COT 6405: Analysis of Algorithms

Giri NARASIMHAN

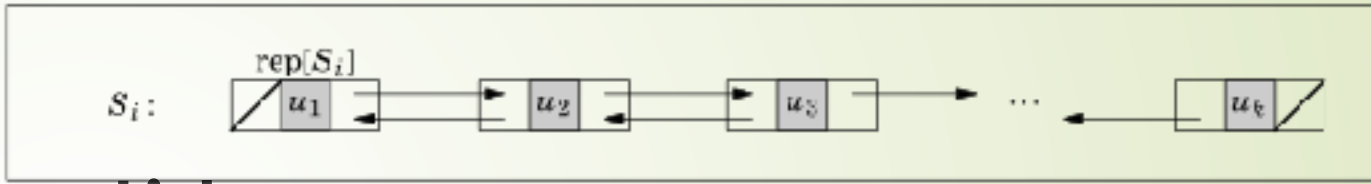
www.cs.fiu.edu/~giri/teach/6405F19.html

Heap Operations & Complexities

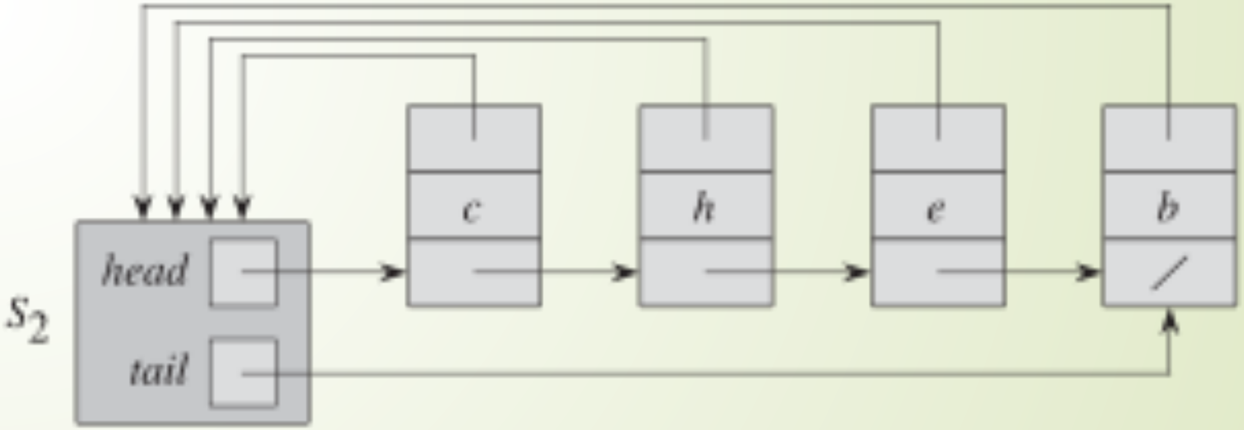
Operation	LinkedList	Binary Heap	Binomial Heap	Fibonacci Heap
MakeHeap	1	1	1	1
Insert	1	Log n	Log n	1
findMin	n	1	Log n	1
deleteMin	n	Log n	Log n	Log n
decreaseKey	1	Log n	Log n	1
Delete	n	Log n	Log n	Log n
Union /Merge	1	n	Log n	1

Static Sets

c	h	e	b



- Search: Is x in S?
  - Implement set as a List
    - Array
    - Linked List
    - Bit Maps



