Data Structures

Extra Credit Project and Disjoint Sets

CS 225
Brad Solomon & G Carl Evans

October 16, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Discuss extra credit project

Finish analyzing efficiency of minHeap

Introduce disjoint sets

Big Picture: Extra credit project

Do something that is of personal interest to you!

Want to do undergrad research? Find a foundational algorithm!

Want to go off into industry? Demonstrate knowledge with code!

Want extra credit points? Use one of the suggested algorithms!

ECP Proposal

You are 'writing' your own assignment skeleton

1. Function I/O (in written proposal)

2. Tests (in Github repo)

3. Datasets (in Github repo)

ECP Proposal

You dont need to know how to implement to propose a structure!

ECP Mid-Project Check-in

Meet with your mentor to confirm your algorithm works!

ECP Final Deliverables

Prove your algorithm is correct and estimate runtime

1. Submit code base (GitHub repo)

2. Write a report that describes proof of correctness and efficiency

3. Present your work! Highlight what you did!

Proving buildHeap Running Time

Theorem: The running time of buildHeap on array of size **n** is:

Strategy:

1. Call heapifyDown() on every non-leaf node

2. Every node we heapifyDown() has its height as worst case work.

Proving buildHeap Running Time

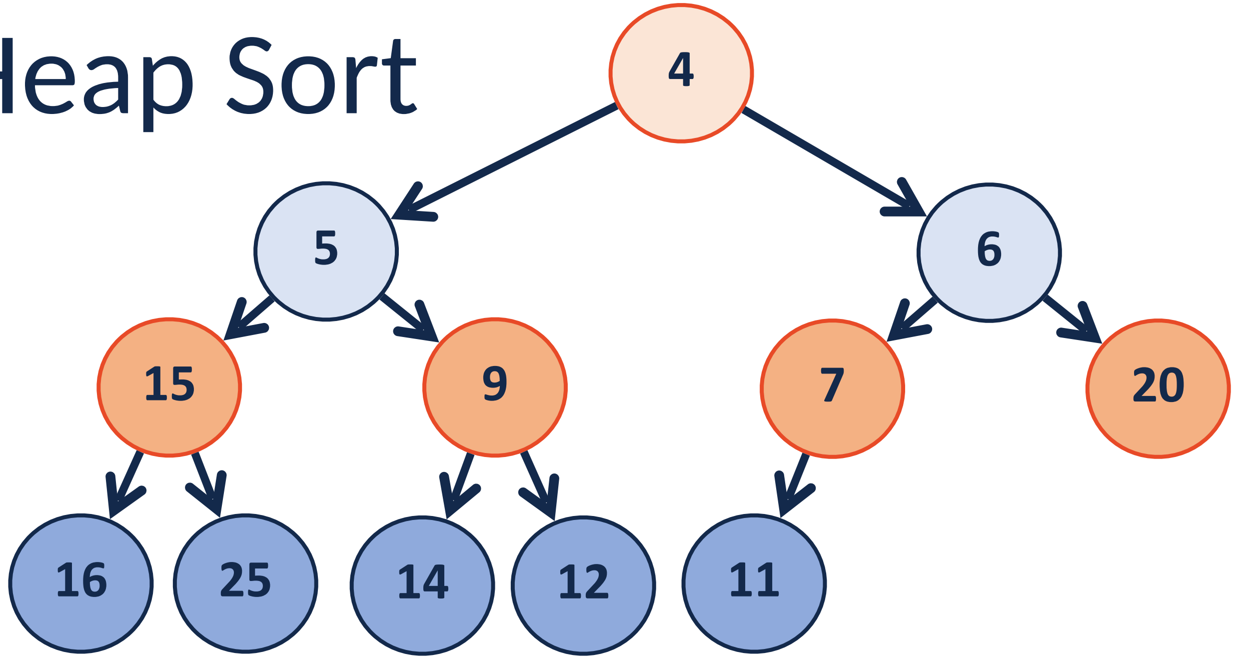
Theorem: The running time of buildHeap on array of size **n** is O(n)

$$S(h) = s^{h+1} - 2 - h$$

How can we relate **h** and **n**?

How can we estimate running time?

Heap Sort



1.

2.

