Data Structures
Disjoint Sets 2

CS 225 October 18, 2023

Brad Solomon & G Carl Evans

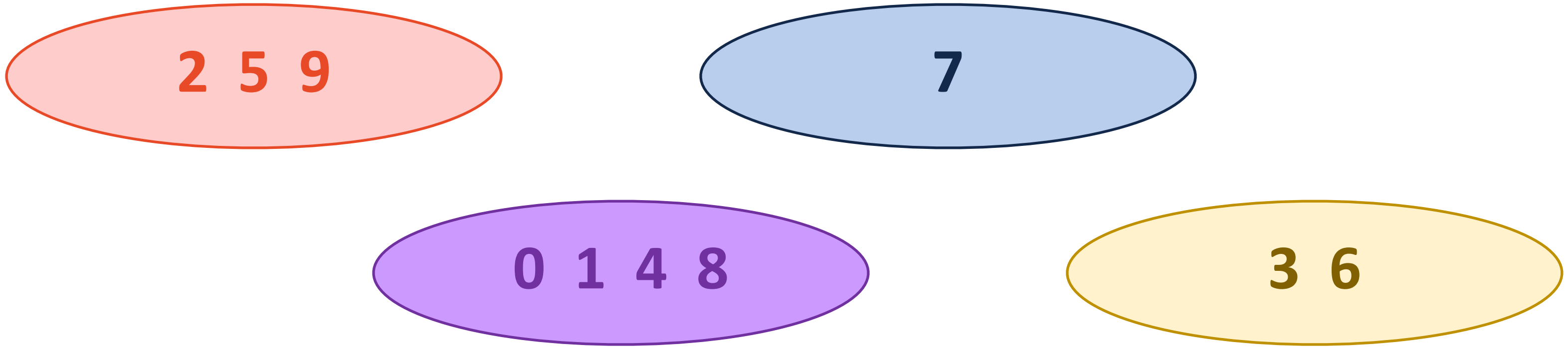
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Finish disjoint set implementation

Discuss efficiency of disjoint sets

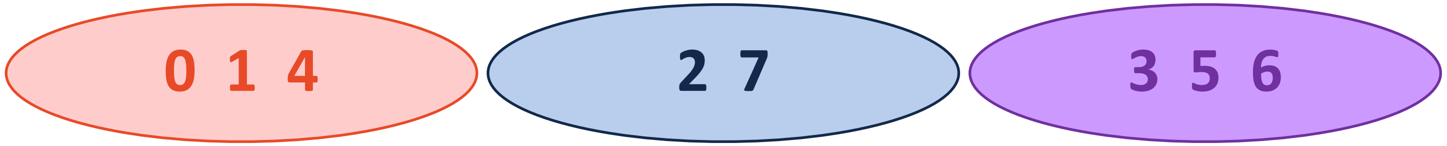
Disjoint Sets



Key Ideas:

- Each element exists in exactly one set.
- Every item in each set has the same representation
- Each set has a different representation

Implementation #2

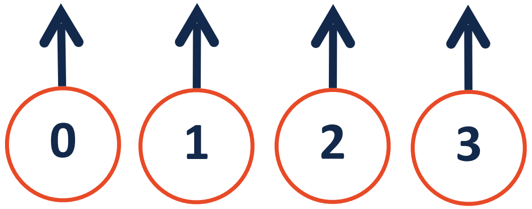


0	1	2	3	4	5	6	7

Find(k):

Union(k_1, k_2):