0	1	1	0	1	0	0	1

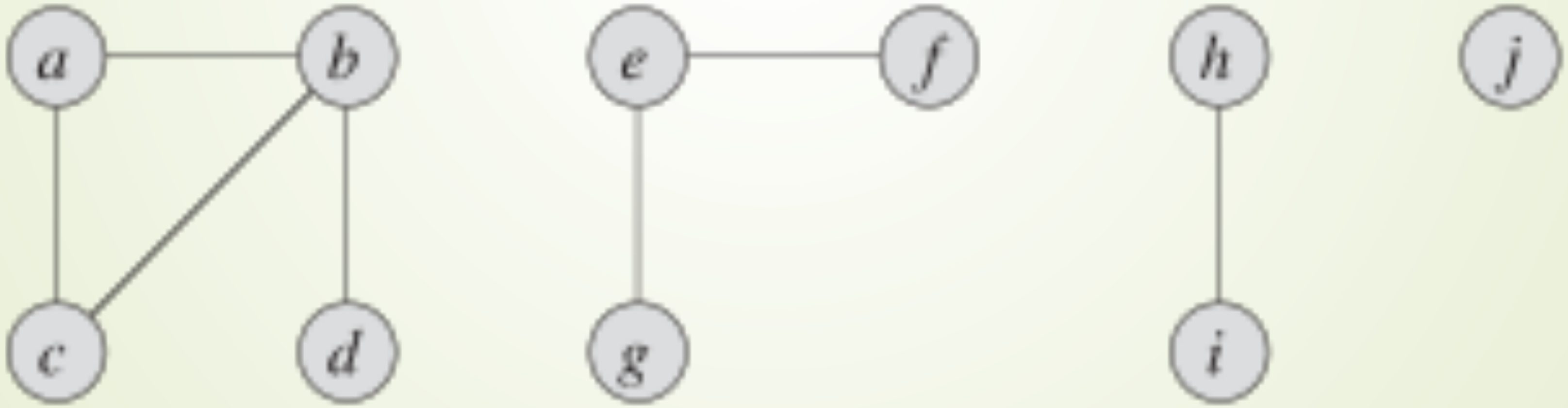
Dynamic Set operations

Need to maintain a collection of dynamic Sets

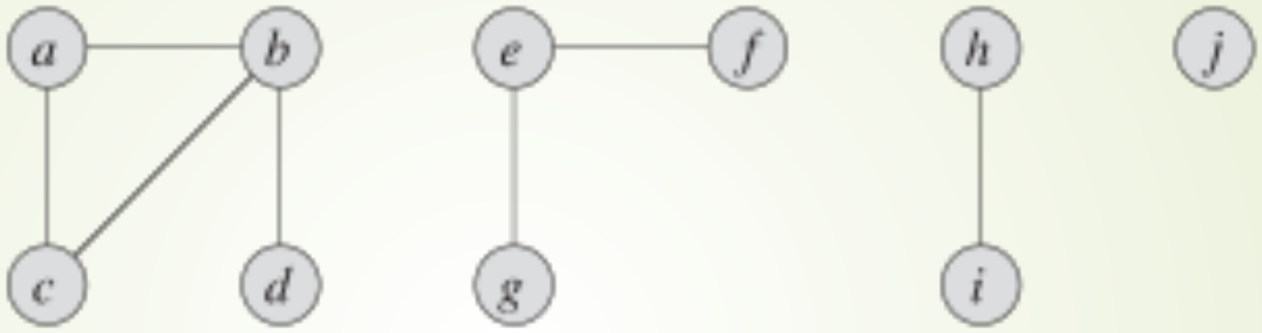
- SEARCH: Find-Set(u)
- INSERT: Union(u,v)
- Make-Set(u)
- Implementations
  - Lists, BitMaps, …

Connected Components

- Given a graph, compute all connected components
  - DFS or BFS   O(m+n) time



Connected Components w/ Sets



Edge processed	Collection of disjoint sets									
initial sets	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(b,d)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e\}$	$\{f\}$	$\{g\}$	$\{h\}$	$\{i\}$	$\{j\}$
(e,g)	$\{a\}$	$\{b,d\}$	$\{c\}$		$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(a,c)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h\}$	$\{i\}$	$\{j\}$
(h,i)	$\{a,c\}$	$\{b,d\}$			$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(a,b)	$\{a,b,c,d\}$				$\{e,g\}$	$\{f\}$		$\{h,i\}$		$\{j\}$
(e,f)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$
(b,c)	$\{a,b,c,d\}$				$\{e,f,g\}$			$\{h,i\}$		$\{j\}$

