CSE221

Lecture 21:
Disjoint Sets

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong. Some slides are based on https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/UnionFind.pdf

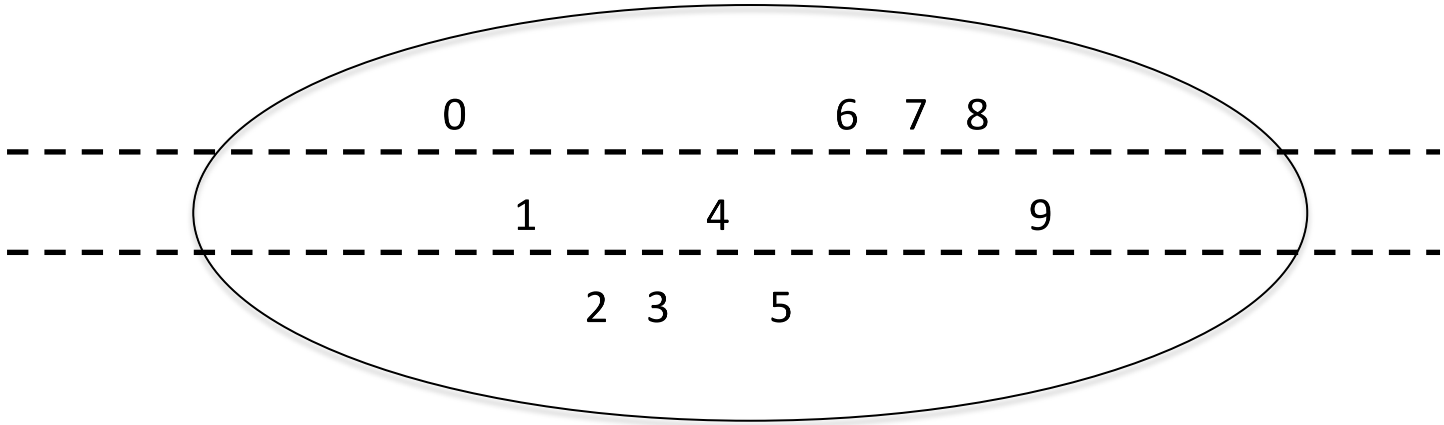
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Disjoint Sets

- A disjoint set is a partition of a set of elements.
- For example, S = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