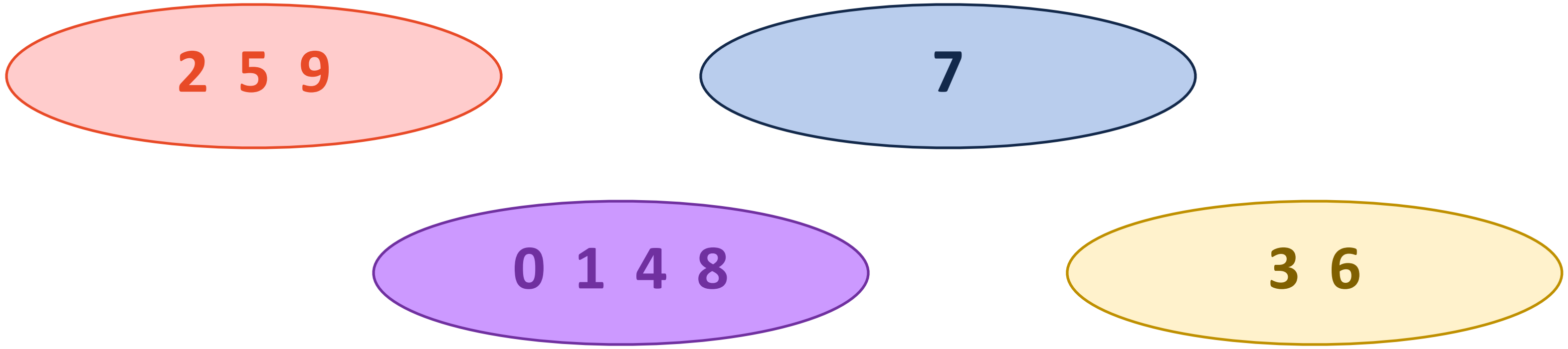
3.

	4	5	6	15	9	7	20	16	25	14	12	11			

Running time?

minHeap is a good example of tradeoffs:

Disjoint Sets

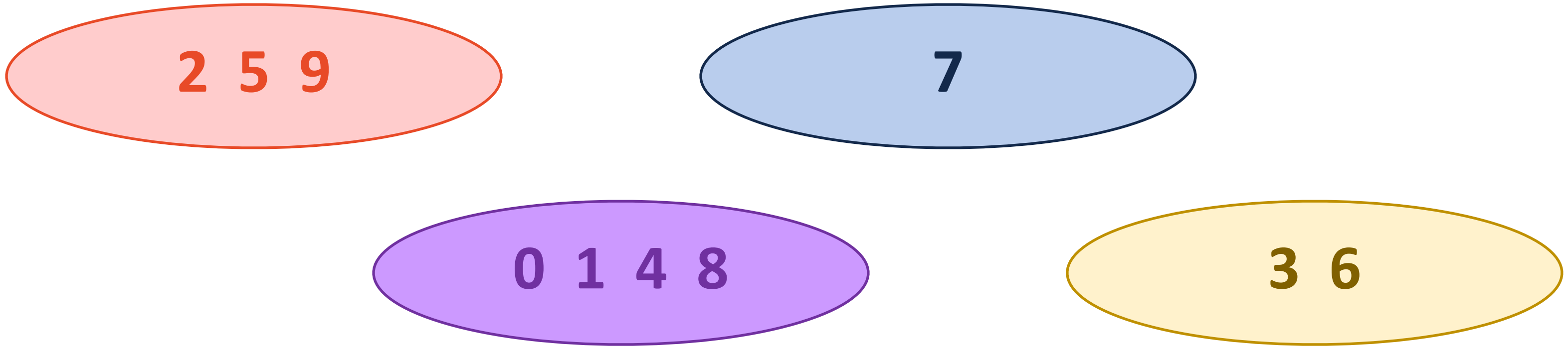


Key Ideas:

- Each element exists in exactly one set.
- Every item in each set has the same representation
  - In other words: find(4) == find(8) == find(0) …
- Each set has a different representation
  - In other words: find(7) != find(4)

Disjoint Sets

Each set is represented by a canonical element (internally defined)