UpTrees



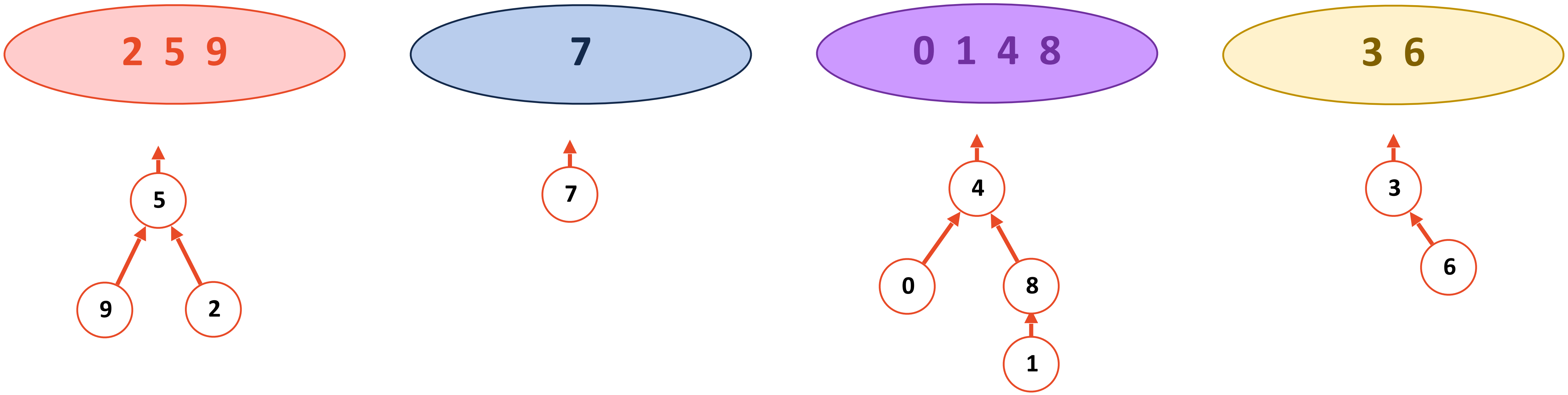
0	1	2	3

0	1	2	3

0	1	2	3

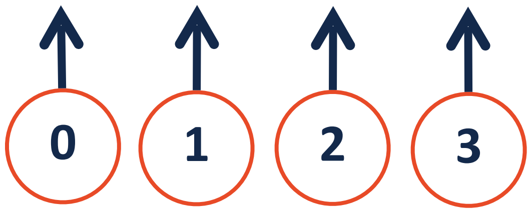
0	1	2	3

Disjoint Sets



0	1	2	3	4	5	6	7	8	9

UpTrees: Worst Case



0	1	2	3

0	1	2	3

0	1	2	3

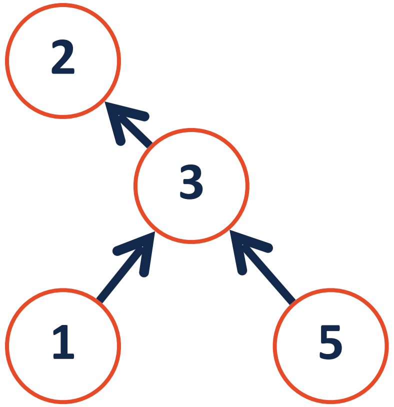
0	1	2	3

Disjoint Sets Representation

We can represent a disjoint set as an array where the key is the index

The values inside the array stores our sets as a pseudo-tree (UpTree)

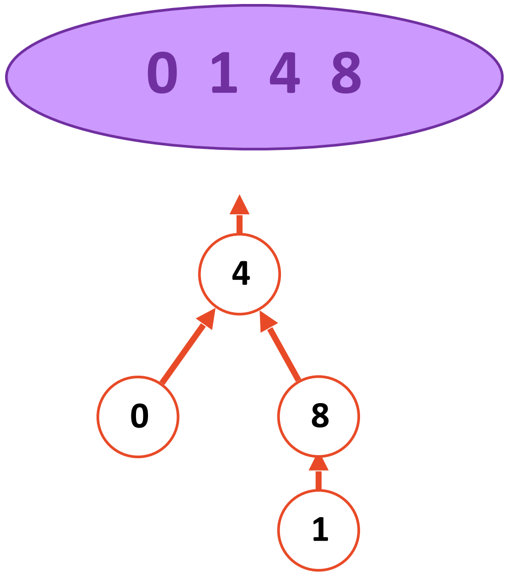
- The value **-1** is our representative element (the root)
- All other set members store the index to a parent of the UpTree



Disjoint Sets Find

Find(1)

```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return find( s[i] ); }
}
```



Running time?