Connected Components w/ Sets

```
CONNECTED-COMPONENTS(G)
for each vertex   ∈ G.V
    MAKE-SET( )
for each edge (u,  ) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET( )
        UNION(u,  )
```

```
SAME-COMPONENT(u,  )
if FIND-SET(u) == FIND-SET( )
    return TRUE
else return FALSE
```

Dynamic Set operations

Need to maintain a collection of dynamic Sets

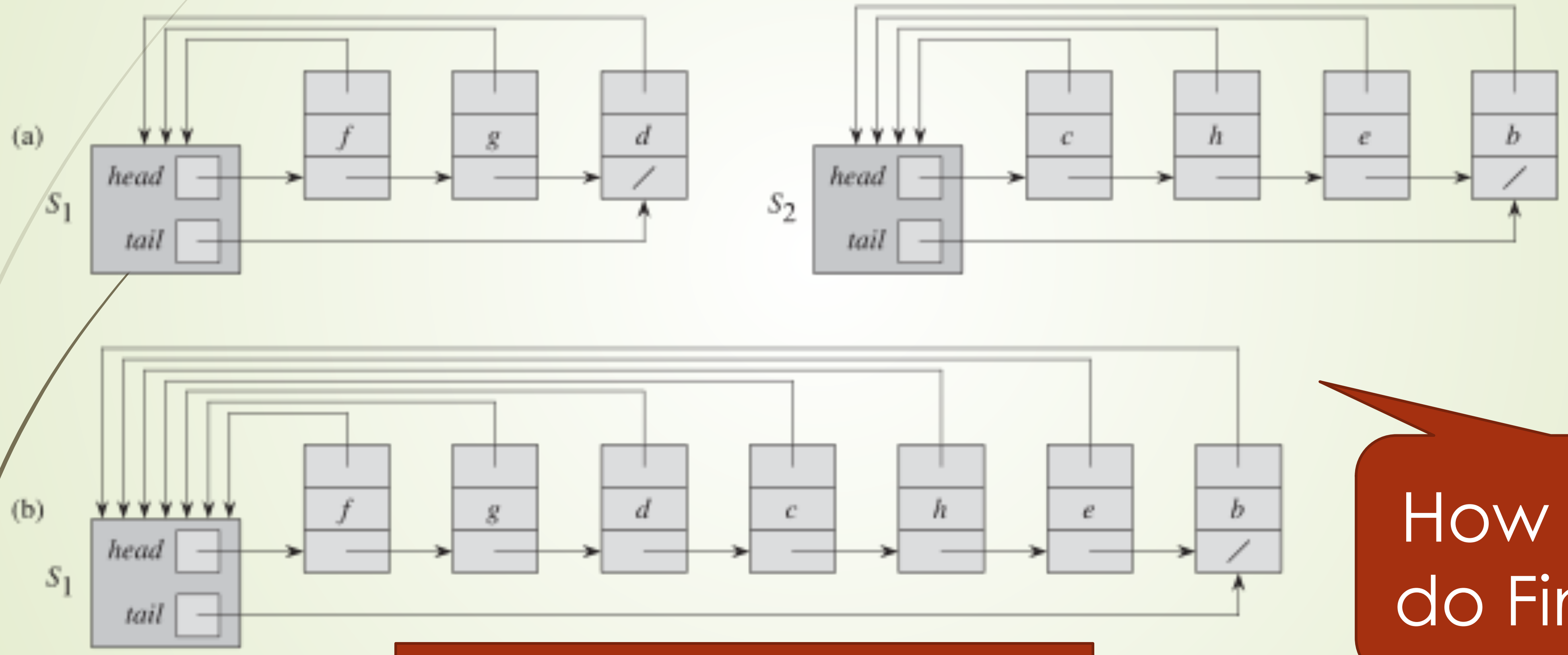
- SEARCH: Find-Set(u)
- INSERT: Union(u,v)
- Make-Set(u)
- Implementations
  - Lists, BitMaps, …