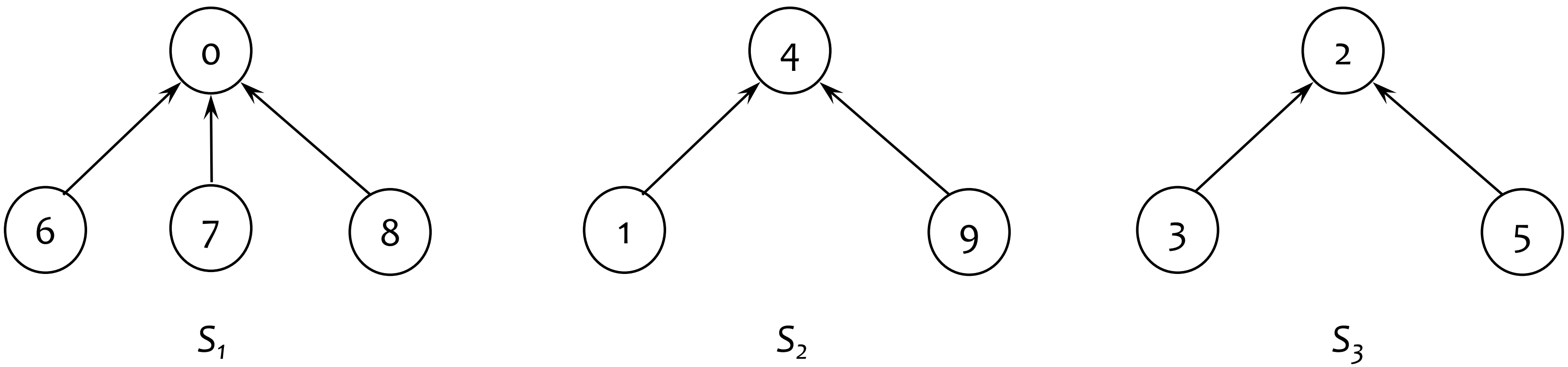
A partition of S is

{ {0, 6, 7, 8}, {1, 4, 9}, {2, 3, 5} }



Tree Representation of Sets

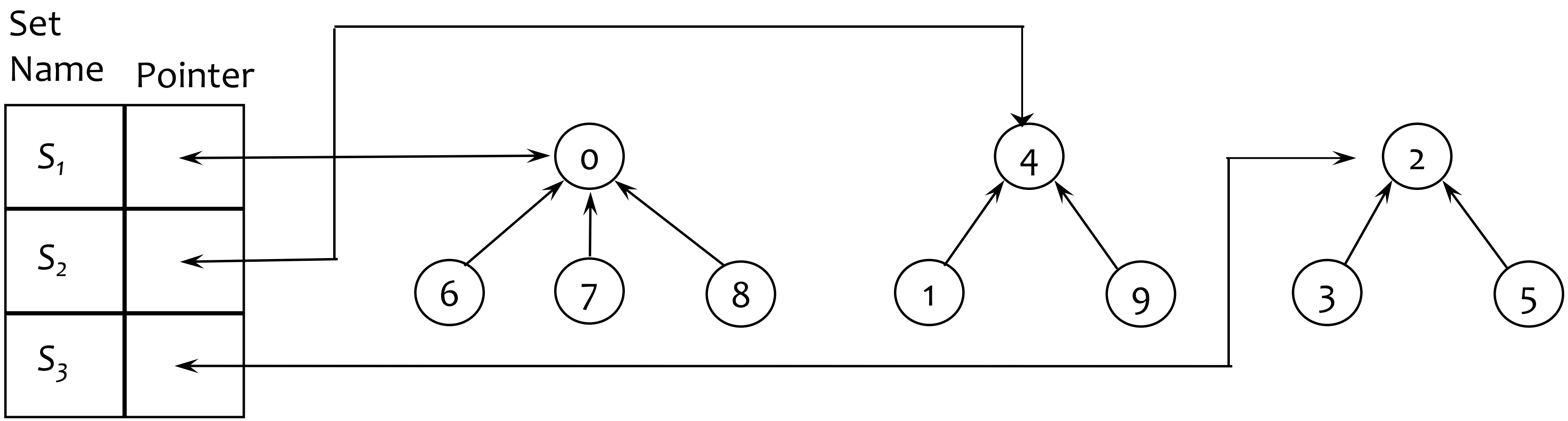
- Set representation using a tree



- The element at the root = the identity of the set
  - For all elements in a set, chasing the pointers will reach the same root node.
- Easy to check whether two elements are in the same set.

