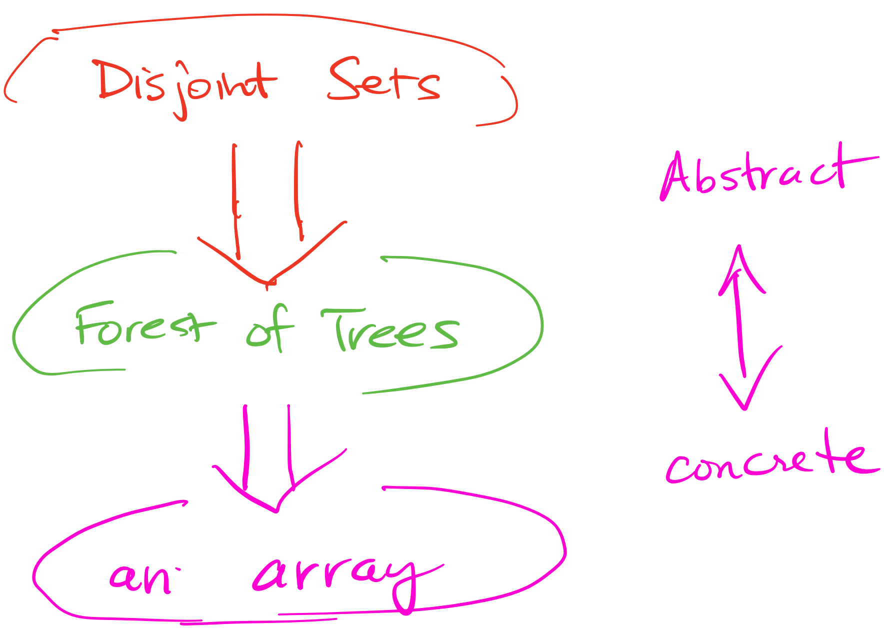
Disjoint Sets [Union-Find] Forests

Trade-off: Quick-Find or Quick-Union?

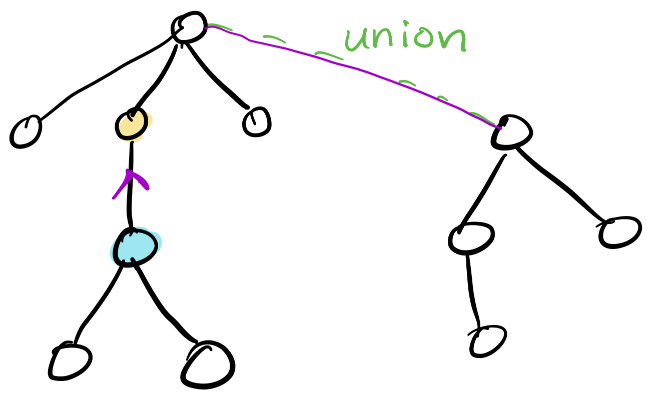
Recall:

- logically (abstractly) we are representing sets as <u>trees</u> in a <u>forest</u>

- <u>implementation</u> of forest: an array.

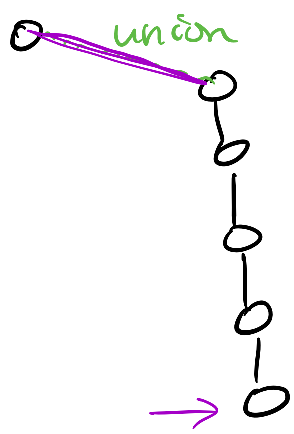


Our implementation thus far in the lab is:



If no weigh/height/size heuristic is used:

in worst case:



- can have $\Theta(n)$ height trees
- what is the worst case running time of Find(x)? $\Theta(n)$
- what is the worst-case running time of Union(x,y)?

a) if x + y are roots $\Theta(1)$

b) if x, y could be any elements in the forest.

$\Theta(n)$

In the above, the great amount of work is done in the <u>Find</u> op's:

<u>Union</u> ops are quick ( ) once the Find ops within them are completed.

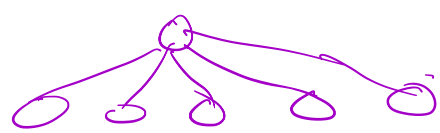
$\Rightarrow$ we call that implementation "Quick Union".