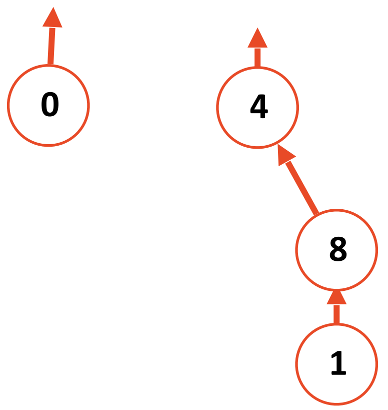
What is ideal UpTree?

0	1	2	3	4	5	6	7	8	9
4	8			-1				4	

Disjoint Sets Union

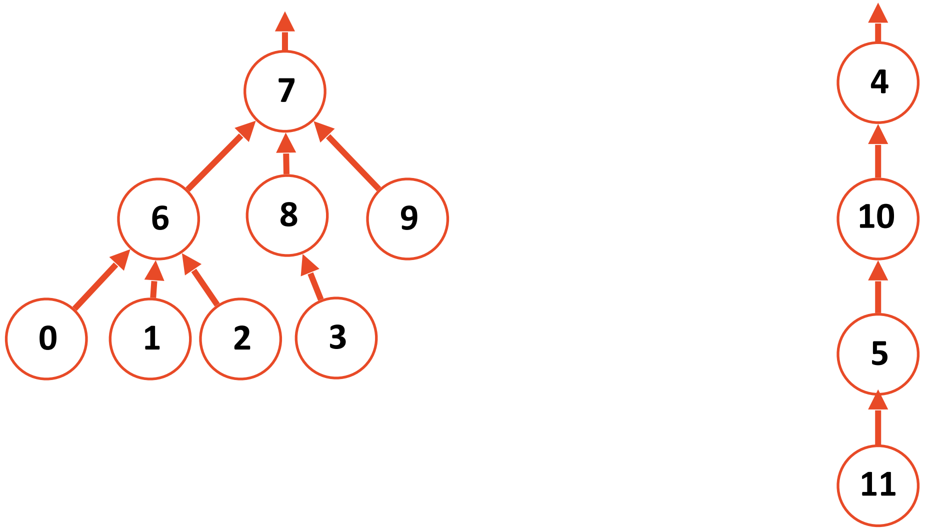
Union(0, 4)

```
int DisjointSets::union(int r1, int r2) {



}
```



