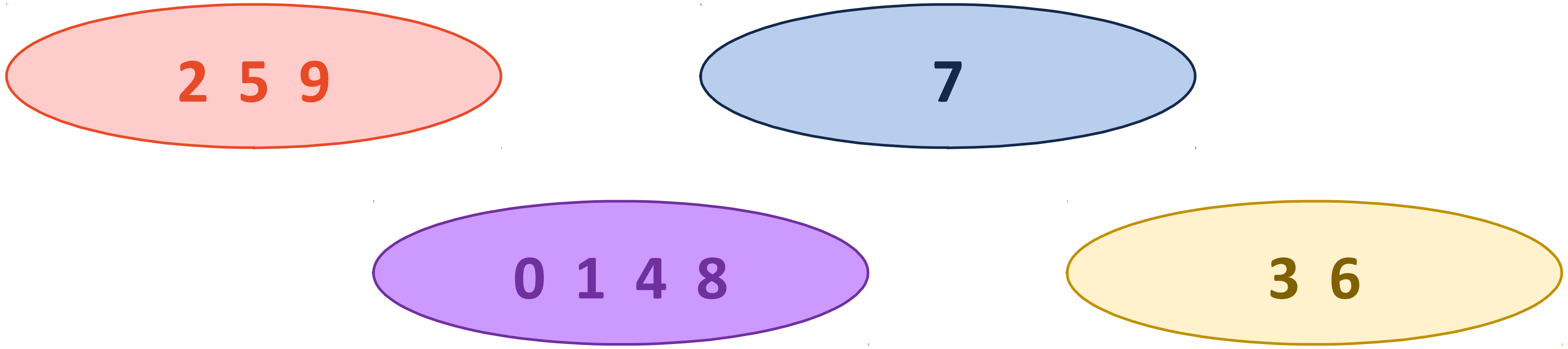
CS 400

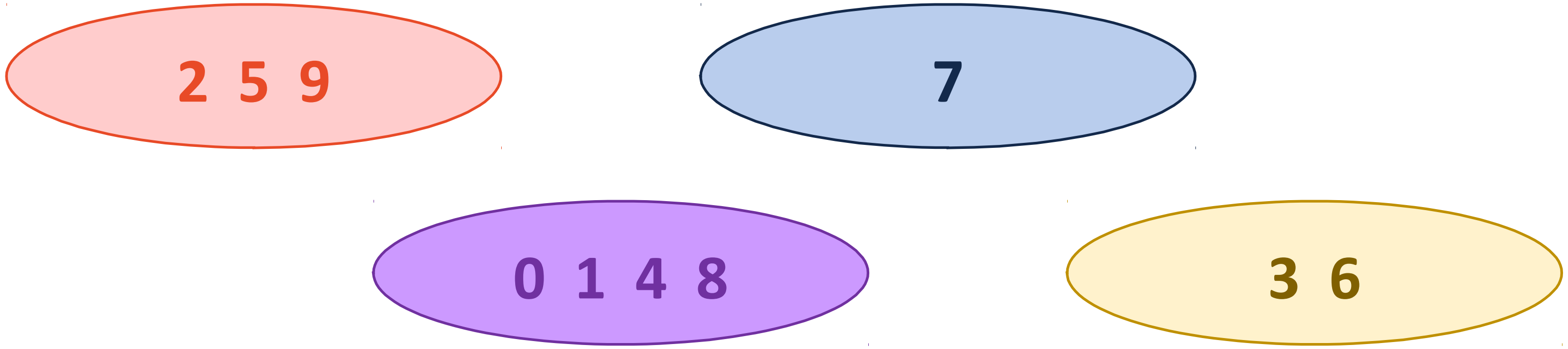
Disjoint Sets

ID: 11-01

Disjoint Sets

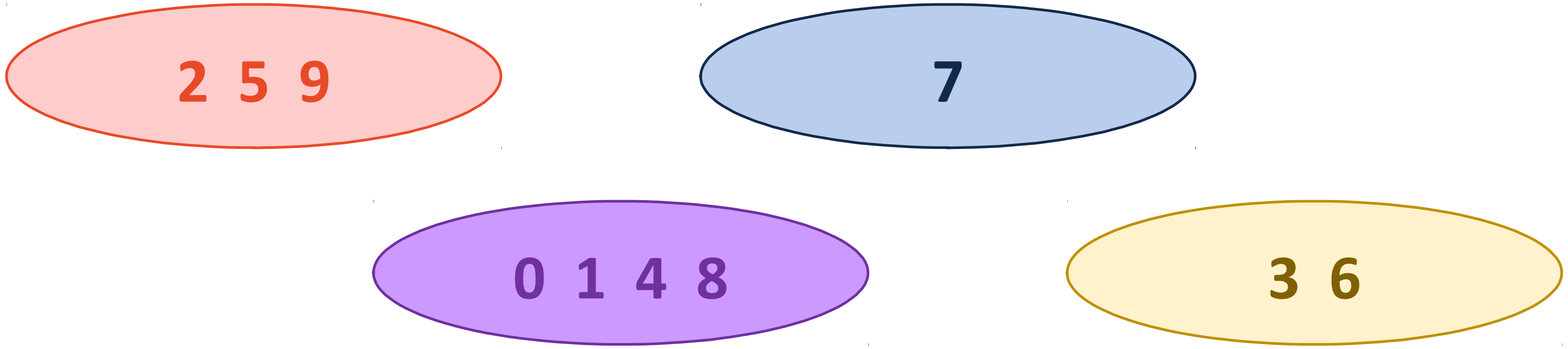


Disjoint Sets



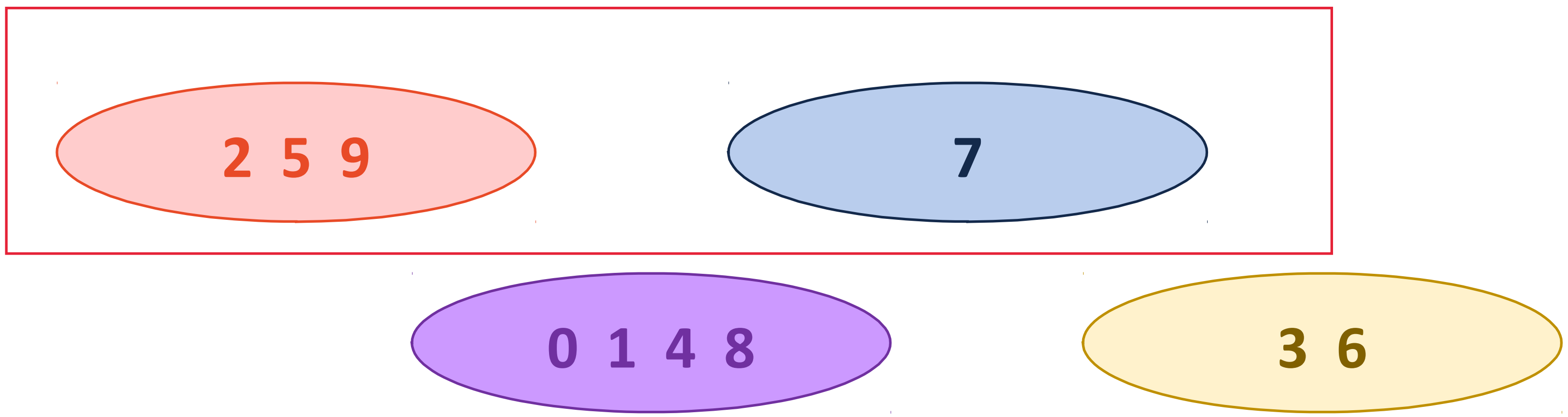
Operation: find(4)

Disjoint Sets



Operation: find(4) == find(8)

Disjoint Sets



Operation:

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

Disjoint Sets ADT

- Maintain a collection $S = \{s_0, s_1, \ldots s_k\}$

- Each set has a representative member.