Enough to deal with Disjoint Sets

Implementation Challenges

- Array Implementations
  - Series of unions become very expensive
    - $O(n^2)$
- Linked List Implementations
  - Series of unions become very expensive $O(n^2)$
- Bit Implementations
  - Can be $O(n^2)$ bit operations

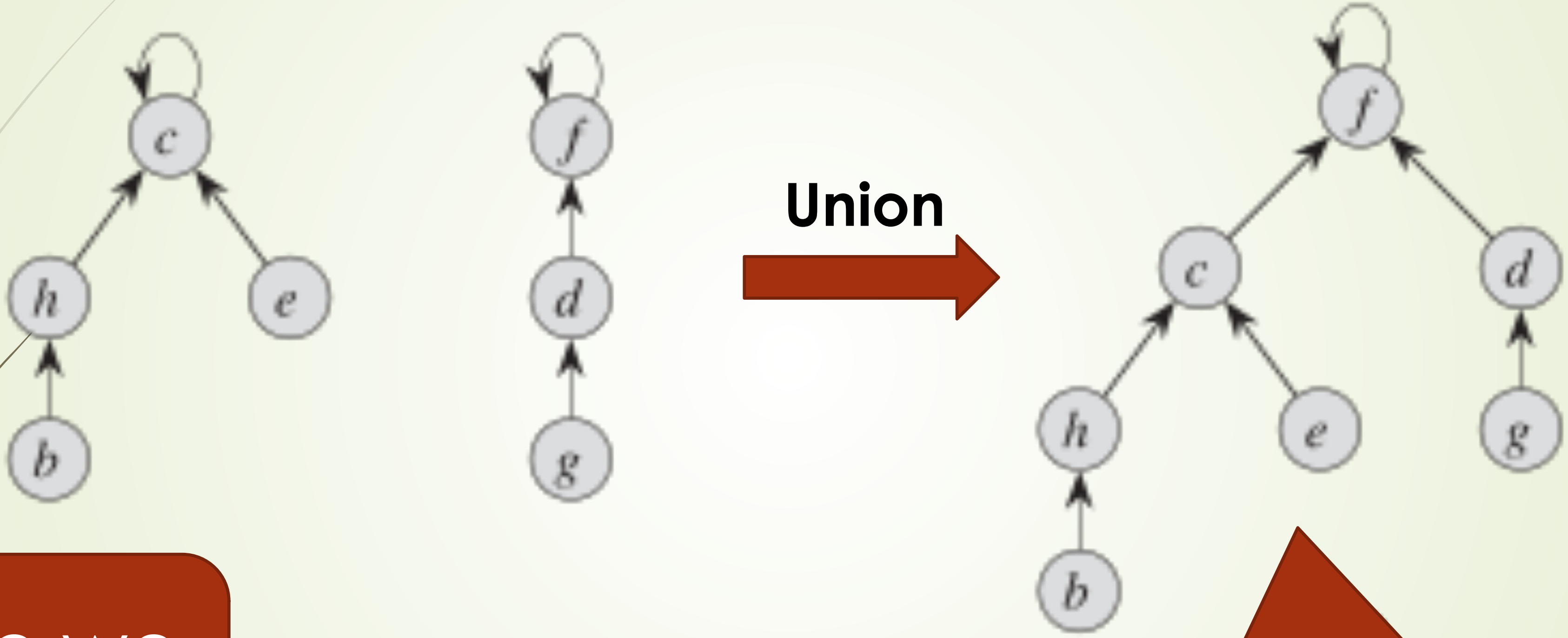
Union Operation



How do we do FindSet?

FindSet is O(1) time.
Union is O(n) time.

