Formalization: Disjoint Set ADT

`Find(x)`:

- Returns the name of the set containing x.
  - Name can be anything as it long as it uniquely identifies the set and everyone agrees to that name
- Ex: Given sets
  - **Find(10)** returns 1
  - **Find(4)** returns 2

Group 1	Group 2	Group 3
3, 5, 10, 13	2, 4, 8	9

Find Runtime:
amortized O(1)
Worst case O(log n)

`Union(group1, group2)`:

- Takes the union of two sets named `group1` and `group2`
- Ex: Given the above sets
  - **Union(2, 3)** =>

Group 1	Group 2
3, 5, 10, 13	2, 4, 8, 9

Union Runtime:
Worst case O(1)

Why we need a new datastructure?

Let's try to implement the Disjoint Set ADT using a familiar datastructure: HashMap! *(assume no hash collisions)*

Group 1	Group 2
A D	B C

Approach 1: Value -> Set ID

A	1
B	2
C	2
D	1

Approach 2: Set ID -> Values (LinkedList)

1	[A, D]
2	[B, C]

n elements	Find()	Union()
Approach 1		
Approach 2		

Implementation

```
int Find(int x) {

  while (up[x] != 0) {
    x = up[x];
  }

  return x;
}
```

```
void Union(int x, int y) {
  up[y] = x
}
```

Worst runtime for Union():

Worst runtime for Find():

Overall Runtime:
(*m* finds and ≤ *n-1* unions)

Goal: *O(m + n)*

Try it yourself

Draw result of PC-Find(4)