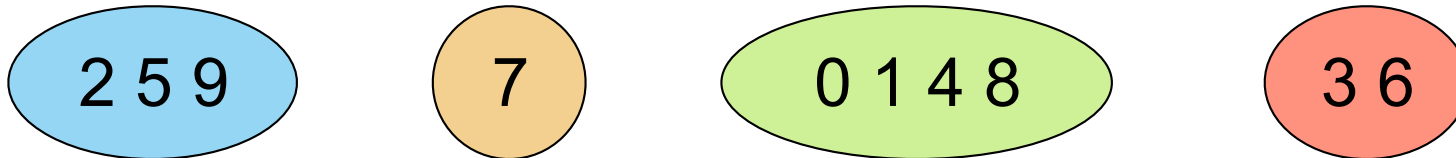


Today's announcements:

MP7 available, due 4/28, 11:59p. EC due 4/21, 11:59p.

A Disjoint Sets example:

Let R be an equivalence relation on the set of students in this room, where $(s, t) \in R$ if s and t have the same favorite among A, FB, TR, CC, PMC, ____}.



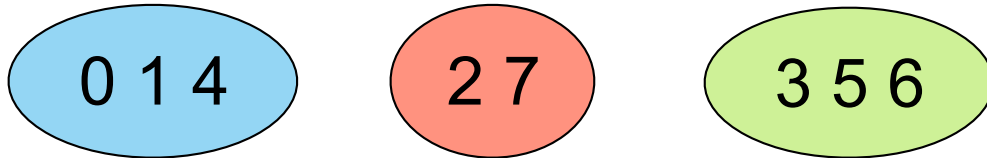
1. Find(4)
2. Find(4)==Find(8)
3. If $!(\text{Find}(7) == \text{Find}(2))$ then Union(Find(7), Find(2))

Disjoint Sets ADT

We will implement a data structure in support of “Disjoint Sets”:

- Maintains a collection $S = \{s_0, s_1, \dots, s_k\}$ of disjoint sets.
- Each set has a representative member.
- Supports functions:
 void MakeSet(const T & k);
 void Union(const T & k1, const T & k2);
 T & Find(const T & k);

A first data structure for Disjoint Sets:



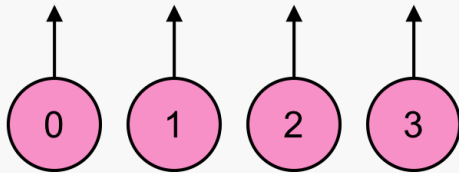
0	1	2	3	4	5	6	7
0	0	2	3	0	3	3	2

Find:

Union:

A better data structure for Disjoint Sets: UpTrees

- if array value is -1, then we've found a root, o/w value is index of parent.
- x and y are in the same tree iff they are in the same set.



