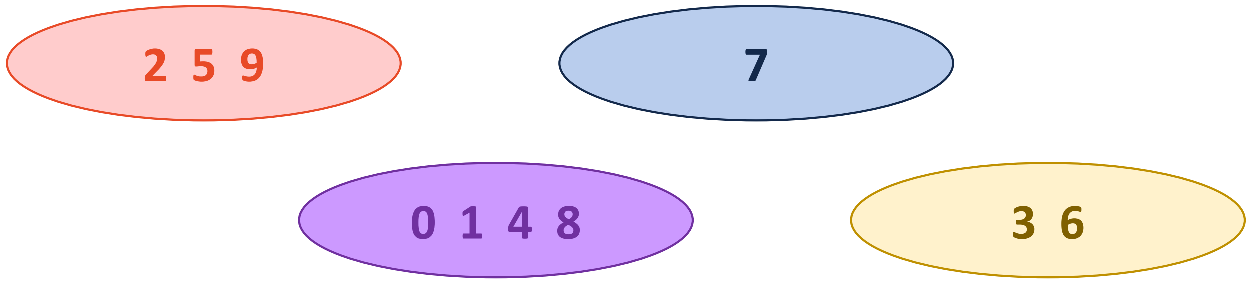
CS 225

Data Structures

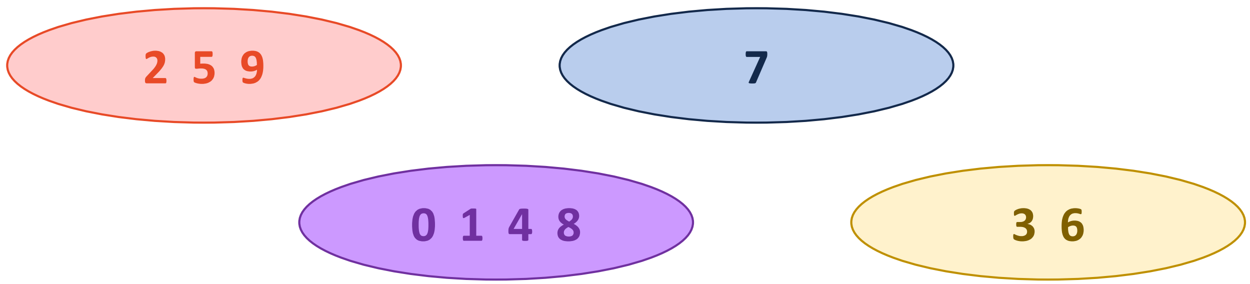
March 6 – Disjoint Sets

G Carl Evans

Disjoint Sets

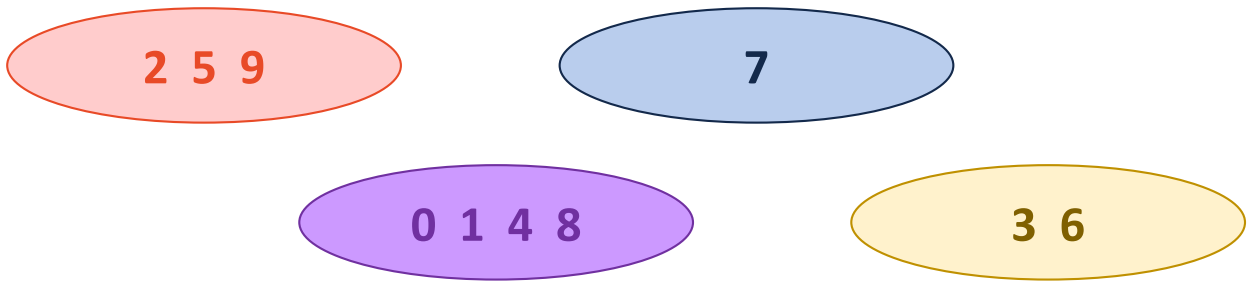


Disjoint Sets



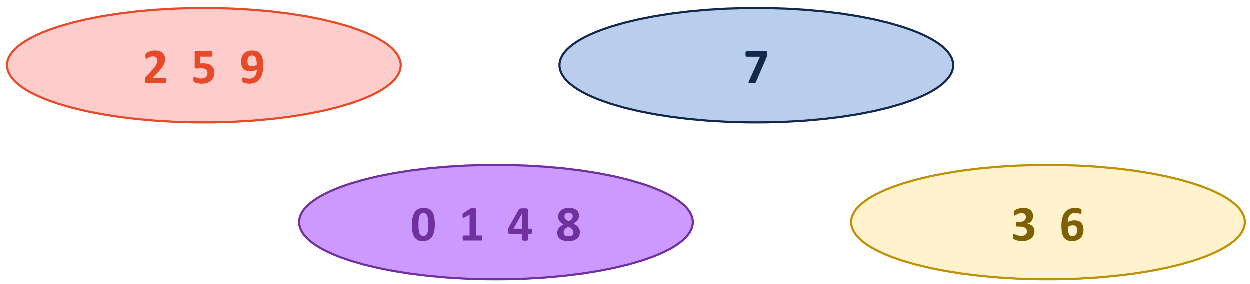
Operation: find(4)

Disjoint Sets



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

Disjoint Sets



Key Ideas:

- Each element exists in exactly one set.
- Every set is an equitant representation.
  - Mathematically: $4 \in [0]_R \rightarrow 8 \in [0]_R$
  - Programmatically: find(4) == find(8)

