Data Structures
Disjoint Sets 2

Announcements

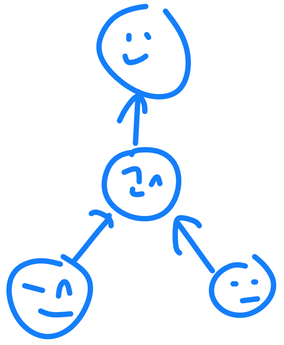
IEF → ~600
↳ ~70% → +5 points

Cheating (Mosaics)
↳ Way down mosaics!

CS 225

Brad Solomon & G Carl Evans

October 18, 2023



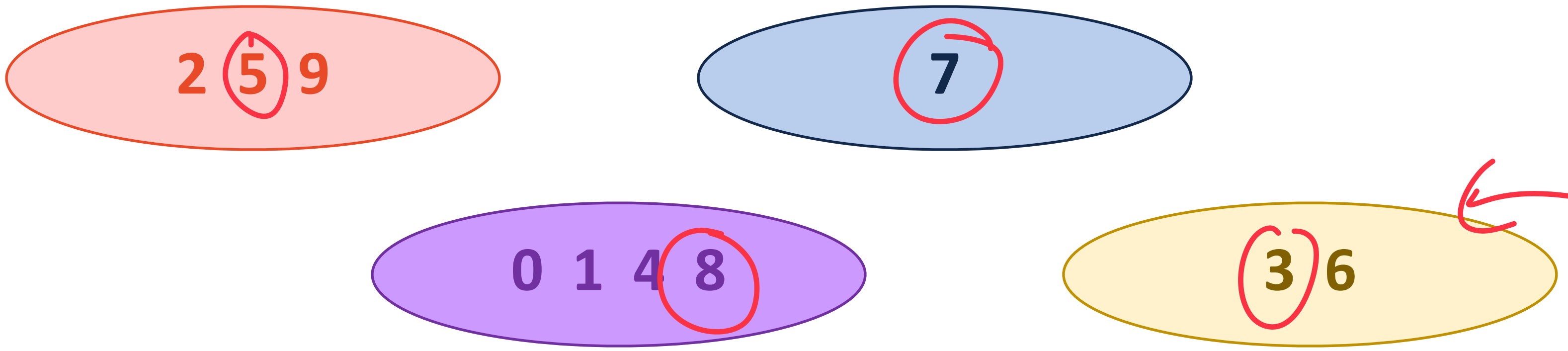
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

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

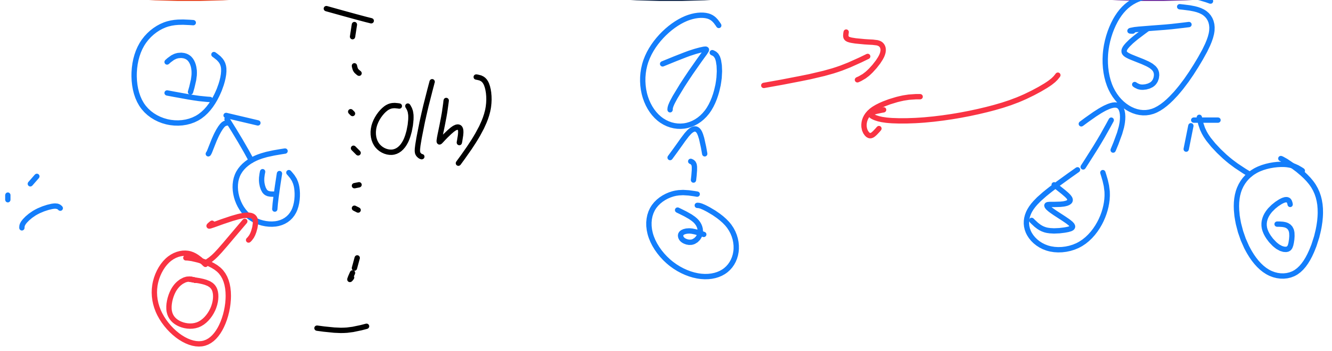
No duplicates
overlaps

canonical element

Implementation #2

union(5, 7)