Forest of Trees Implementation



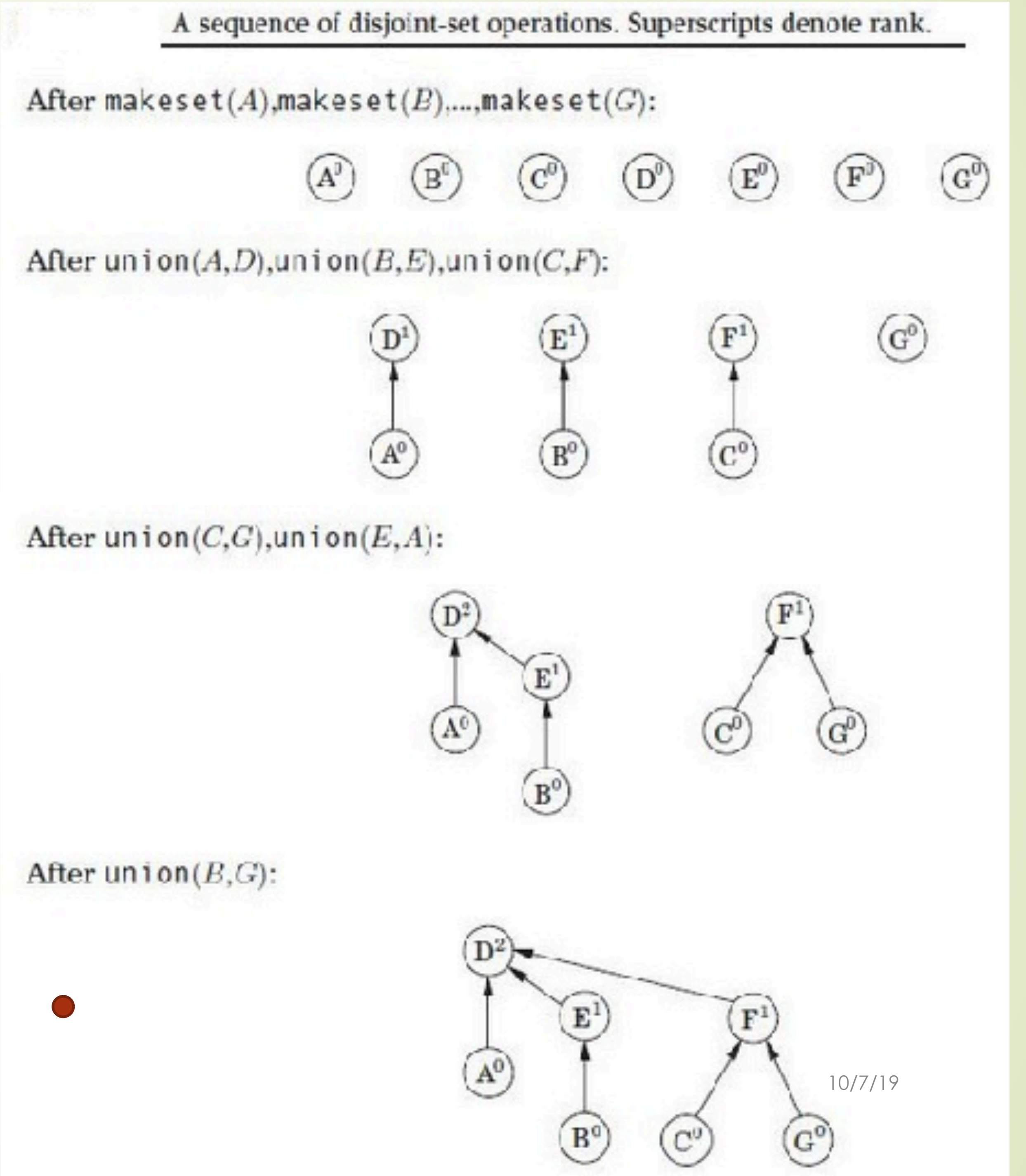
How do we do FindSet?