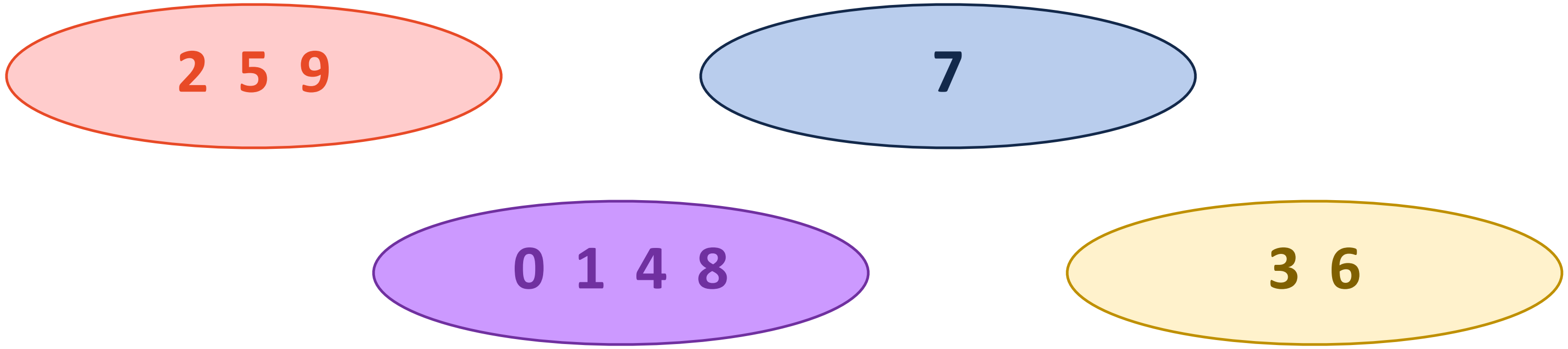


Operation:

```
find(4) == find(8)
```

Disjoint Sets

The union operation combines two sets into one set.

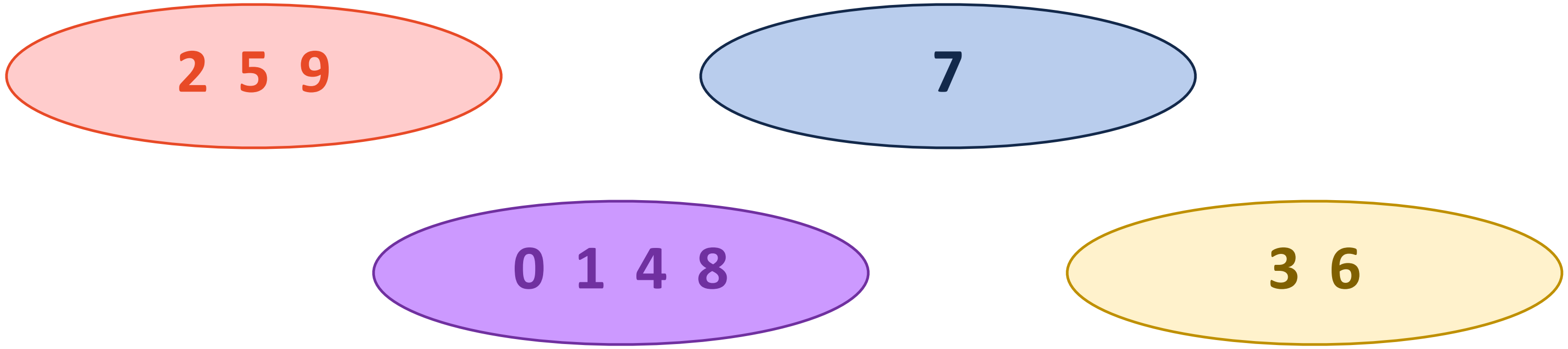


Operation:

```
if find(2) != find(7){
  union( find(2), find(7) );
}
```

Disjoint Sets

We add new items to our 'universe' by making new sets.



Operation:

```
makeSet(10);
```

Disjoint Sets ADT

Constructor

makeSet

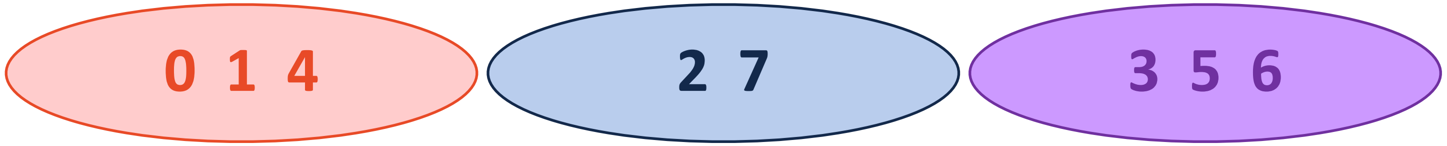
Find

Union

Disjoint Sets

How might we implement a disjoint set?