0 1 4	2 7	3 5 6



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

Find(k): O(h) ⤷ A little slower

↳ last class: O(1) w/ array

Union(k₁, k₂): O(1) b/c we draw one arrow

(we change one value)

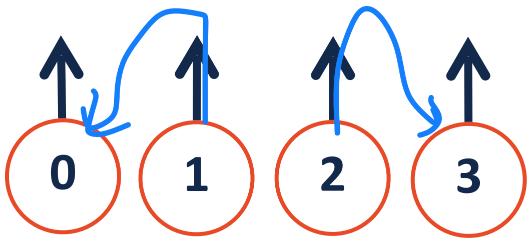
↳ last class: O(n)

up tree

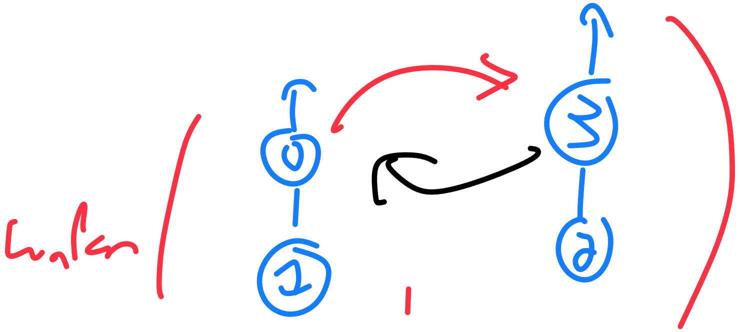
1) All non-canonical elements point to canonical (or 'root')