- API:
```
void makeSet(const T & t);
void union(const T & k1, const T & k2);
T & find(const T & k);
```

CS 400

Disjoint Sets: Implementation #1

ID: 11-02

Implementation #1

0 1 4

2 7

3 5 6

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

value is ID

Find(k): return value, so O(1)

Union(k1, k2): need to go through whole list using find(k) and change ID, so O(n)

CS 400

Disjoint Sets: UpTrees

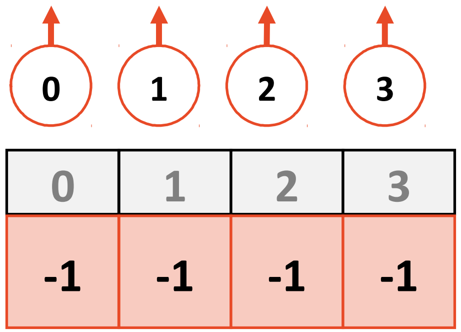
ID: 11-03

Implementation #2

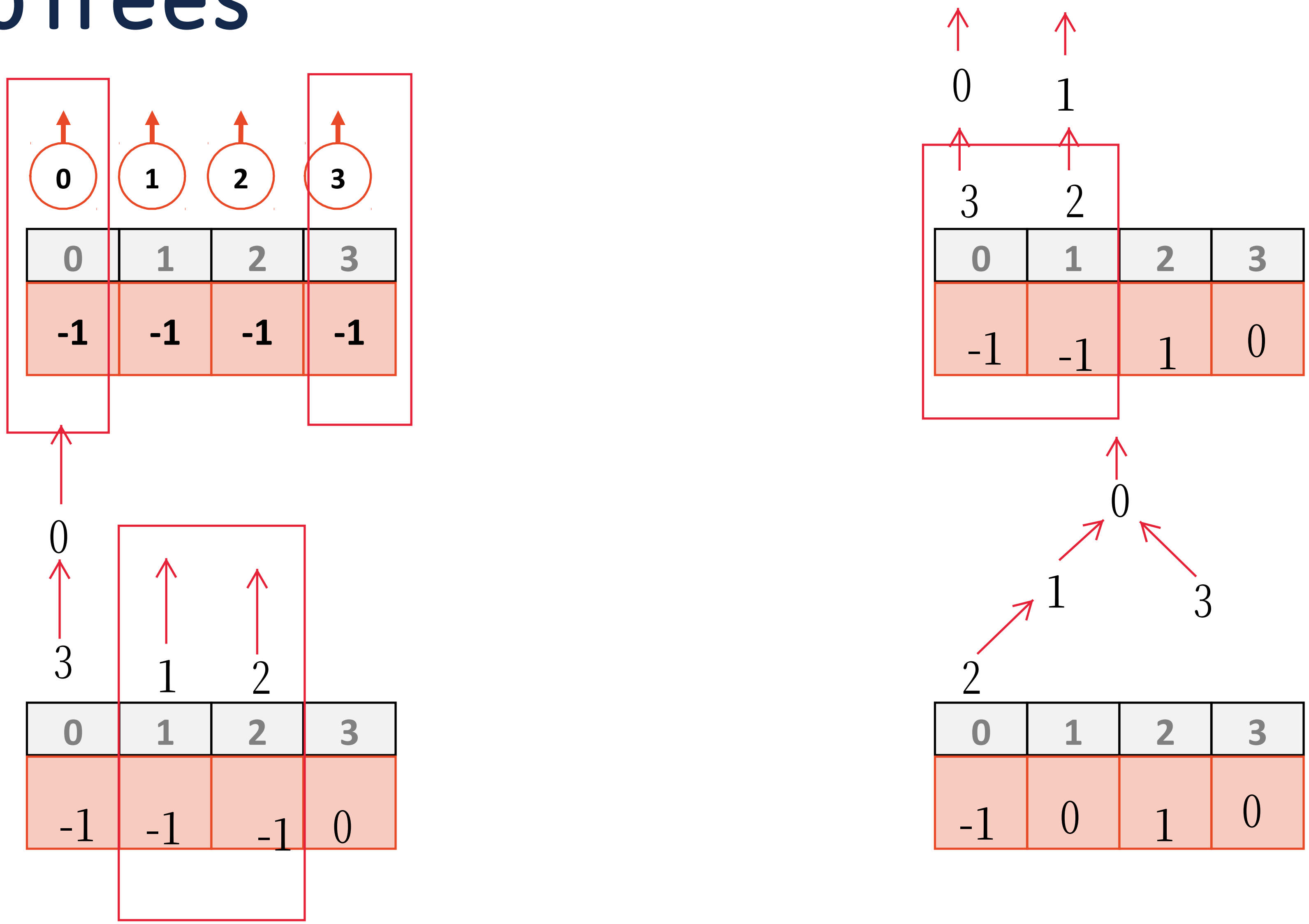
- We will continue to use an array where the index is the key