Set Representation

- Data representation



- Array representation

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

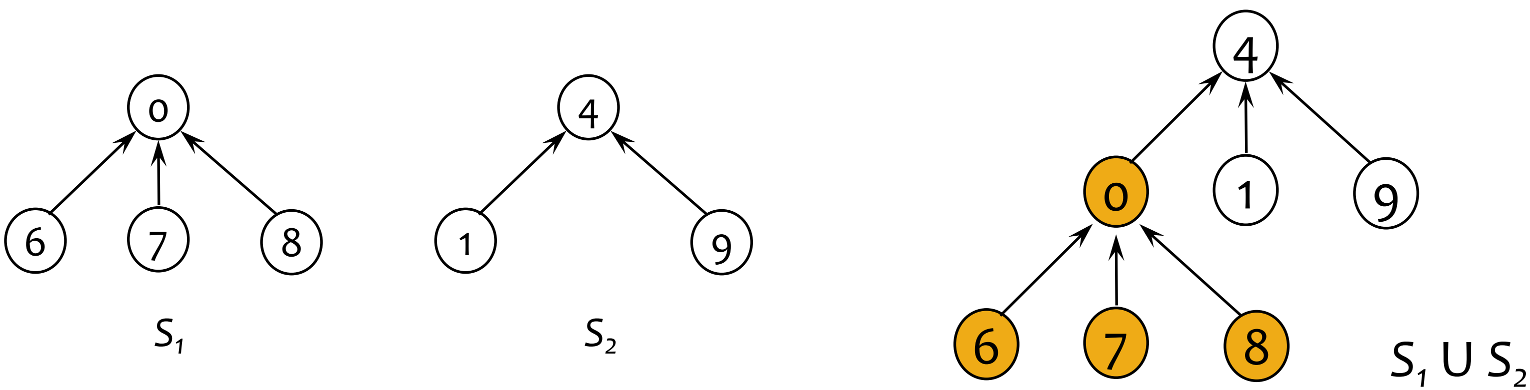
Union-Find Operations

- **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 U B, destroying the old A and B
- **find(p)**: Return the set containing the element at position p

Simple Union

- Merging two disjoint sets by setting a parent of one set as the other set

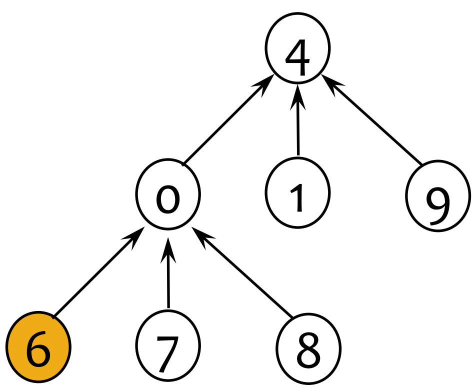
```
void Sets::SimpleUnion(int i, int j)
{
   parent[i] = j;
}
```



Simple Find

- Find the root of the tree containing i

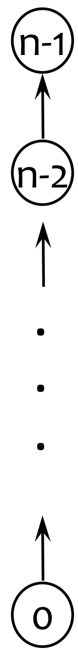
```
int Sets::SimpleFind(int i)
{
   while(parent[i] >= 0) i = parent[i]);
   return i;
}
```