2) Canonical (root) elements store -1

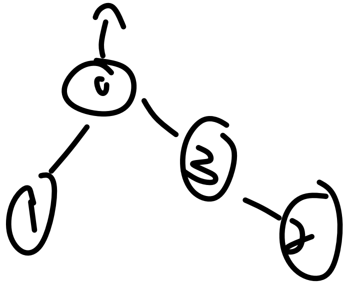
UpTrees



0	1	2	3
-1	-1	-1	-1



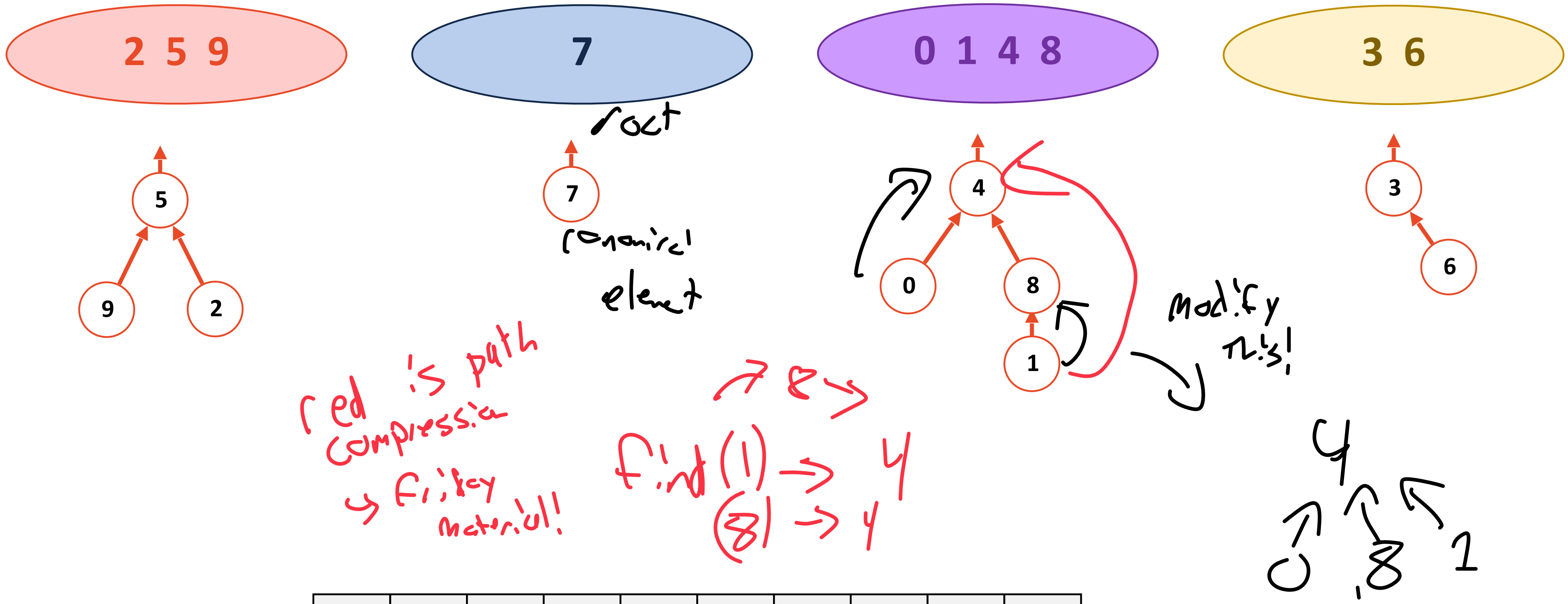
0	1	2	3
-1	0	3	-1



0	1	2	3
-1	0	3	0

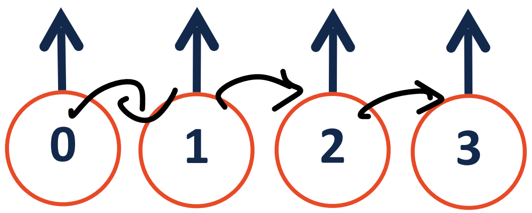
0	1	2	3

Disjoint Sets

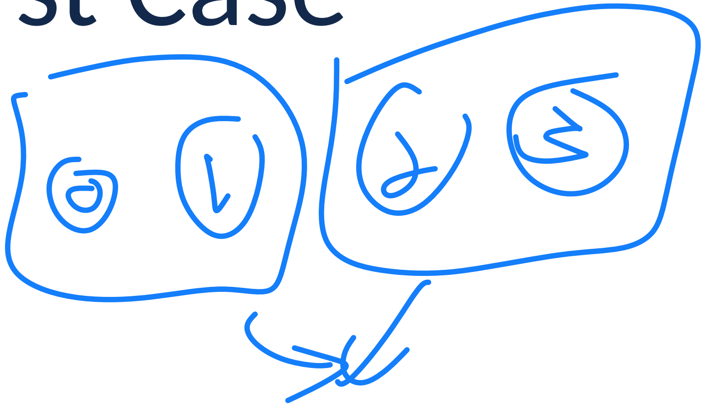


0	1	2	3	4	5	6	7	8	9
4	8	5	-1	-1	-1	3	-1	4	5

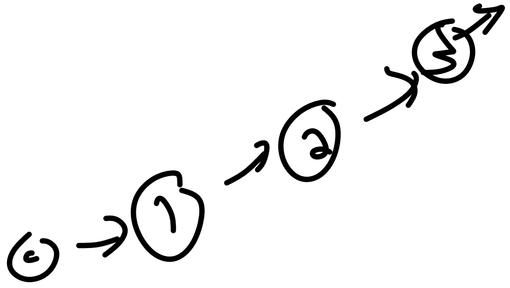
UpTrees: Worst Case



0	1	2	3

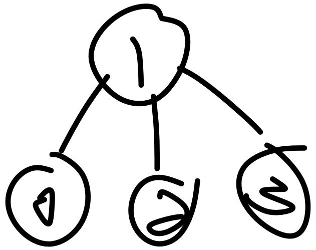


The user decides!