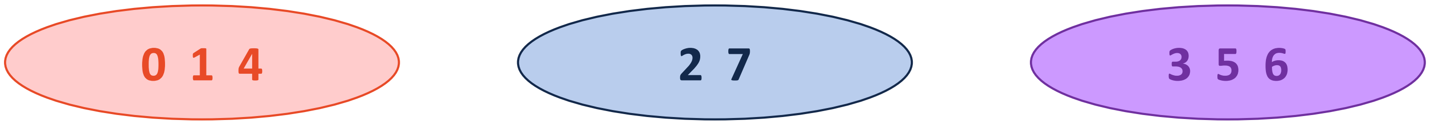
Disjoint Sets ADT

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

- Each set has a representative member.

- API:
```
void makeSets(int number);
void union(int k1, const int k2);
int  find(int k);
```

Implementation #1



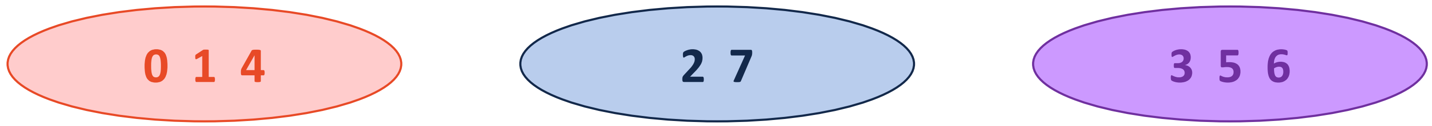
0	1	2	3	4	5	6	7

Find(k):

Union(k1, k2):


YOU EXPECTED A NEW DATA STRUCTURE
BUT IT WAS ME, TREE ALL ALONG
imgflip.com

Implementation #2



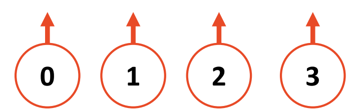
0	1	2	3	4	5	6	7

Find(k):

Union(k1, k2):