- FindSet is O(h) time.
- Union is O(1) time.
- Union is O(1) + 2h time

Forest of trees Implementation

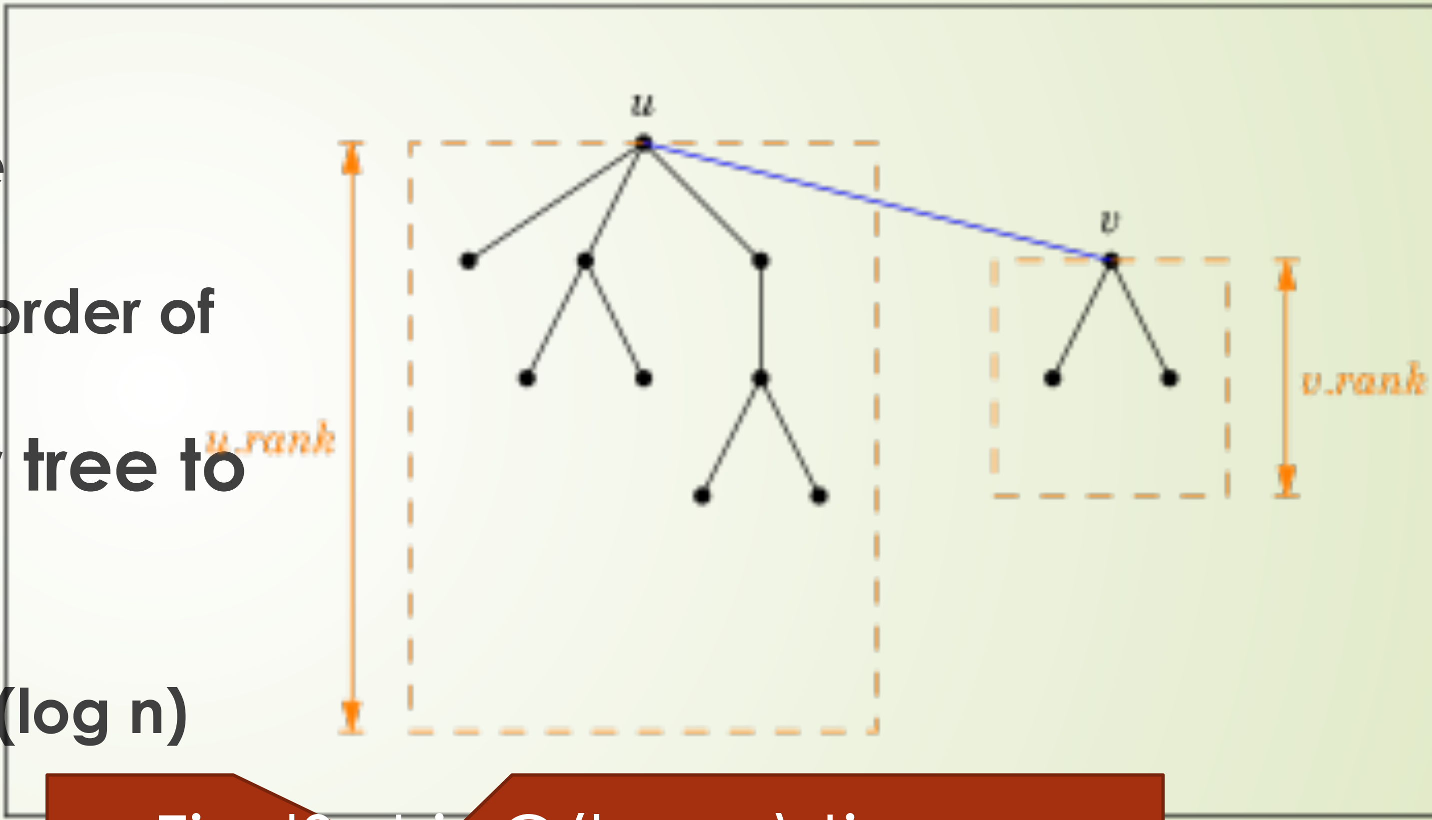
Another example

Trees in forest need not be binary trees



Even better Union operations