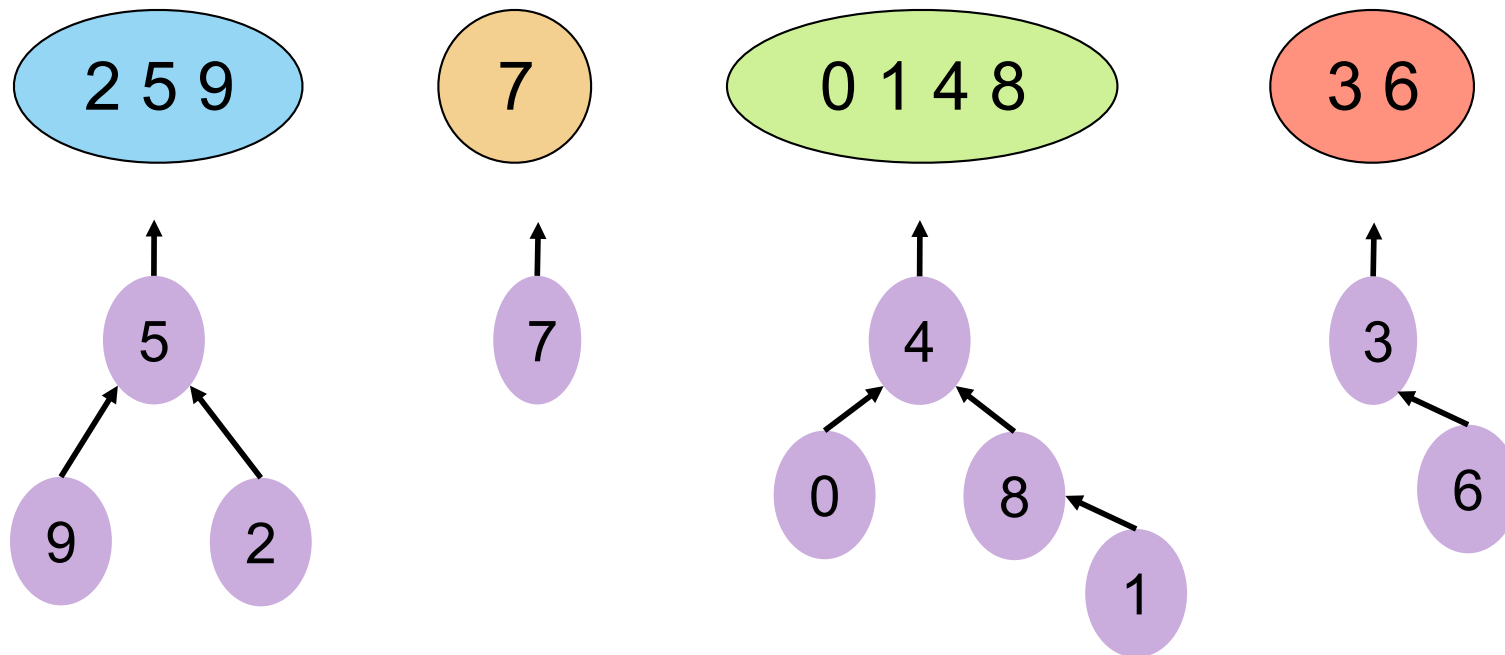
0	1	2	3
-1	-1	-1	-1

0	1	2	3

0	1	2	3

0	1	2	3

Let R be an equivalence relation on the set of students in this room, where $(s, t) \in R$ if s and t have the same favorite among $\{AB, FN, DJ, ZH, FB\}$.



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

A better data structure for Disjoint Sets:

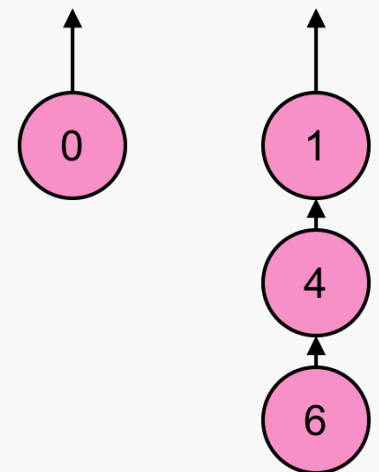
```
int DS::Find(int i) {  
    if (s[i] < 0) return i;  
    else return Find(s[i]);  
}
```

Running time depends on _____.

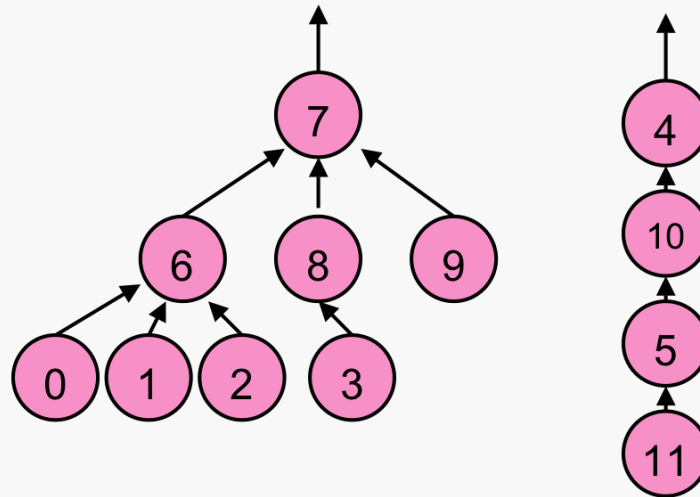
Worst case?

What's an ideal tree?

```
void DS::Union(int root1, int root2) {  
    _____;  
}
```



Smart unions:



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

Keeps overall height of 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

Increases distance to root for fewest nodes.

Both of these schemes for Union guarantee the height of the tree is _____.

Smart unions:

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

```
void DS::UnionBySize(int root1, int root2) {  
    int newSize = s[root1]+s[root2];  
    if (isBigger(root1,root2)) {  
        s[root2]= root1;  
        s[root1]= newSize;  
    }  
    else {  
        s[root1] = root2;  
        s[root2]= newSize;  
    }  
}
```

Path Compression:

