CS 400

Disjoint Sets

ID: 11-01

Disjoint Sets

Disjoint Sets

2 5 9

7

0 1 4 8

3 6
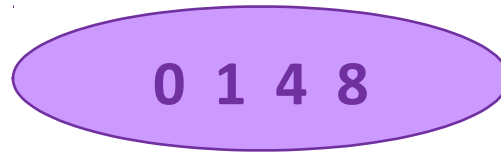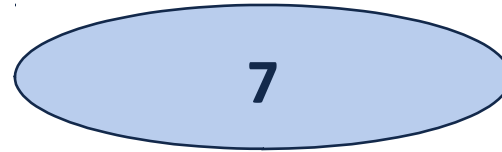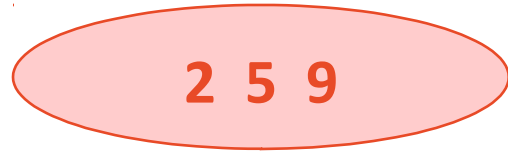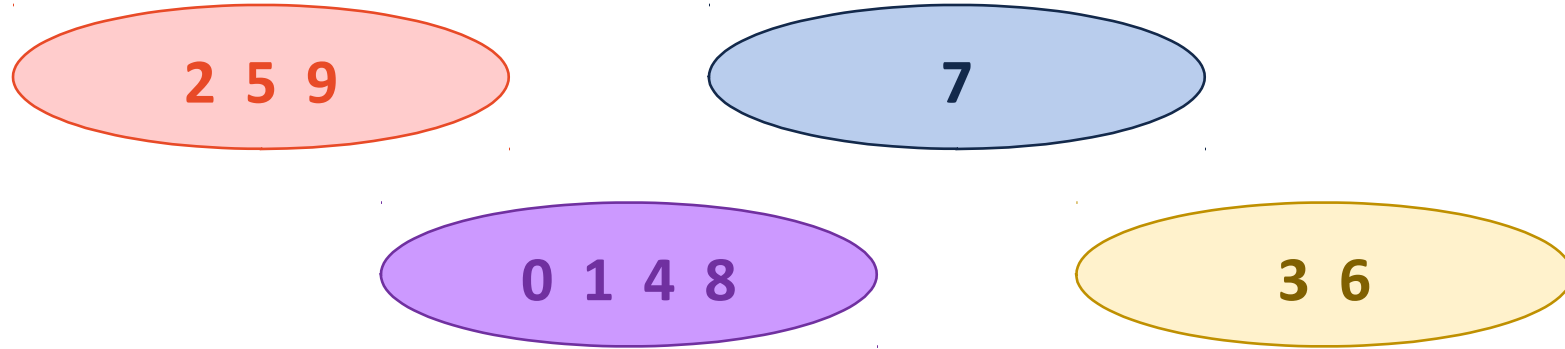
Operation: find(4)

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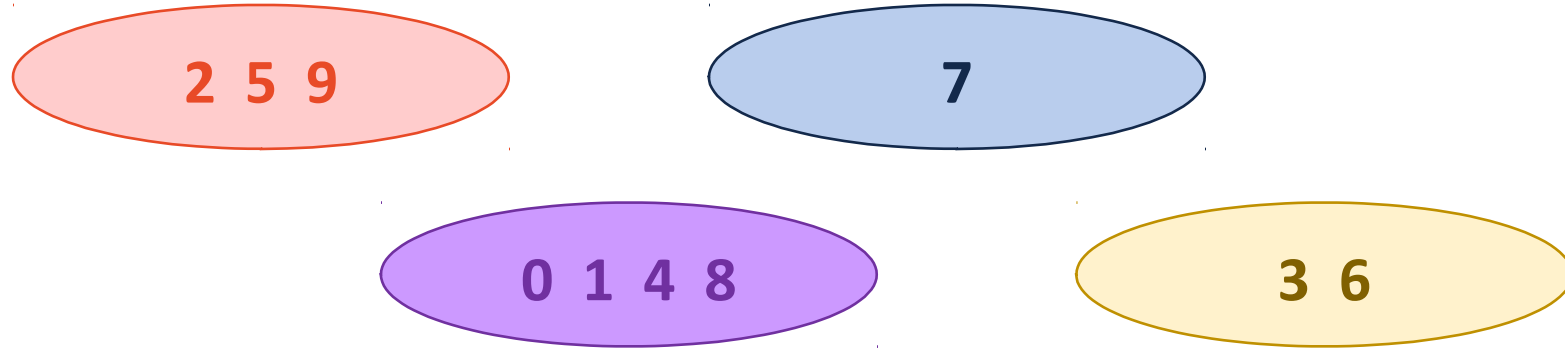
Operation: find(4) == find(8)