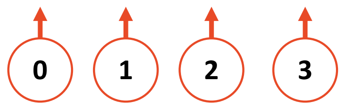
Implementation #2

- 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



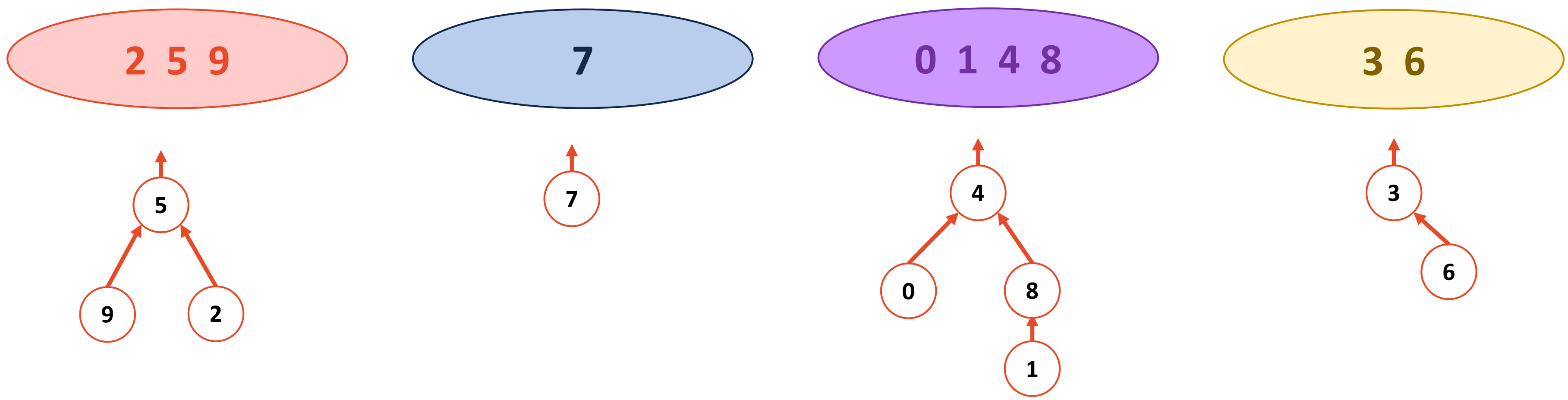
0	1	2	3

0	1	2	3

0	1	2	3

0	1	2	3

Disjoint Sets



0	1	2	3	4	5	6	7	8	9

Find(k):

Union(k1, k2):

Disjoint Sets Find

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

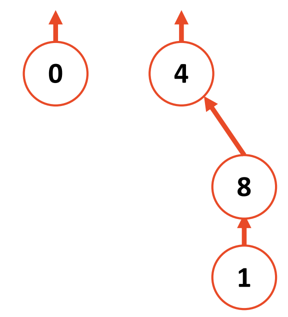
Running time?

What is the ideal UpTree?