Find 6 : 6->0->4

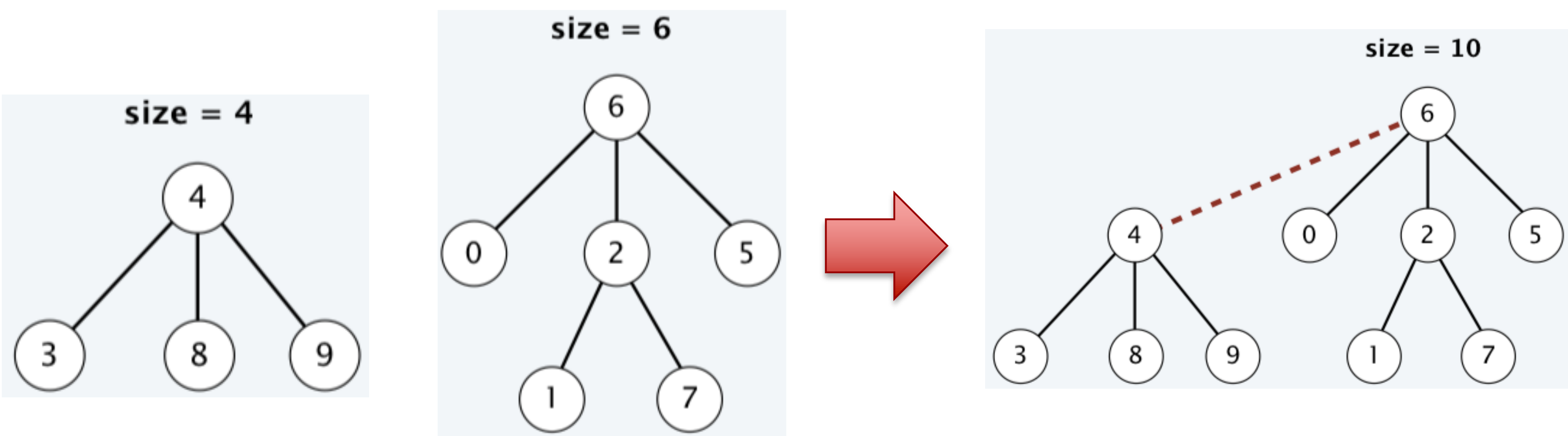
Degenerate Case

- union(0,1), union(1,2),... makes a degenerate tree
  - O(n^2) for find(0), find(1), ..., find(n-1)



Union-by-Size (i.e., Weighted Union)

- Maintain a subtree count for each node
- Initially, the count is 1
- Link the root of the smaller tree to the root of the larger tree (breaking ties arbitrarily).
- E.g., union(7, 3)