Disjoint Sets

2 5 9

7

0 1 4 8

3 6
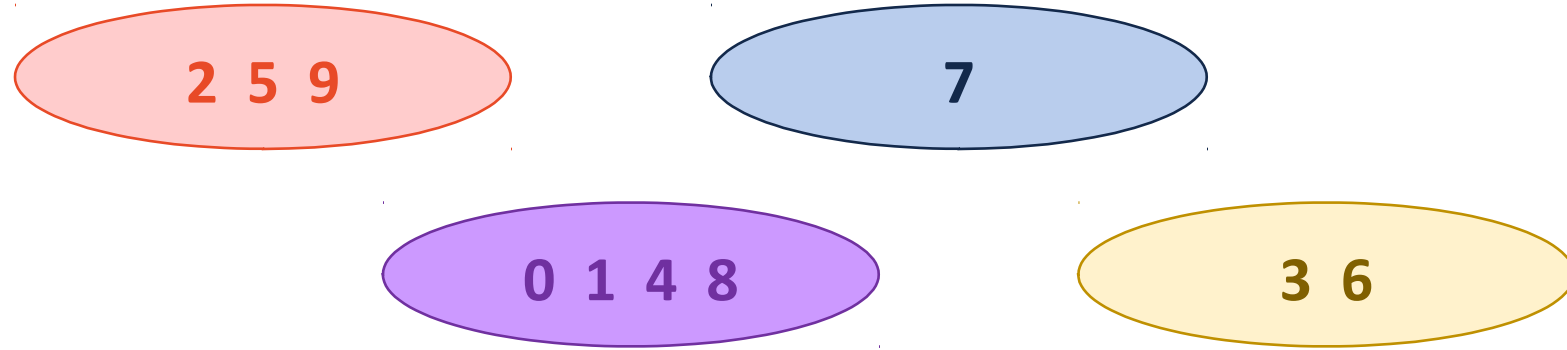
Operation:

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

Disjoint Sets ADT

- Maintain a collection S = {s_0, s_1, ... 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

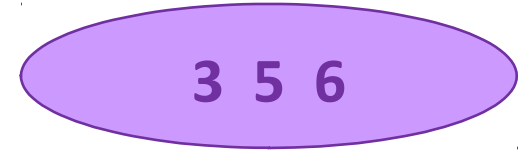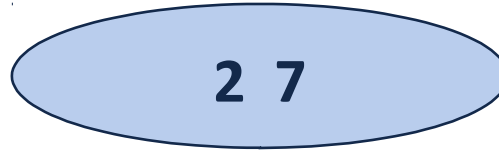| 0	1	2	3	4	5	6	7
0	0	2	3	0	3	3	2

Find(k):

Union(k1, k2):

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**:
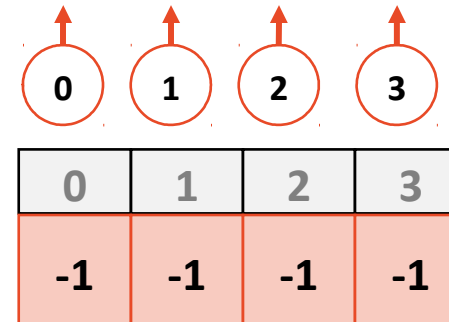
0	1	2	3
-1	-1	-1	-1
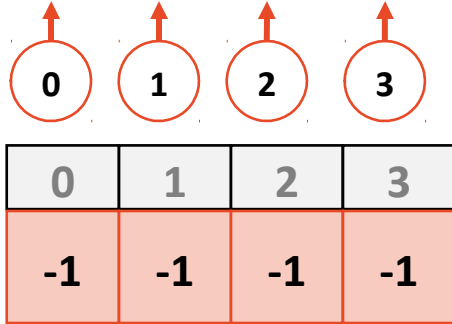
UpTrees

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

0	1	2	3

0	1	2	3

0	1	2	3

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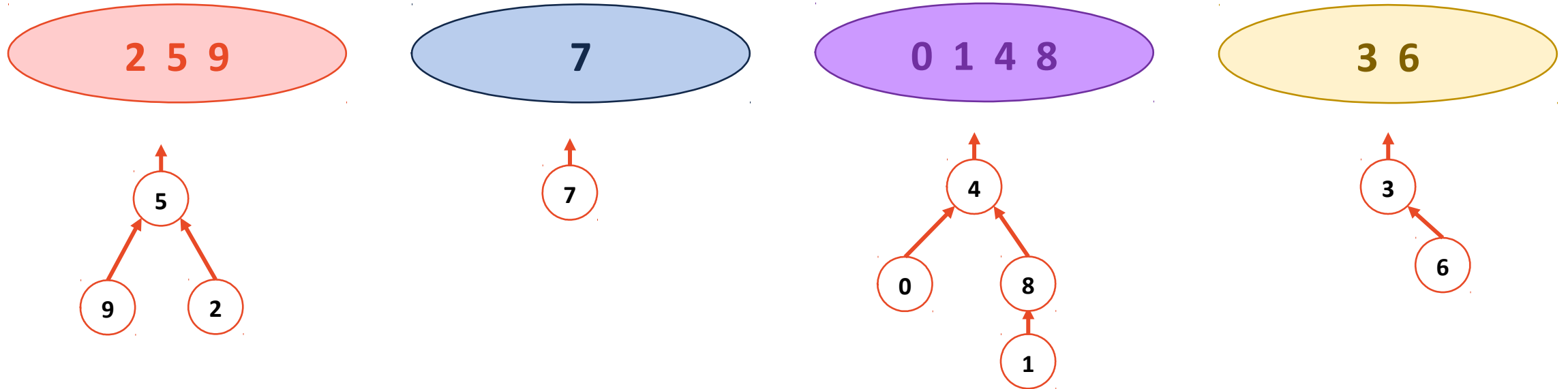
0	1	2	3	4	5	6	7	8	9
4	8	5	6	-1	-1	-1	-1	4	5

CS 400

UpTrees: Simple Running Time

ID: 11-04

Disjoint Sets Find

```
1  int DisjointSets::find() {
2    if ( s[i] < 0 ) { return i; }
3    else { return _find( s[i] ); }
4  }
```

Running time?

What is the ideal UpTree?