Implementation #1

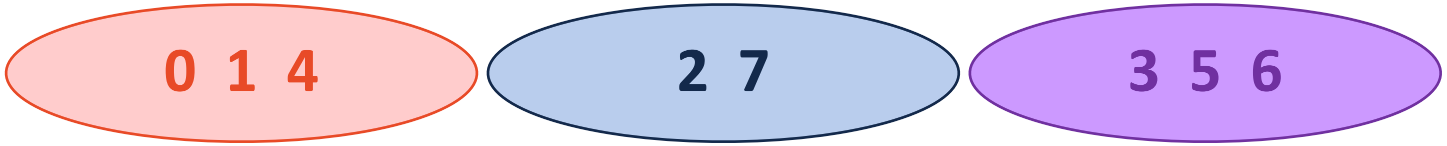


0	1	2	3	4	5	6	7

Find(k):

Union(k_1, k_2):

Implementation #2

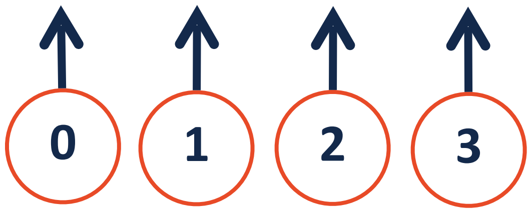


0	1	2	3	4	5	6	7

Find(k):

Union(k_1, k_2):

UpTrees



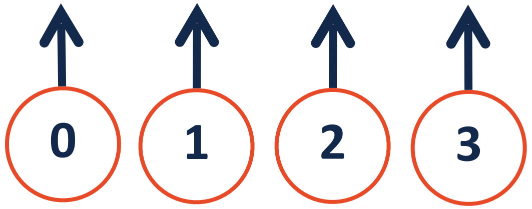
0	1	2	3

0	1	2	3

0	1	2	3

0	1	2	3

UpTrees



0	1	2	3

0	1	2	3

0	1	2	3

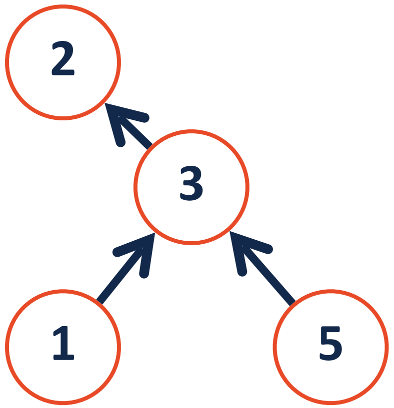
0	1	2	3

Disjoint Sets Representation

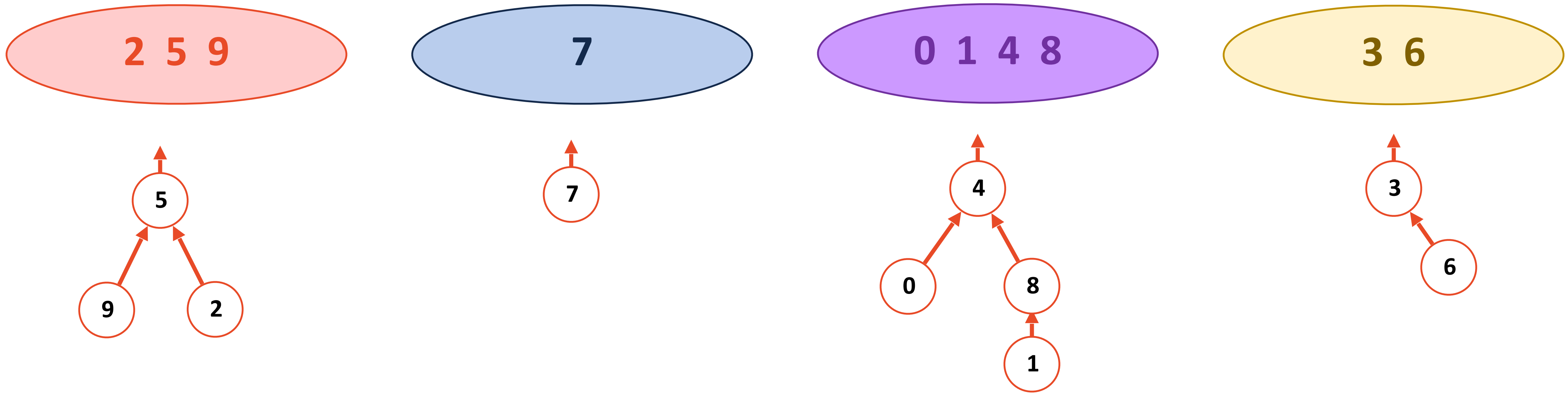
We can represent a disjoint set as an array where the key is the index

