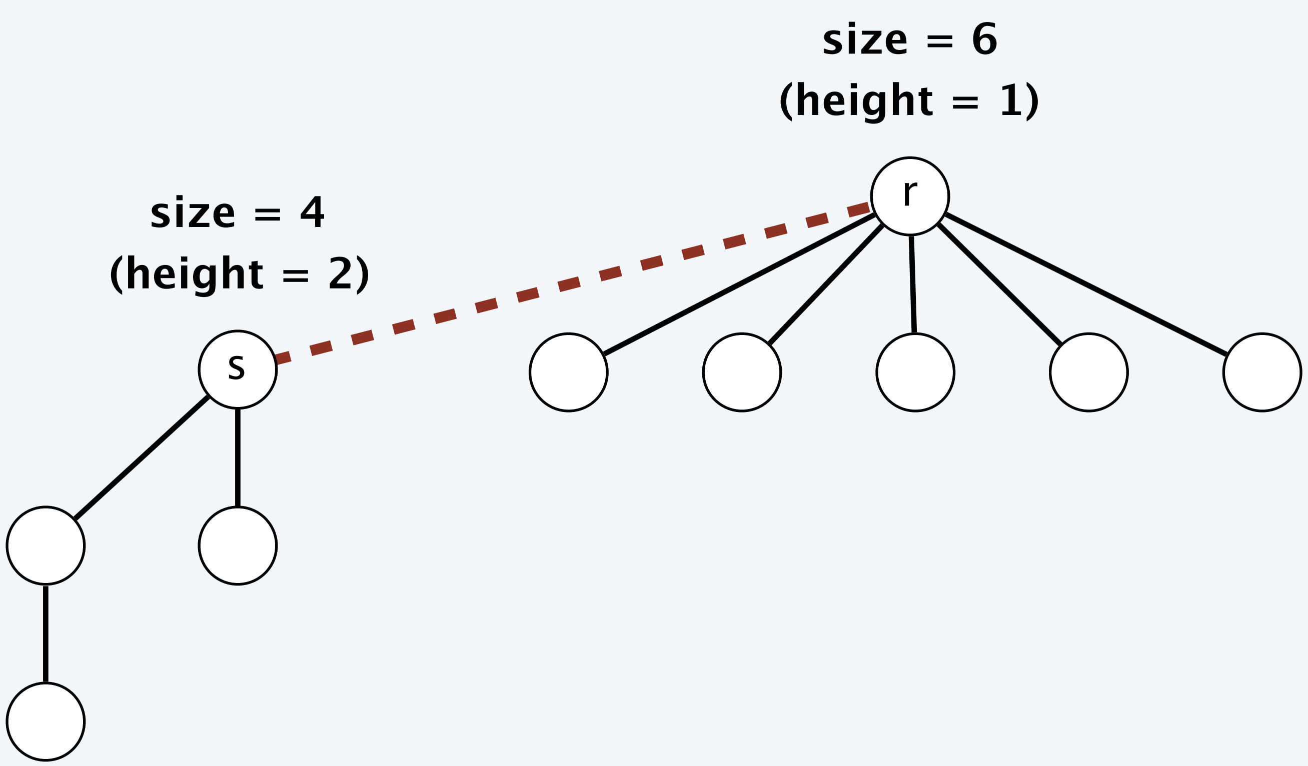
Link-by-size: analysis

Property. Using link-by-size, for every root node r : $size[r] \geq 2^{height(r)}$.

Pf. [by induction on number of links]

- Base case: singleton tree has size 1 and height 0.
- Inductive hypothesis: assume true after first i links.
- Tree rooted at r changes only when a smaller (or equal) size tree rooted at s is linked into r.
- Case 2. [$height(r) \leq height(s)$]

$$\begin{aligned} size'[r] &= size[r] + size[s] \\ &\geq 2\, size[s] && \longleftarrow \text{link-by-size} \\ &\geq 2 \cdot 2^{height(s)} && \longleftarrow \text{inductive hypothesis} \\ &= 2^{height(s)+1} \\ &= 2^{height'(r)}. \quad \blacksquare \end{aligned}$$



Link-by-size: analysis

Theorem. Using link-by-size, any UNION or FIND operation takes $O(\log n)$ time in the worst case, where n is the number of elements.

Pf.

- The running time of each operation is bounded by the tree height.
- By the previous property, the height is $\leq \lfloor \lg n \rfloor$. ▪

$\lg n = \log_2 n$

Note. The UNION operation takes $O(1)$ time except for its two calls to FIND.

A tight upper bound

Theorem. Using link-by-size, a tree with n nodes can have height $= \lg n$.

Pf.

- Arrange $2^k - 1$ calls to UNION to form a binomial tree of order k.
- An order-k binomial tree has 2^k nodes and height k. ▪