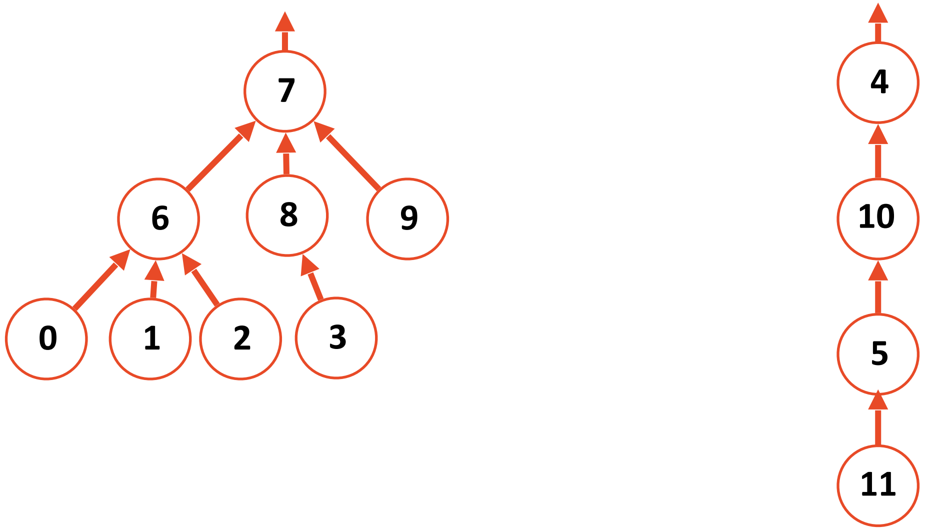
0	1	2	3	4	5	6	7	8	9
-1	8			-1				4	

Disjoint Sets – Union



0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	-1	10	7	-1	7	7	4	5

Disjoint Sets – Smart Union

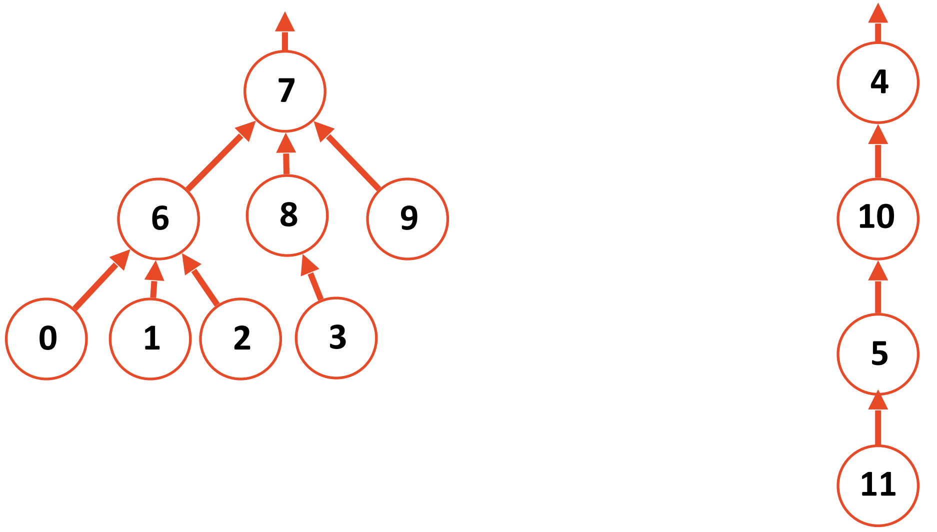


Union by height

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

***Idea**: Keep the height of the tree as small as possible.*

Disjoint Sets – Smart Union

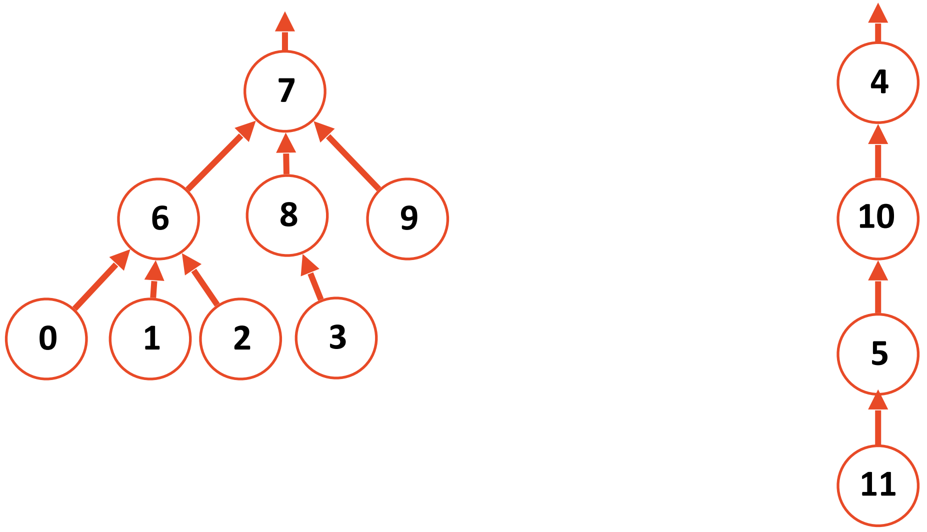


Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

Idea: *Minimize the number of nodes that increase in height*

Disjoint Sets – Smart Union



Union by height

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

Idea: *Keep the height of the tree as small as possible.*

Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8		10	7		7	7	4	5

Idea: *Minimize the number of nodes that increase in height*

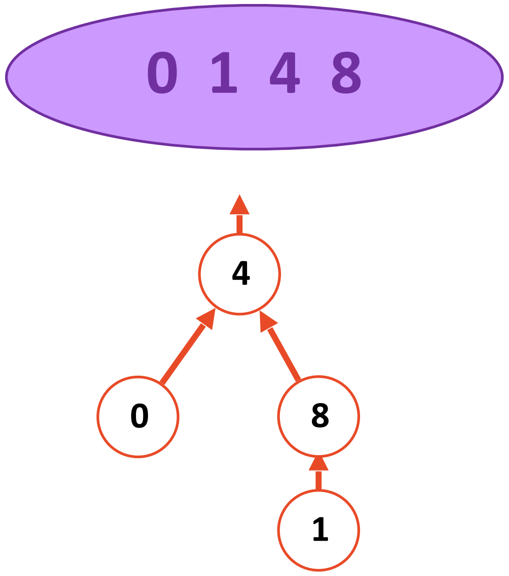
Claim that both guarantee the height of the tree is: _______________.