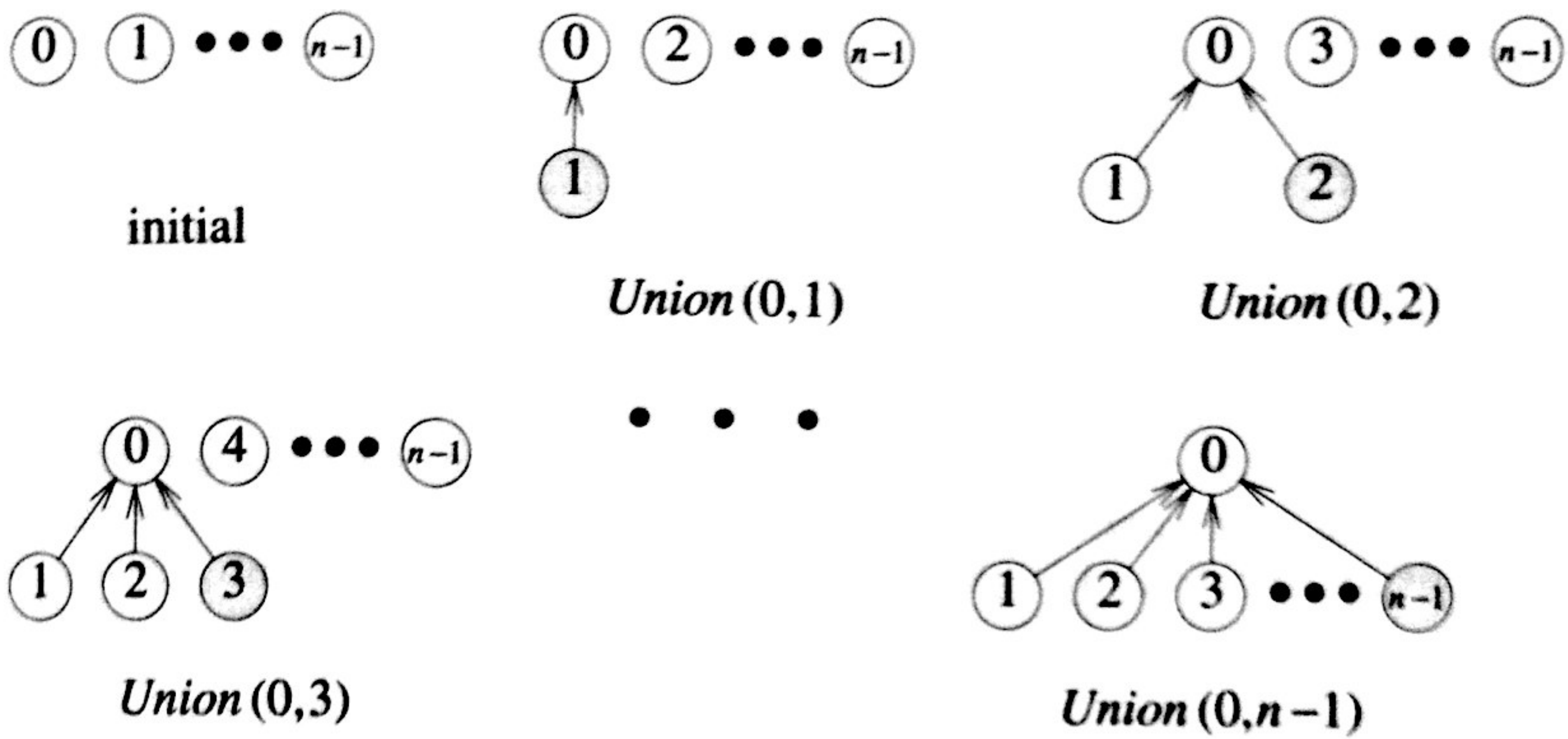


Union-by-Size (i.e., Weighted Union)

- Attach small tree to large tree

```
int Sets::WeightedUnion(int i, int j)
// parent[i] = -count[i], parent[j] = -count[j]
{
   int temp = parent[i] + parent[j];
   if(parent[i] > parent[j]) { i has fewer nodes
      parent[i] = j; // j is the root of united set
      parent[j] = temp;
   }
   else {
      parent[j] = i;
      parent[i] = temp;
   }
}
```

Union-by-Size (i.e., Weighted Union)

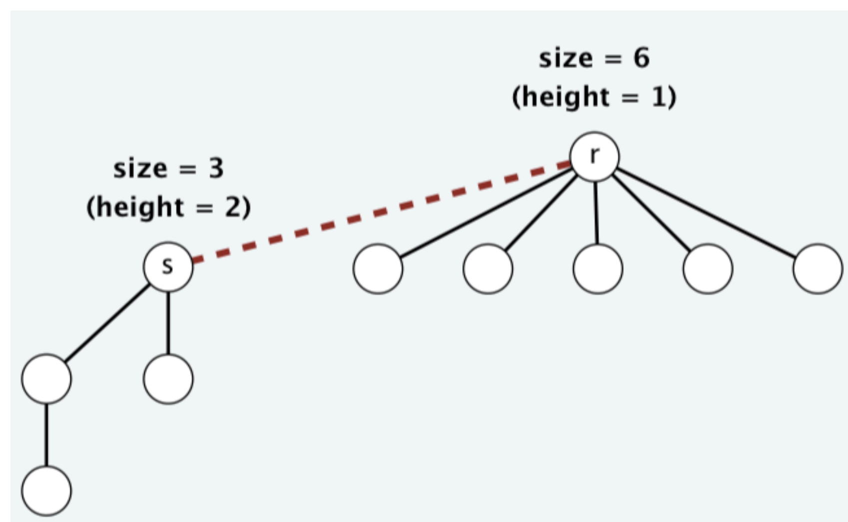


Find O(?)