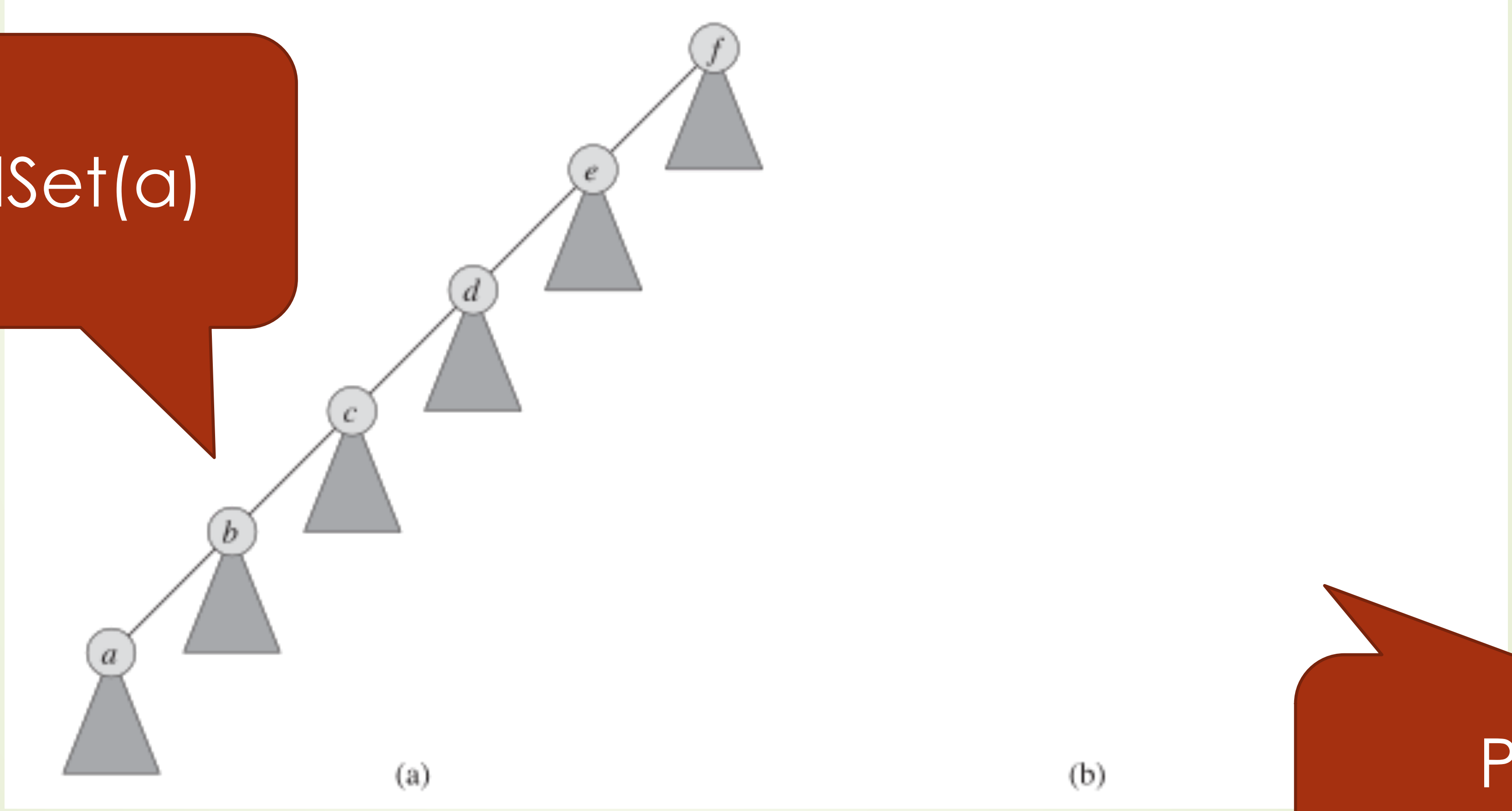
- Problem with FindSet
  - Height determines time complexity
  - Height determined by order of operations
- Always attach smaller tree to larger tree
  - Why is this better?
  - Guarantees height = O(log n)