0	1	2	3
1	2	3	-1

Best case



0	1	2	3

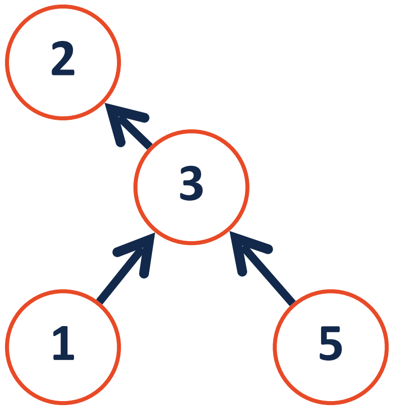
0	1	2	3
1	-1	1	1

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

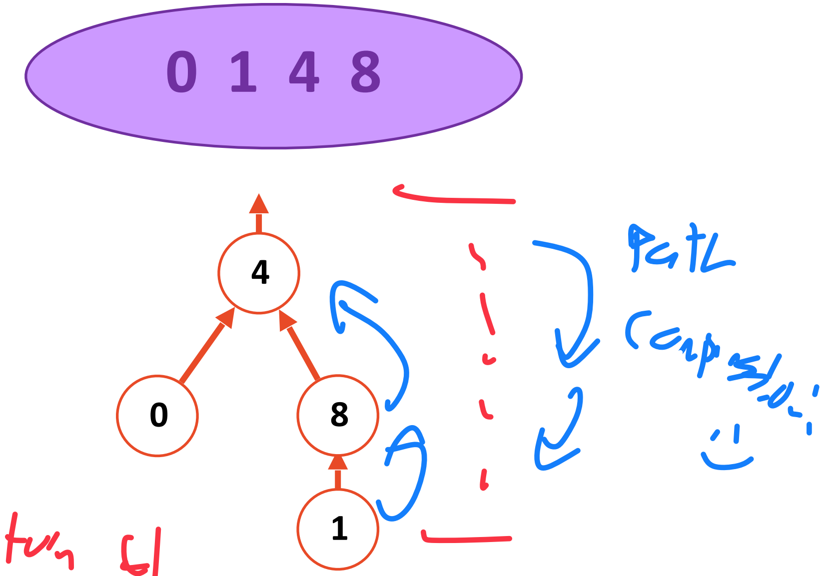
-1 's canonical

Find(1)

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

recurse up

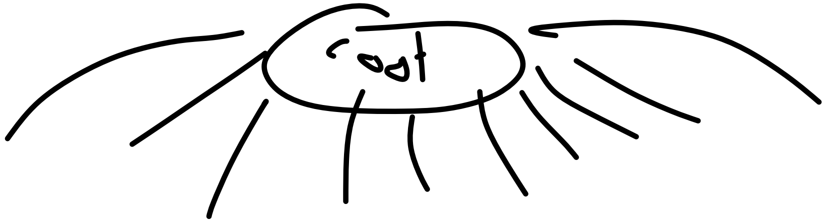
Running time? O(h)



return 4

What is ideal UpTree? height 1 or 0

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



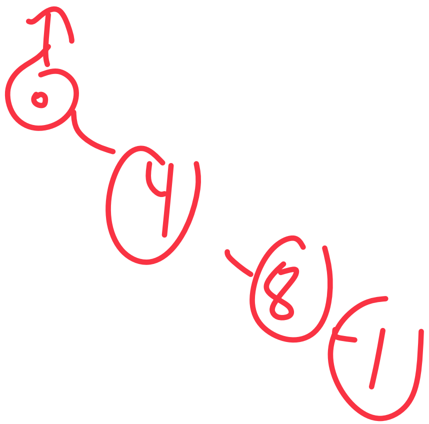
Disjoint Sets Union

Union(0, 4)

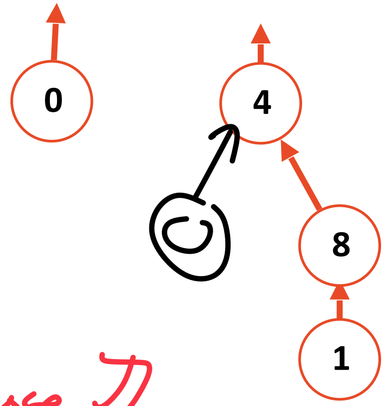
```
int DisjointSets::union(int r1, int r2) {

    s[r1] = r2;

}
```

↳ simple is easy!

But if I do union(4,0)



How do we choose??