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



0	1	2	3	4	5	6	7	8	9

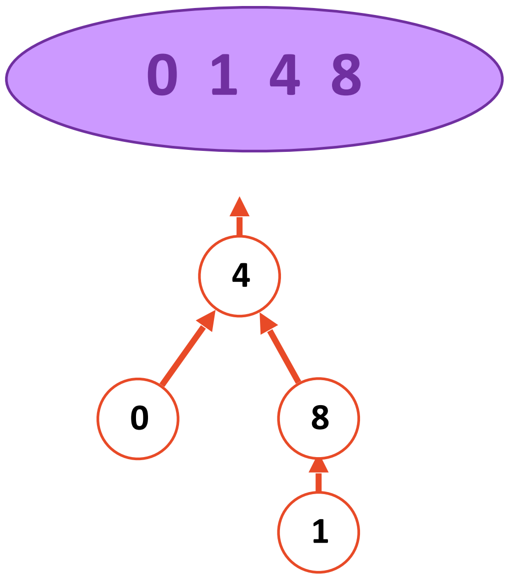
Disjoint Sets Find

Find(1)

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

Running time?

What is ideal UpTree?

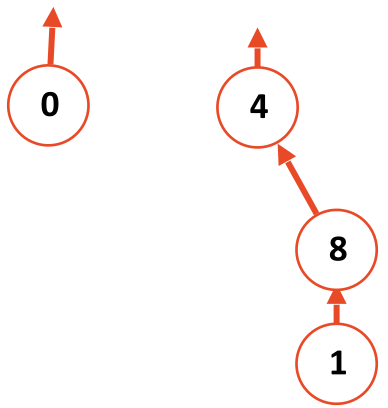


0	1	2	3	4	5	6	7	8	9
4	8			-1				4	

Disjoint Sets Union

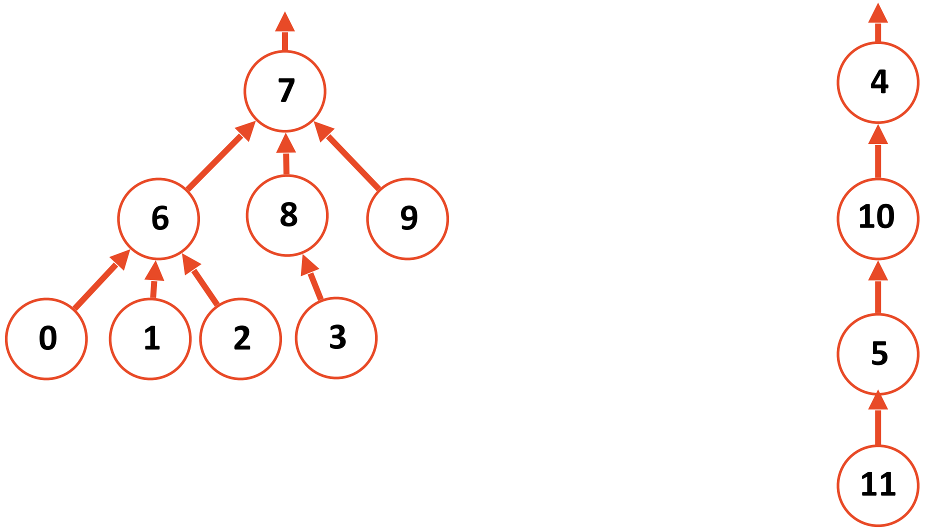
Union(0, 4)