Analysis of Union-by-Size

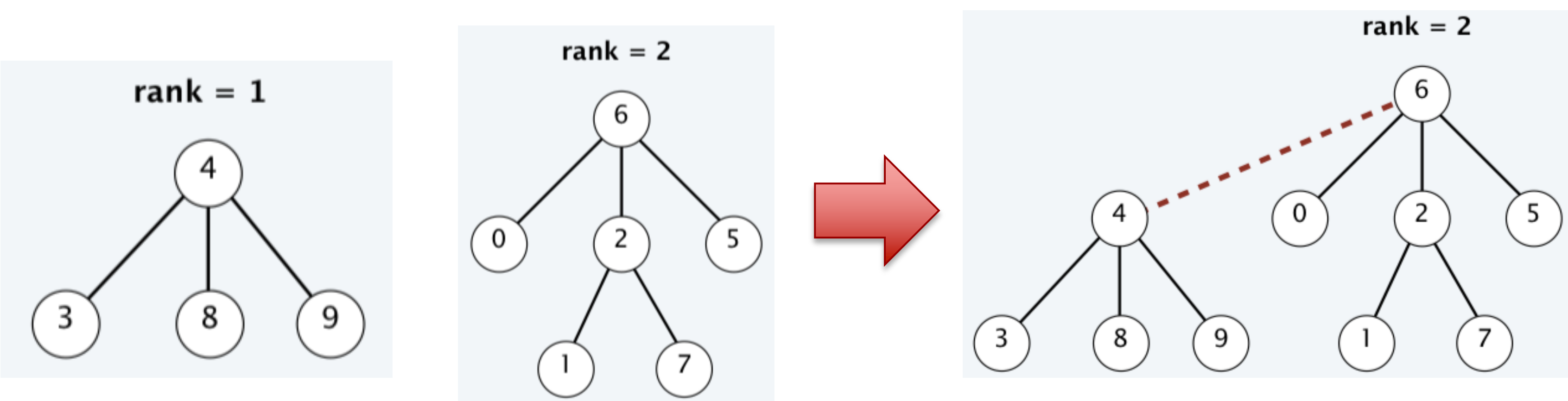
- Theorem: n = size(r) >= $2^{\text{height}(r)}$ for every root node r
- Proof:
  - Base case: when size(r) = 1, height(r) = 0
  - Assume true after adding i links
  - When height(r) <= height(s),
    size'(r) = size(r) + size(s) >= 2 size(s)
    >= 2 x $2^{\text{height}(s)}$ = $2^{\text{height}(s)+1}$ = $2^{\text{height}'(r)}$

- Corollary: Any Union and Find Operations takes O(log n) time in the worst case



Union-by-Rank

- Maintain an integer called *rank* for each node.
- Initially, the rank is 0.
- Link the root of the smaller rank to root of larger rank (if tie, increase rank of the new root by 1).
- Rank = height (before using path compression)
- E.g., union(7, 3)



Union-by-Rank

MAKE-SET (x)

$parent(x) \leftarrow x.$

$rank(x) \leftarrow 0.$

FIND (x)

WHILE $x \neq parent(x)$

  $x \leftarrow parent(x).$

RETURN x.

UNION-BY-RANK (x, y)

$r \leftarrow$ FIND (x).

$s \leftarrow$ FIND (y).

IF $(r = s)$ RETURN.

ELSE IF $rank(r) > rank(s)$

  $parent(s) \leftarrow r.$

ELSE IF $rank(r) < rank(s)$

  $parent(r) \leftarrow s.$

ELSE

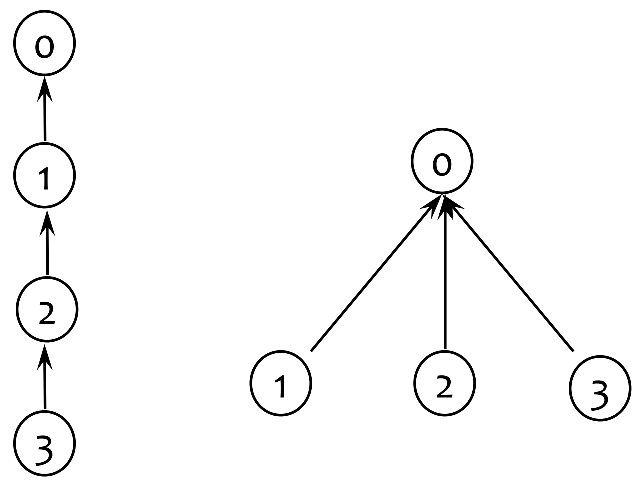
  $parent(r) \leftarrow s.$

  $rank(s) \leftarrow rank(s) + 1.$

- Theorem: Any Union and Find Operations takes O(log n) time in the worst case

Path Compression

- If there is a node j between i and root(i), set parent(j) = root(i)
  - O(1) find after collapsing