Disjoint Sets Find

Find(1)

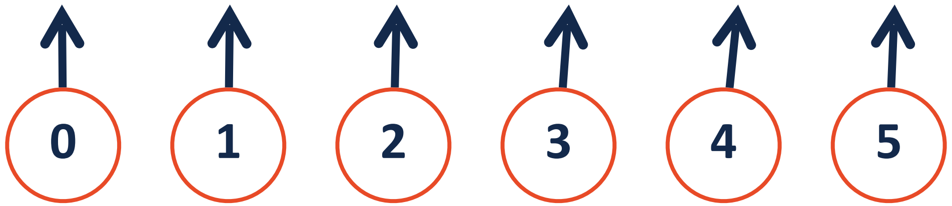
```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return find( s[i] ); }
}
```

Does our metadata change anything?



0	1	2	3	4	5	6	7	8	9
4	8			-3/-4				4	

Disjoint Sets Union Example



0	1	2	3	4	5
-1	-1	-1	-1	-1	-1

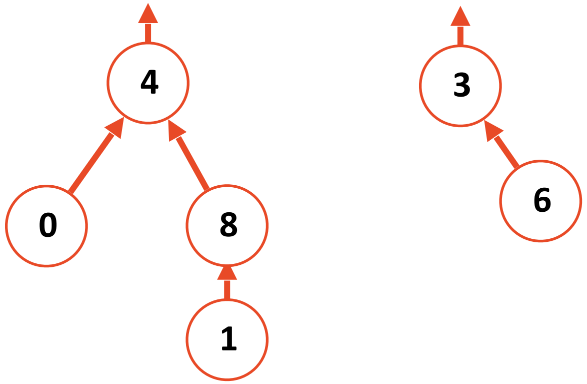
0	1	2	3	4	5

0	1	2	3	4	5

Disjoint Sets Union

`unionBySize(4, 3)`

```
void DisjointSets::unionBySize(int root1, int root2) {
  int newSize = arr_[root1] + arr_[root2];

  if ( arr_[root1] < arr_[root2] ) {

    arr_[root2] = root1;

    arr_[root1] = newSize;

  } else {

    arr_[root1] = root2;

    arr_[root2] = newSize;

  }
}
```



0	1	2	3	4	5	6	7	8	9
4	8		-2	-4		3		4	

Disjoint Sets Union by Size

Claim: Sets unioned by size have a height of at most $O(\log_2 n)$

Claim: An UpTree of height **h** has nodes $\geq$ ______________

Base Case:

Disjoint Sets Union by Size

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH:

Disjoint Sets Union by Size

n(B) $\geq$ n(A)

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 1: height(A) < height(B)

Disjoint Sets Union by Size

$n(B) \geq n(A)$

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 2: height(A) == height(B)

Disjoint Sets Union by Size

n(B) $\geq$ n(A)

Claim: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Case 3: height(A) > height(B)

Disjoint Sets Union by Size

$n(B) \geq n(A)$

Proven: An UpTree of height **h** has nodes $\geq 2^h$

IH: Claim is true for $< i$ unions, prove for ith union.

Each case we saw we have $n \geq 2^h$.

Disjoint Sets – Union by Rank

0	1	2	3	4	5	6	7	8	9

Union by Height (Rank)

Instead of using height, lets use rank.

The change: New UpTrees have rank = 0

Let A, B be two sets being unioned. If:

rank(A) == rank(B): The merged UpTree has rank + 1

rank(A) > rank(B): The merged UpTree has rank(A)

rank(B) > rank(A): The merged UpTree has rank(B)

This is identical to height (with a different starting base)!