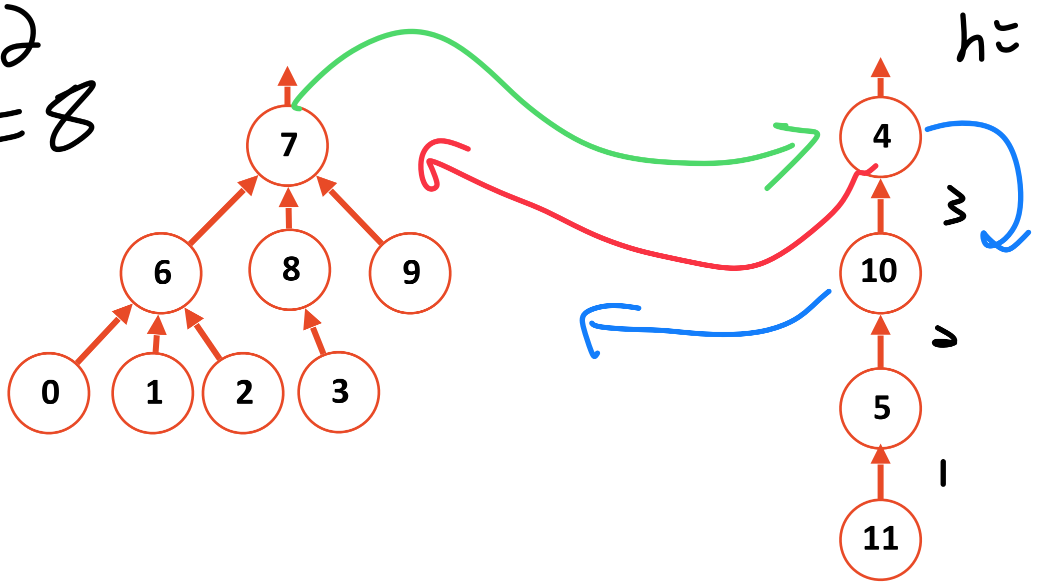
0	1	2	3	4	5	6	7	8	9
~~-1~~ 4	8			-1				4	

Disjoint Sets – Union

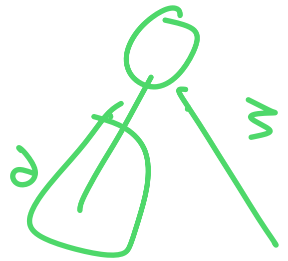
Have to store either size or height ↗

↳ store either height or size at canonical

h = 2
size = 8



h = 3
size = 4



Interesting idea! Break up tree to make it better!

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

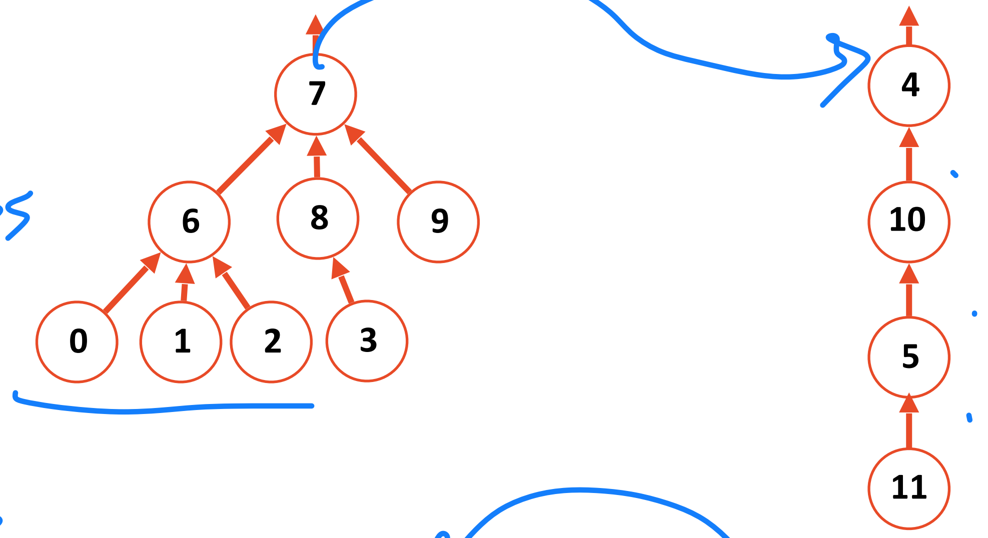
merge 7 to 4 – why? Merge smaller height to larger height (no height increase!)