- The value of the array is:
  - **-1,** if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element

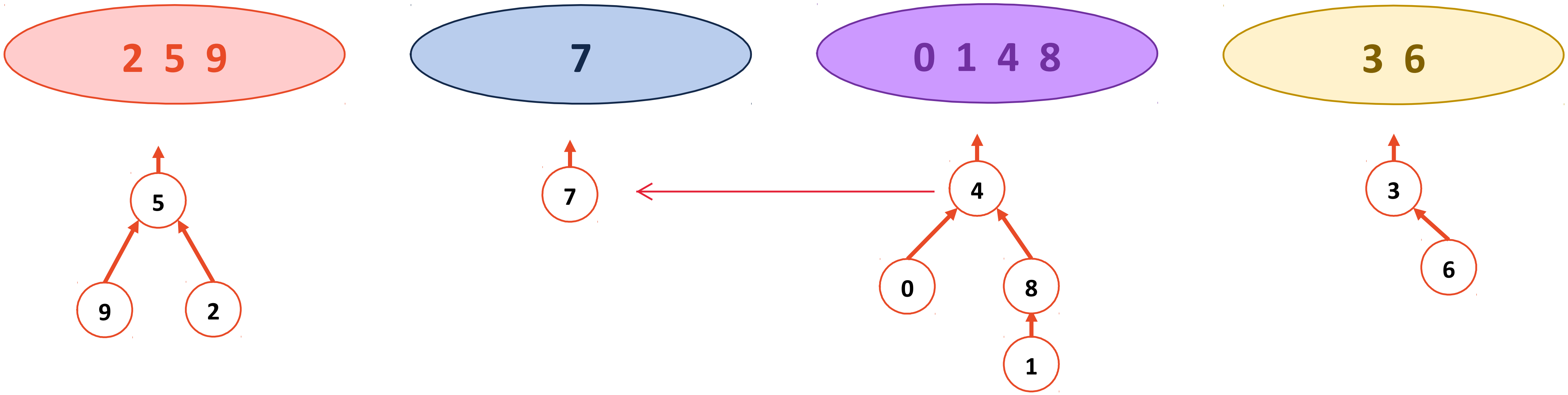
- We will call theses **UpTrees**:



UpTrees



Disjoint Sets



Only update root node when union by adding a single pointer.

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

CS 400

UpTrees: Simple Running Time

ID: 11-04

Disjoint Sets Find

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

Running time?

O(h), where h is the height of the tree

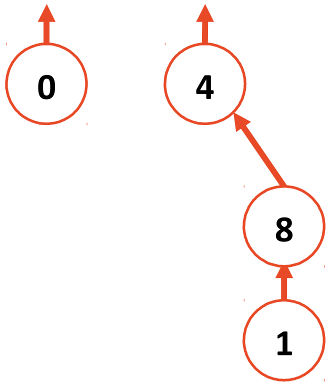
What is the ideal UpTree?

