

Formalization: Disjoint Set ADT

Find(x):

- Returns the name of the set containing x.
 - Name can be anything as 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, <u>10</u> , 13	2, <u>4</u> , 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, <u>10</u> , 13	2, <u>4</u> , 8, 9

Union Runtime:

Worst case **$O(1)$**

5/27/2022

17

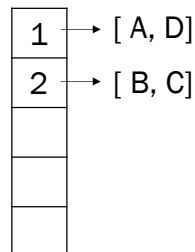
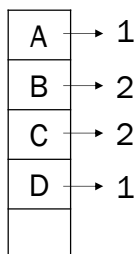
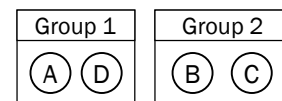
17

Why we need a new datastructure?

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

Approach 1: Value -> Set ID

Approach 2: Set ID -> Values (LinkedList)



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

5/27/2022

20

20

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 $\leq n-1$ unions)

Goal: $O(m + n)$

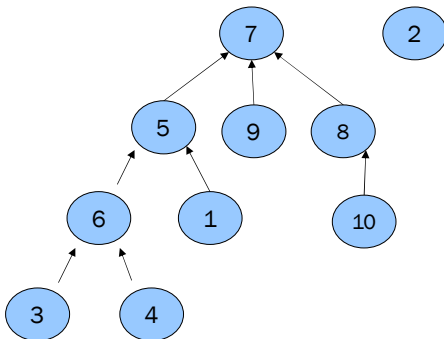
5/27/2022

32

32

Try it yourself

Draw result of PC-Find(4)



5/27/2022

52

52