$\text{FIND}\ (x)$

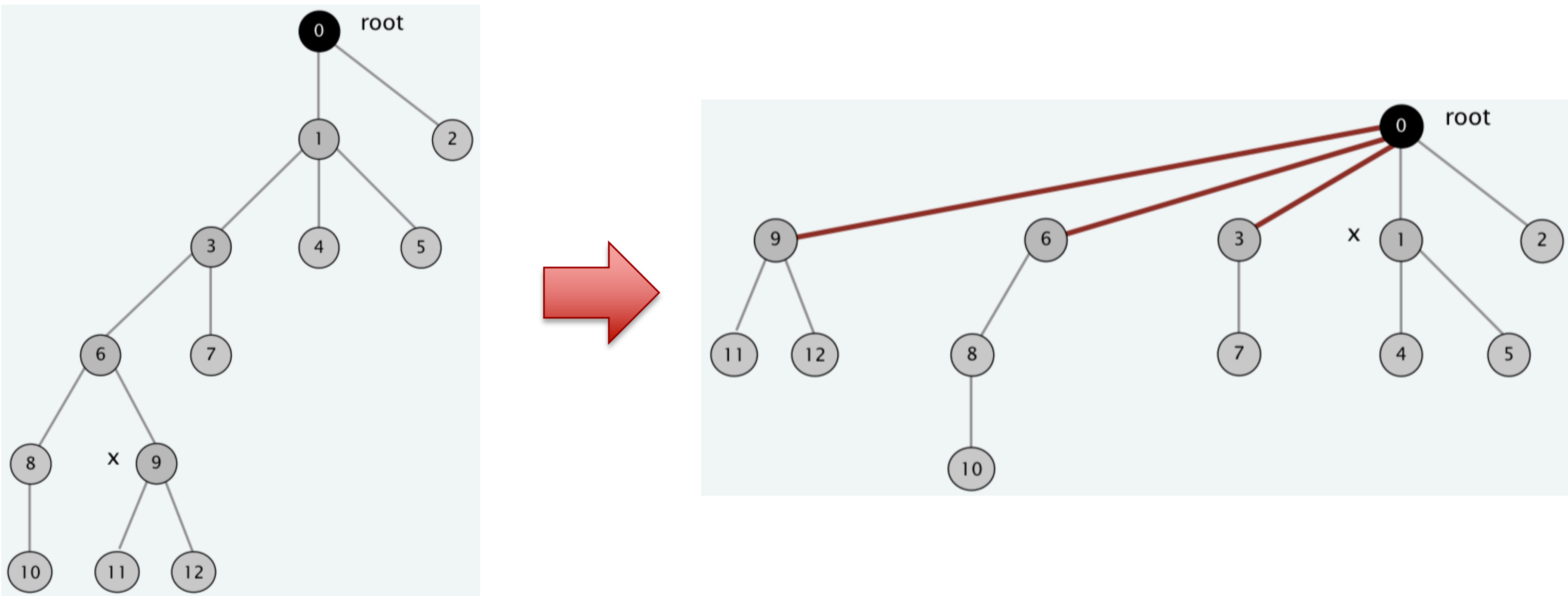
$\text{IF}\ x \neq parent(x)$

$\quad parent(x) \leftarrow \text{FIND}\ (parent(x)).$

$\text{RETURN}\ parent(x).$

Path Compression

- Find(x)



Analysis of Disjoint Set Union Algorithms

- By using Union-by-Size / Union-by-Rank with path compression, the amortized time per operation is $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function:

$$\alpha_k(n) = \begin{cases} 1 & \text{if } n = 1 \\ \lceil n/2 \rceil & \text{if } k = 1 \\ 1 + \alpha_k(\alpha_{k-1}(n)) & \text{otherwise} \end{cases}$$