Flat tree, every children is under ID node

Disjoint Sets Union

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


}
```

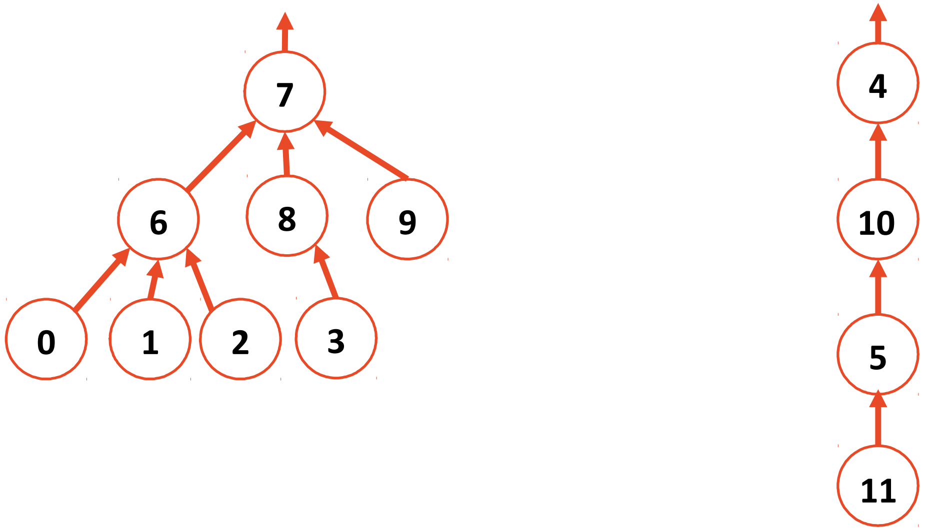


CS 400

UpTrees: Smart Union and Path Compression

ID: 11-05

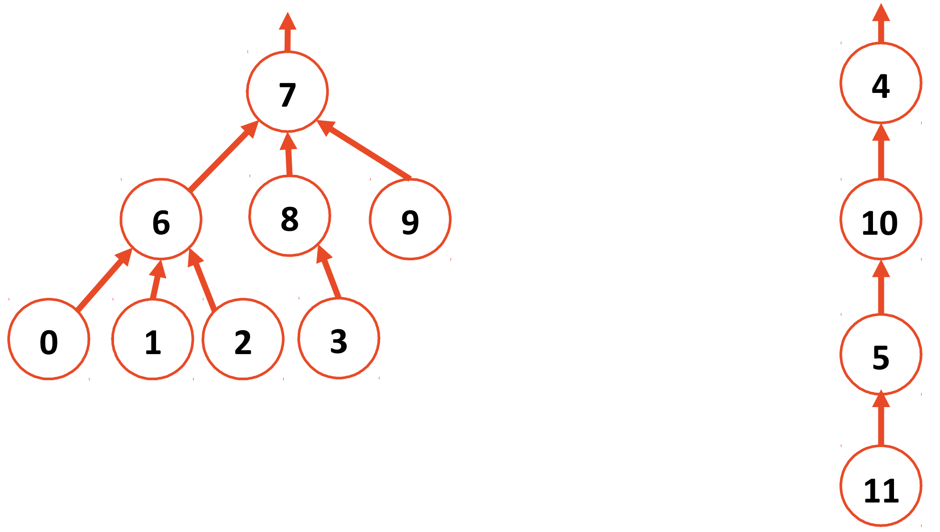
Disjoint Sets – Union



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

Add shorter tree to taller tree, so store height at root node and use (-h-1)

Disjoint Sets – Smart Union



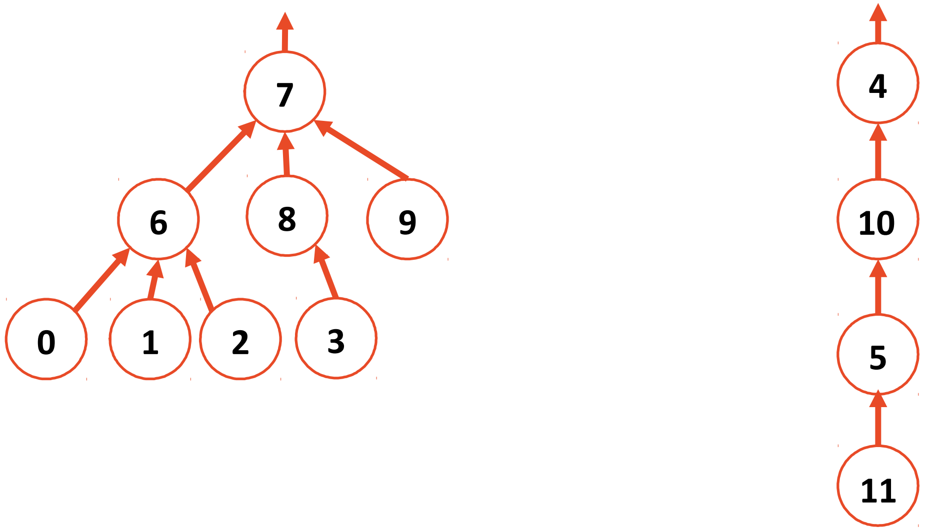
Union by height

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

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

-h-1

Disjoint Sets – Smart Union



$\boxed{O(\log(n))}$

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	-4	10	7	-8	7	7	4	5

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

Both guarantee the height of the tree is: _______________.

Add smaller size to larger size

Disjoint Sets Find

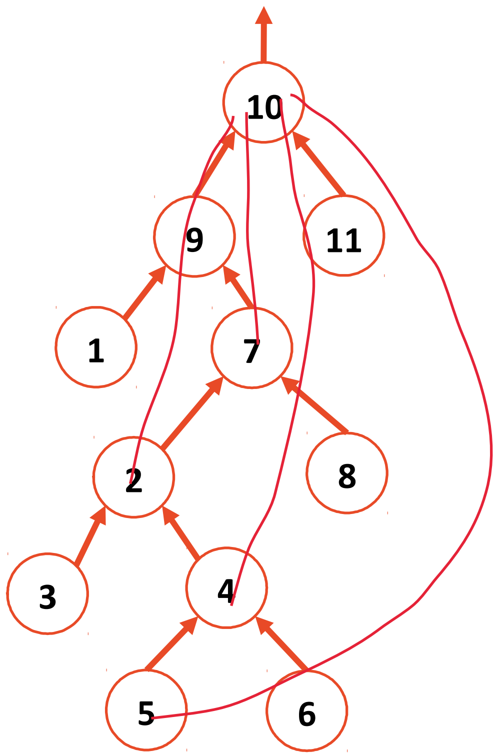
```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return _find( s[i] ); }
}
```

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

  // If arr_[root1] is less than (more negative), it is the larger set;
  // we union the smaller set, root2, with root1.
  if ( arr_[root1] < arr_[root2] ) {
    arr_[root2] = root1;
    arr_[root1] = newSize;
  }

  // Otherwise, do the opposite:
  else {
    arr_[root1] = root2;
    arr_[root2] = newSize;
  }
}
```

Path Compression



Use recursion to record every step down.

Disjoint Sets Analysis

The **iterated log** function:

The number of times you can take a log of a number.

$$\log^*(n) = \begin{cases} 0 & , n \le 1 \\ 1 + \log^*(\log(n)) & , n > 1 \end{cases}$$

What is $\mathbf{lg^*(2^{65536})}$?

Disjoint Sets Analysis

In an Disjoint Sets implemented with smart **unions** and path compression on **find**:

Any sequence of **m union** and **find** operations result in the worse case running time of O(___m * log(n)___),

where **n** is the number of items in the Disjoint Sets.