- FindSet is O(log n) time.
- Union is O(1) time.
- Union is O(log n) time

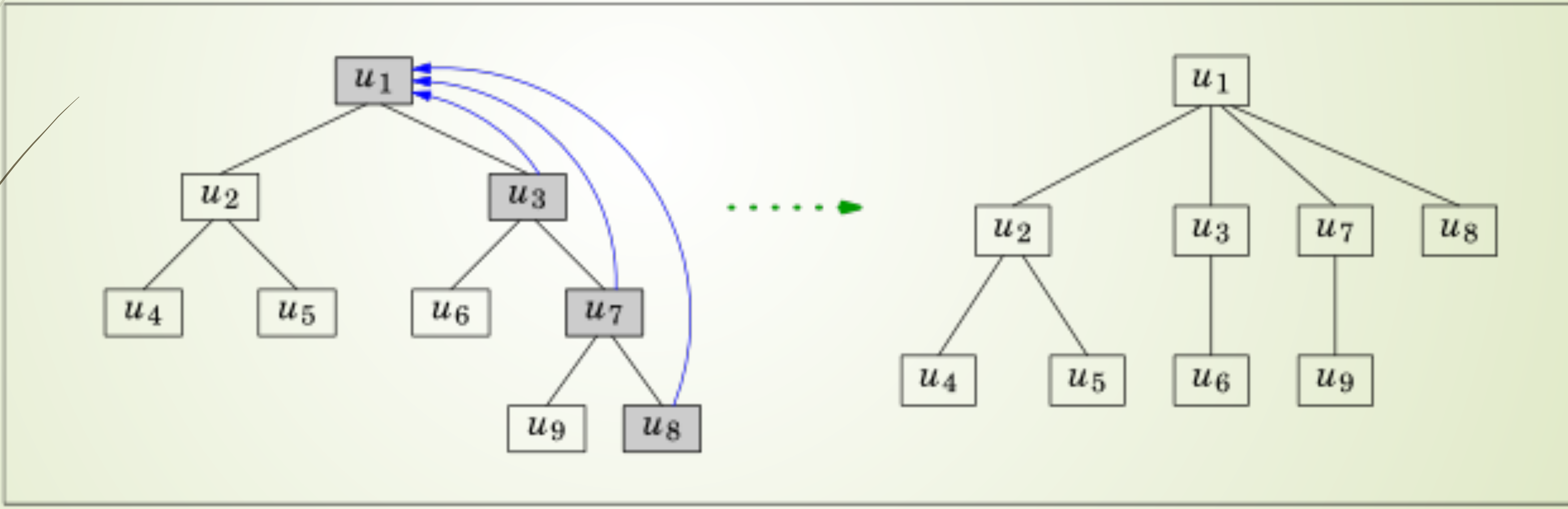
Even better FindSet operation

FindSet(a)



Path Compression

Example of Path Compression



UnionFind Data Structure

```
MAKE-SET(x)
    x.p = x
    x.rank = 0

FIND-SET(x)
    if x ≠ x.p
        x.p = FIND-SET(x.p)
    return x.p

UNION(x, y)
    LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
    if x.rank > y.rank
        y.p = x
    else x.p = y
        if x.rank == y.rank
            y.rank = y.rank + 1
```

Union-FInd w/ Path Compression

- Given m operations on n elements,
  - Time Complexity = O(m α(n))

For integers k 0 and j 1, we define the function A_k . j/ as

$$A_k . j/ \; D \begin{cases} j \; C \; 1 & \text{if } k \; D \; 0 \; ; \\ A_{k \square 1}^{. jC1/} . j/ & \text{if } k \quad 1 \; ; \end{cases}$$

$$\boxed{\alpha(n) = \min\{k : A_k(1) \quad n\}}$$

n	$A_n(1)$
0	2
1	3
2	7
3	2047
4	10^{80}
5	…

n	α(n)
2	0
3	1
7	2
2047	3
10^{80}	4

More about Ackermann function

$$A_1(j) = 2j + 1$$

$$A_2(j) = 2^{j+1}(j+1)$$