Merge 4 to 7 – why? we increase the runtime for the fewest elements

Disjoint Sets – Smart Union

find(0) → used to be 4 ops

:'s now 3 ops



h=0

array: -1

stores -1

-height -1

Union by height

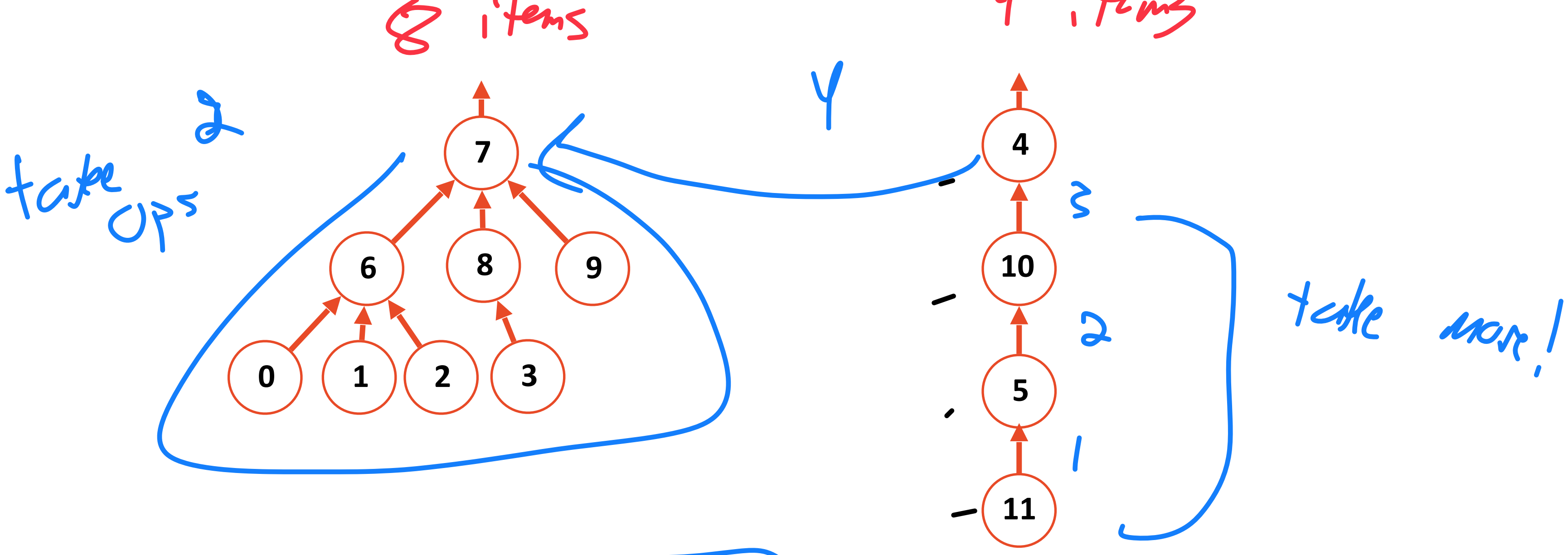
0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	-1	10	7	~~-1~~ 4	7	7	4	5

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

Store at canonical Not -1 but (-height -1)

Disjoint Sets – Smart Union

uptree has h = 4



Union by size

0	1	2	3	4	5	6	7	8	9	10	11
6	6	6	8	~~-4~~	10	7	~~[illegible]~~	7	7	4	5
				7			-12				