Disjoint Sets Union

```
1   void DisjointSets::union(int r1, int r2) {
2
3
4   }
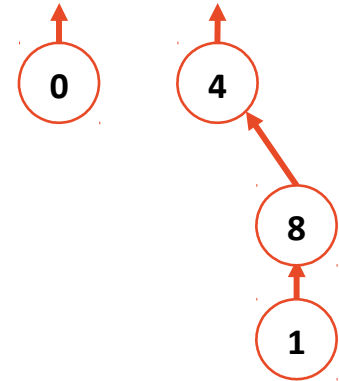```

# CS 400

**UpTrees: Smart Union and Path Compression**


**ID: 11-05**

# Disjoint Sets – Union
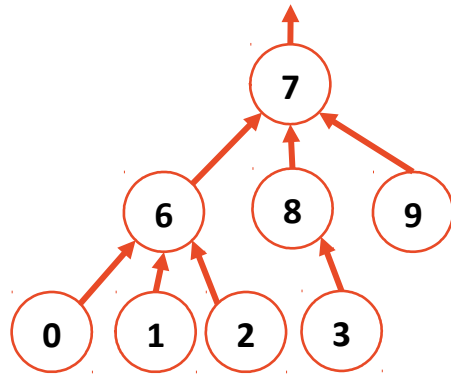


| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|----|----|
| 6 | 6 | 6 | 8 | -1 | 10 | 7 | -1 | 7 | 7 | 4 | 5 |

# Disjoint Sets – Smart Union
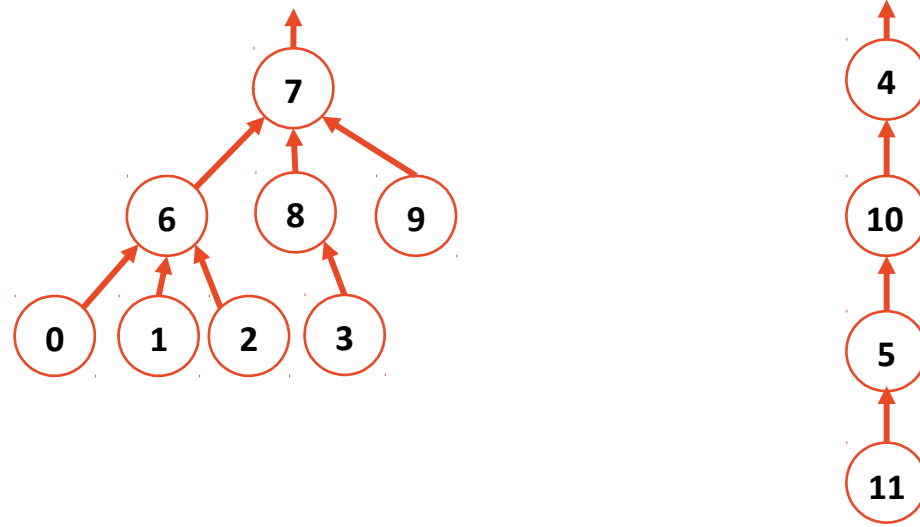


**Union by height**

| 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
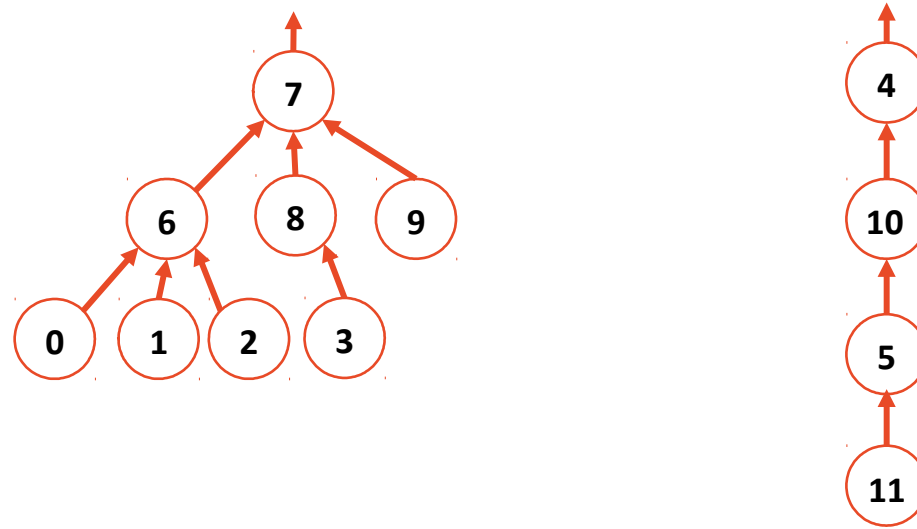


**Union by height**

| 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: _____.**

# Disjoint Sets Find

```
1  int DisjointSets::find(int i) {
2    if ( s[i] < 0 ) { return i; }
3    else { return _find( s[i] ); }
4  }
```

```
1   void DisjointSets::unionBySize(int root1, int root2) {
2     int newSize = arr_[root1] + arr_[root2];
3
4     // If arr_[root1] is less than (more negative), it is the larger set;
5     // we union the smaller set, root2, with root1.
6     if ( arr_[root1] < arr_[root2] ) {
7       arr_[root2] = root1;
8       arr_[root1] = newSize;
9     }
10
11    // Otherwise, do the opposite:
12    else {
13      arr_[root1] = root2;
14      arr_[root2] = newSize;
15    }
16  }
```

Path Compression
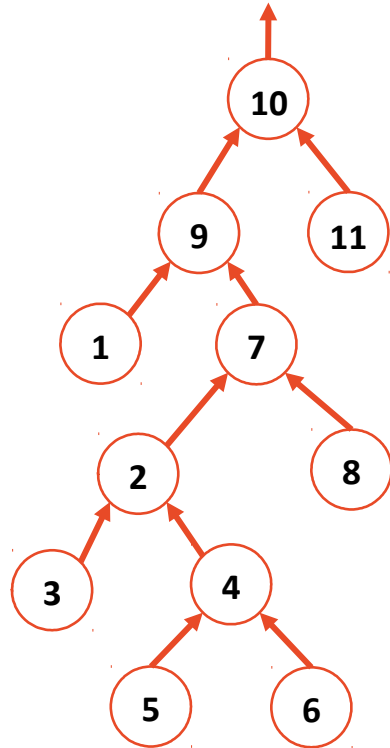
Disjoint Sets Analysis

The **iterated log** function:

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

$\log^*(n) =$

    0 $, n \leq 1$

    $1 + \log^*(\log(n))$ $, n > 1$

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

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(_____),
    where **n** is the number of items in the Disjoint Sets.