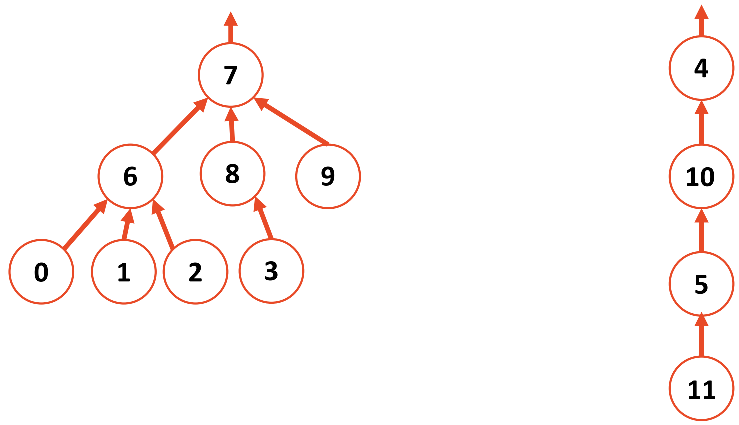
Disjoint Sets Union

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


}
```

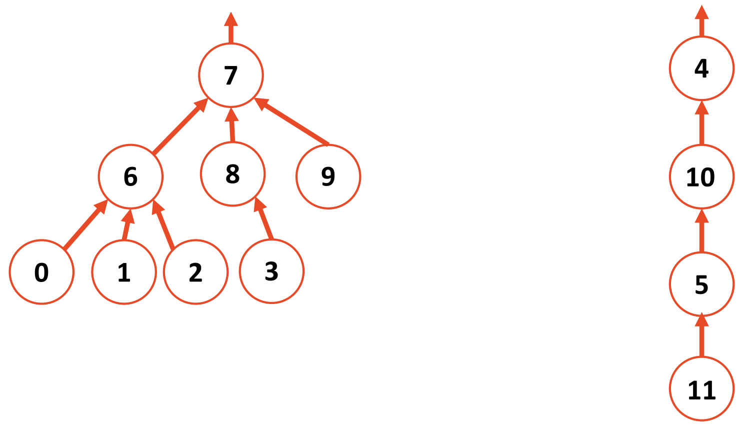


Disjoint Sets – Union



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

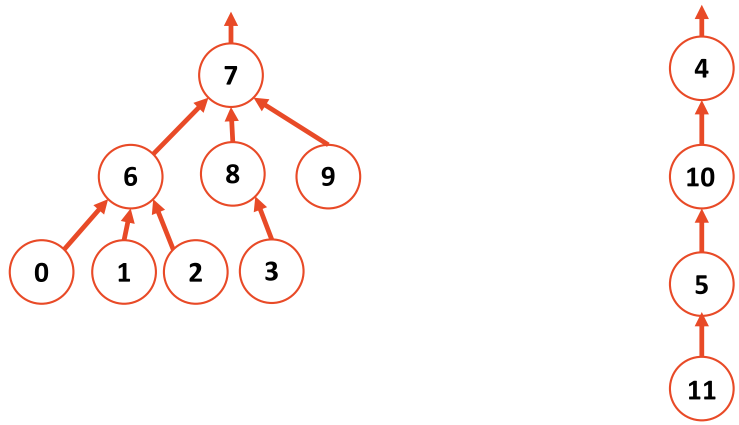


Union by height

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

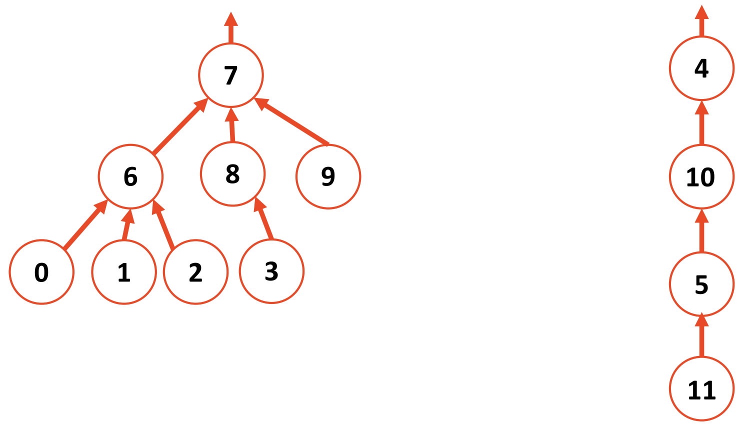


Union by size

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

***Idea**: 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

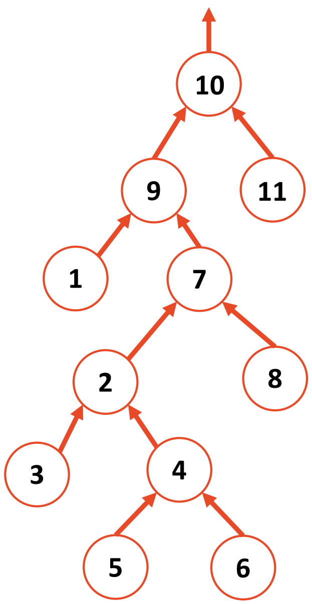
```
int DisjointSets::find(int i) {
  if ( s[i] < 0 ) { return i; }
  else { return find( s[i] ); }
}
```

```
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



Disjoint Sets Analysis

The **iterated log** function:

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

log*(n) =

$0 \quad , n \leq 1$

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

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

Disjoint Sets Analysis

In a 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.