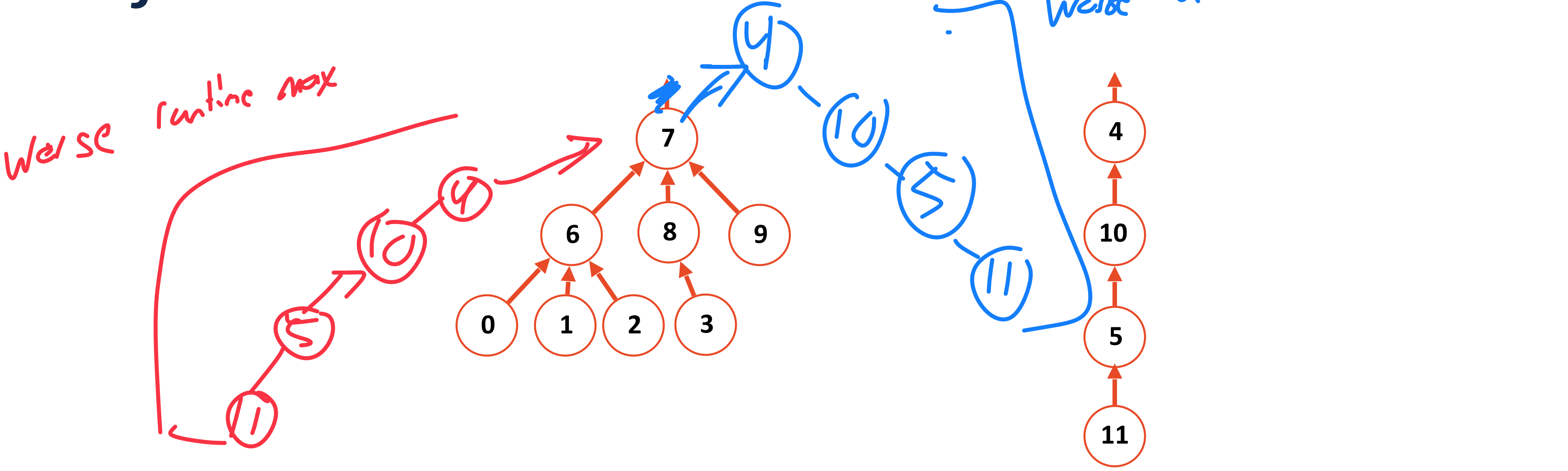
Idea: *Minimize the number of nodes that increase in height*

-4 & -8

-12 ← 4

size(7) = size(7) + size(4)

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

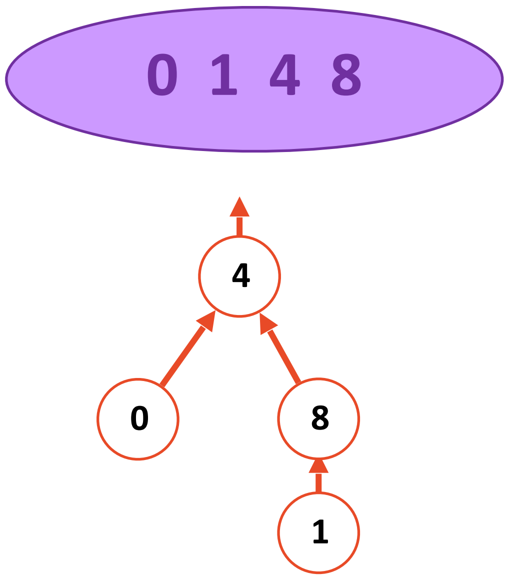
***Idea**: Minimize the number of nodes that increase in height*

Claim that both guarantee the height of the tree is: $\log_2 n$ **.**

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?

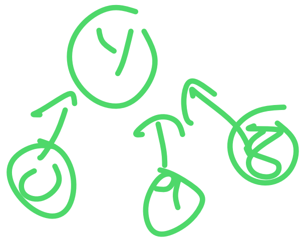
always storing -1

↳ store -height -1

or

- size

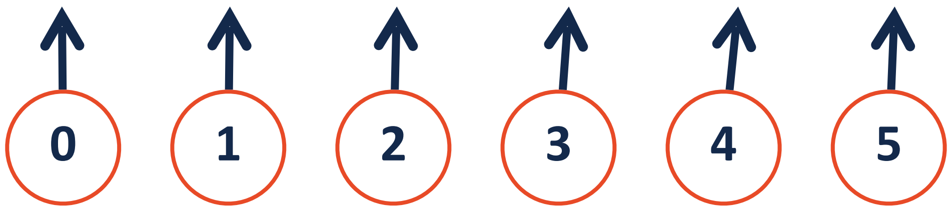
Ideal tree:



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

Disjoint Sets Union Example

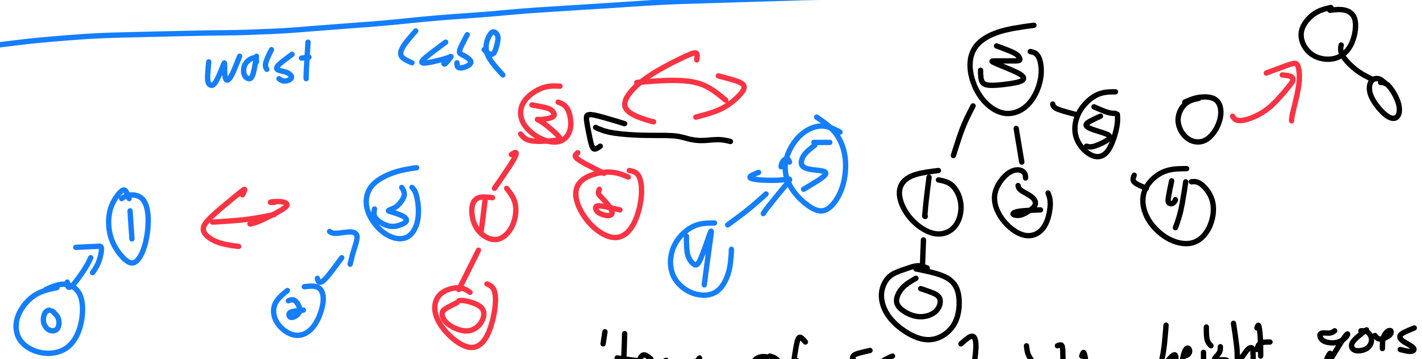
Only in our heads



actual storage

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

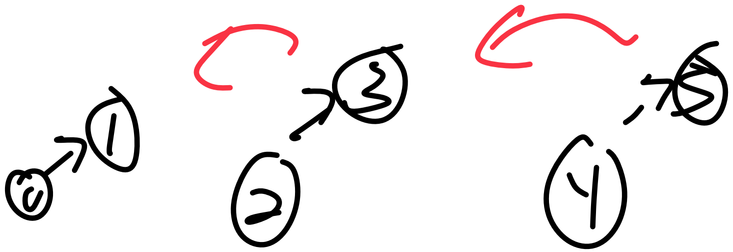
worst case



By height

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

when I union two items of same height, height goes up



By size

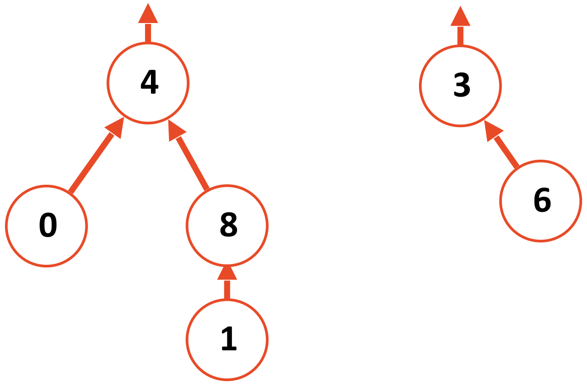
ends up same ish

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

h is O(log n)

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

Claim: An UpTree of height **h** has nodes ≥ 2^h

Base Case:

height 0 uptree

1 node == 2^0 = 1

Disjoint Sets Union by Size

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

IH: For all trees built w/ i-1 unions or less our claim is true

Prove true for the i-th union

Let A, B be two sets

$n(B)$ = size of B / $n(A)$ = size of A

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

Case 1: height(A) < height(B) :) → height won't increase

Case 2: height(A) == height(B) :(→ height will increase!

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

Disjoint Sets Union by Size

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

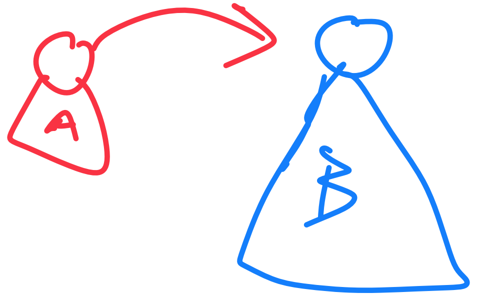
union $< i$th union

Built sometime before now

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)



larger in height and size

new size of B = $n(B') \leq n(B) + n(A)$

my height remains $h(B)$

Trivial proof b/c by IH:

$n(B) \geq 2^{h(B)}$

$n(B') > n(B) \geq 2^{h(B)}$

:)

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)!