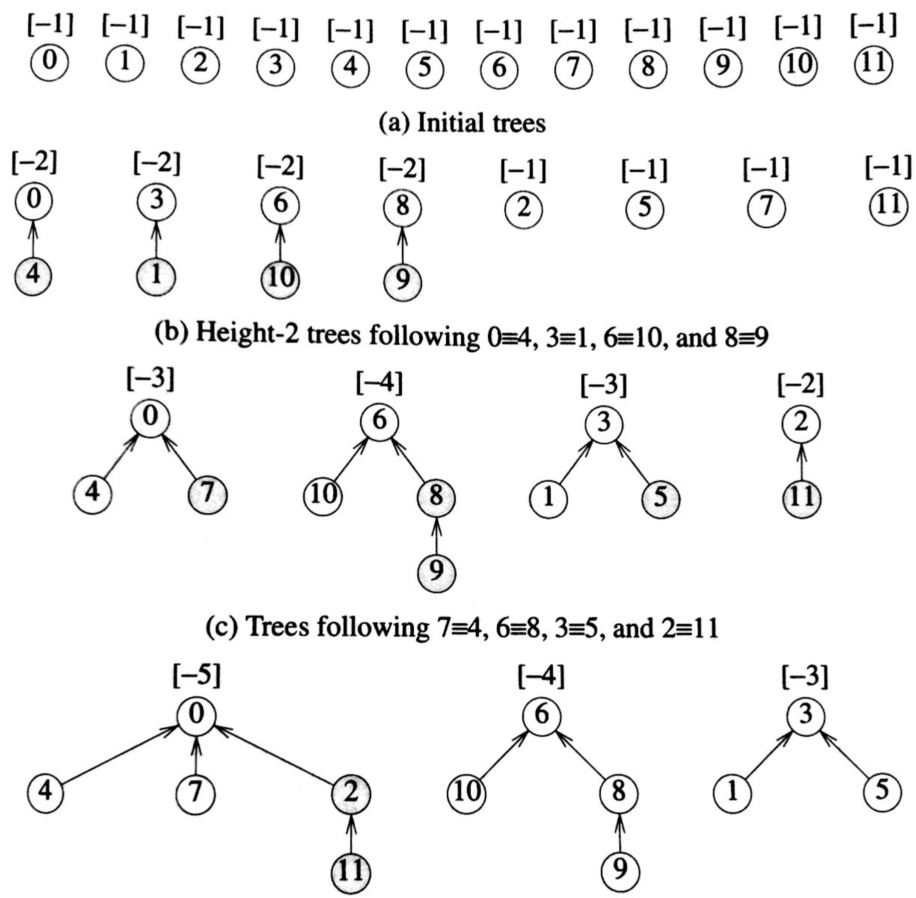
$$\alpha\,(n) = min\,\{\,k : \ \alpha_k(n) \leq 3\,\}$$

$$2 \uparrow 65536 \;=\; \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{65536 \text{ times}}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	2^{16}	...	2^{65536}	...	$2 \uparrow 65536$
$\alpha_1(n)$	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	...	2^{15}	...	2^{65535}	...	*huge*
$\alpha_2(n)$	1	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4	...	16	...	65536	...	$2 \uparrow 65535$
$\alpha_3(n)$	1	1	2	2	3	3	3	3	3	3	3	3	3	3	3	3	...	4	...	5	...	65536
$\alpha_4(n)$	1	1	2	2	3	3	3	3	3	3	3	3	3	3	3	3	...	3	...	3	...	4

Equivalent Classes

- For each pair (i,j): if Find(i) != Find(j), then Union(Find(i),Find(j))



(a) Initial trees

(b) Height-2 trees following 0≡4, 3≡1, 6≡10, and 8≡9

(c) Trees following 7≡4, 6≡8, 3≡5, and 2≡11

(d) Trees following 11≡0

Find(x) returns root of the tree containing x

Questions?