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



}
```



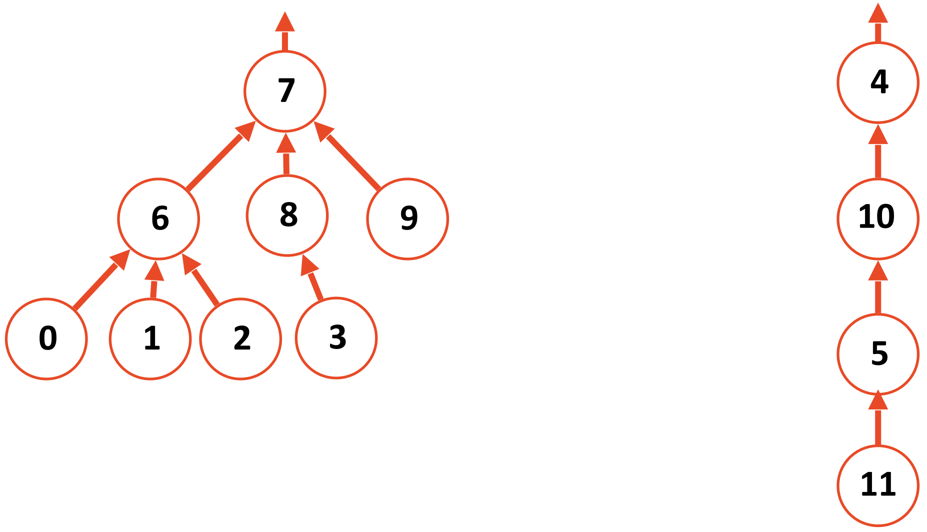
0	1	2	3	4	5	6	7	8	9
-1	8			-1				4	

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

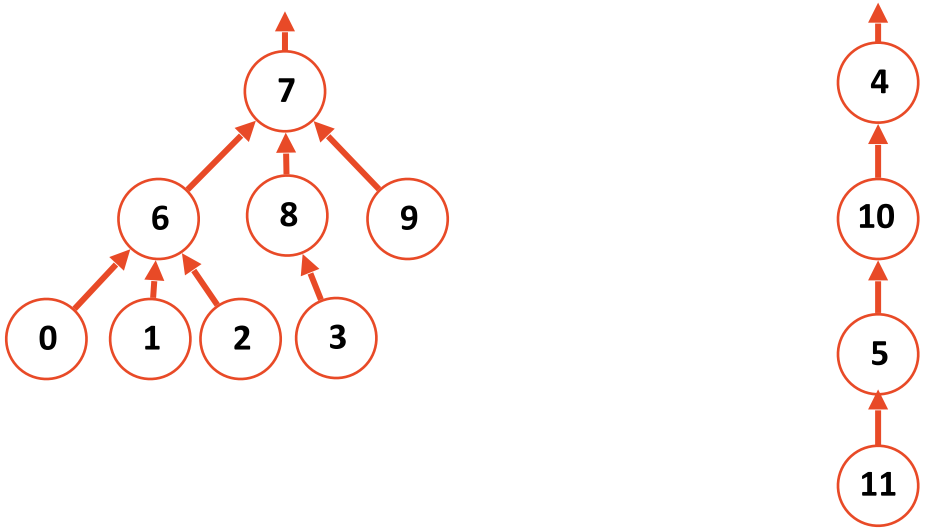


Union by height

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

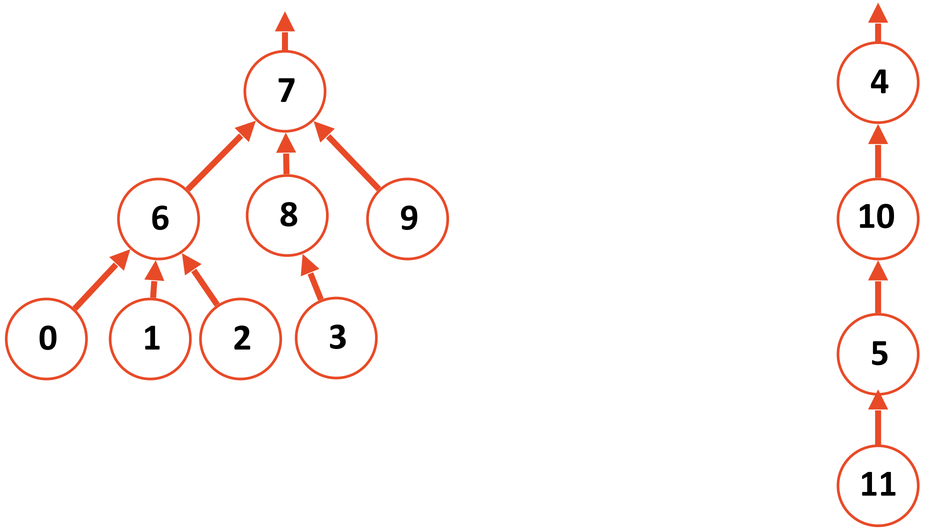


Union by size

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*

Both guarantee the height of the tree is: _______________.