Union Find

Union Find Connected Components Algorithm and the Union Find Data Structure made with linked-lists

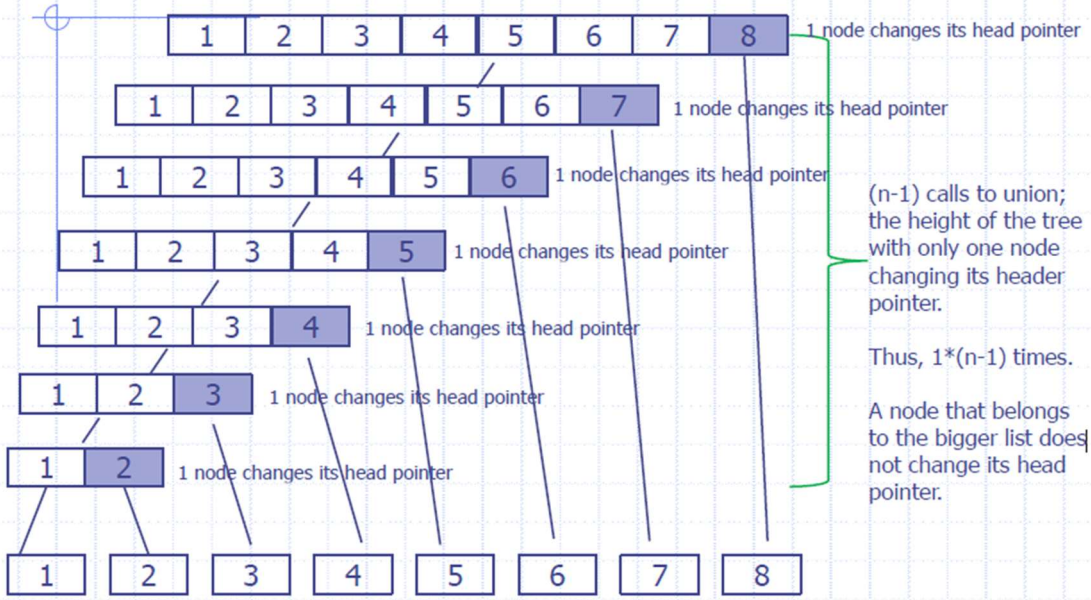
- operations (a.k.a. functions)

```
Algorithm MakeSet(e):
    Create a link list for e, and make e as representative
Algorithm Find(e):
    return e.head
Algorithm Union(A, B):
    if |A| >= |B| then
        for each x in Link list B do
            add x into A
            x.head <- A's repensentative
    else
        for each x in Link list A do
            add x into B
            x.head <- B's representative
```

- runtime analysis for different operations

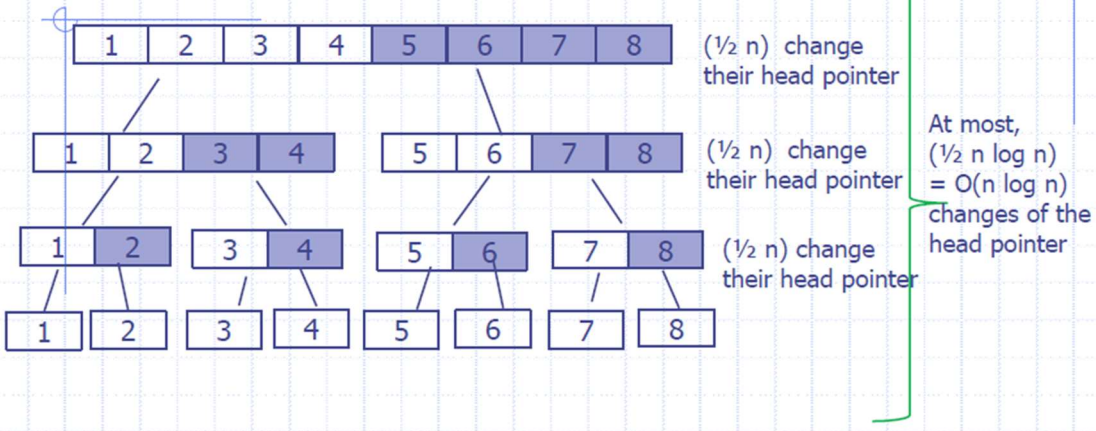
```
Algorithm MakeSet(e): ------------ O(1)
    Create a link list for e, and make e as representative


Algorithm Find(e): -------------O(1)
    return e.head
Algorithm Union(A, B): ------------O(min{|A|, |B|})
    if |A| >= |B| then
        for each x in Link list B do
            add x into A
            x.head <- A's repensentative
    else
        for each x in Link list A do
            add x into B
            x.head <- B's representative
-------------------------------------------------------------------------------
If |A| + |B| = n then Union(A, B) runtime in worst case |A| = |B|, O(n/2) ~ O(n).
-------------------------------------------------------------------------------
```

Largest Number of Calls to Union Until Only 1 Component Exists



Slide by Dianne Foreback

Worst Case Number of Times a Node Changes it Head Pointer Until Only One Component Exists



- runtime for creating the data structure

Theorem 7.1: *Performing a sequence, σ, of m* union *and* find *operations, starting from n singleton sets, using the above list-based implementation of a union-find structure, takes $O(n \log n + m)$ time.*

Theorem 7.2: *Using a list-based implementation of a union-find structure, in a series of* makeSet, union, *and* find *operations, involving a total of n initially singleton sets, the amortized running of each* union *operation is $O(\log n)$ and the amortized running time for each* makeSet *and* find *operation is $O(1)$.*