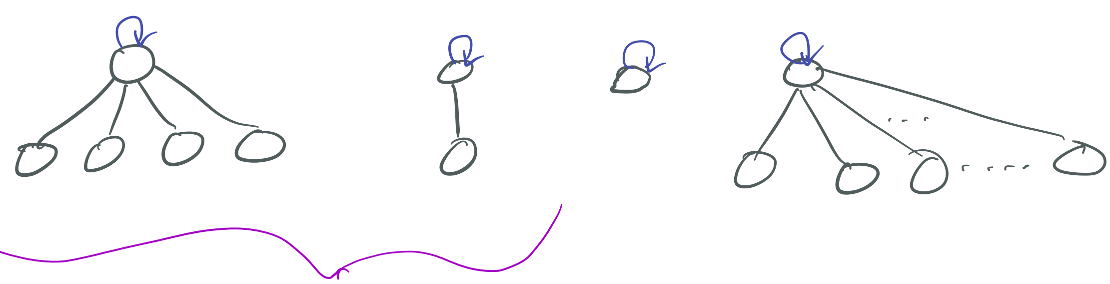
Quick-Find Implementation:

Let's make <u>Find</u> do less work.

[Trade-off: <u>Union</u> will have to do more work]

What kind of trees make <u>Find</u> ops quick?





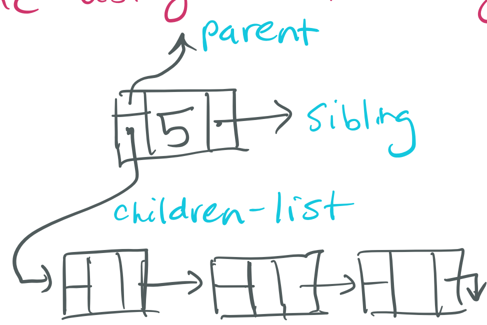
depth = 1 trees

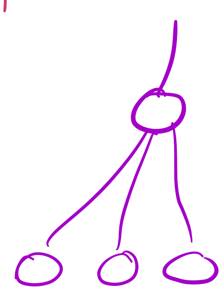
QUICK-FIND

```
Find (x)
    return parent[x]
```

```
Union (x, y)
    p = x -> parent
    q = y -> parent
    if (p -> numchildren < q -> numchildren)
        swap p and q
    for each child c of q
        c -> parent = p
    q -> parent = p
```

[Best done using the following representation of a tree:

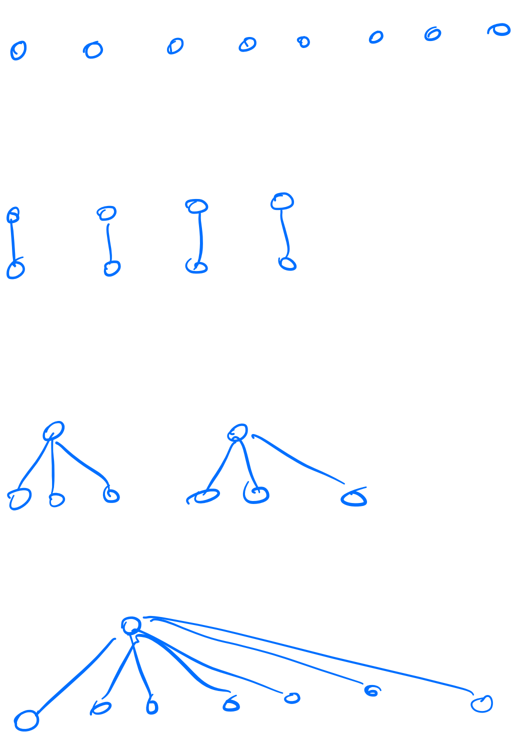




]

What is the running time for Union?

- worst case: trees are equal in size when they are unioned.



8 trees

4 trees

2 trees

1 tree

There are lg n stages, and in each stage, HALF of the nodes ($\approx \frac{n}{2}$) must have their parent pointers reassigned $\therefore$ a total of $\Theta(n \lg n)$ work

Claim: The work done in the worst case is $\Theta(\lg n)$ amortized per union

Proof: There are lg n <u>stages</u>, where at each stage we go from 2^k trees of size $\frac{n}{2}$

to 2^{k-1} trees of size $\frac{n}{2^{k-1}}$, for $k = \lg n$ down to 0. 2^{π}

Each stage reassigns $\frac{n}{2}$ parent pointers; thus the work is $\Theta(n \lg n)$.

The number of ops in all these stages is $\frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{\lg n}} = n \cdot \sum_{i=1}^{\lg n} \frac{1}{2^i}$

$\in \Theta(n)$

$\therefore$ amortized running time is $\frac{\Theta(n \lg n)}{\Theta(n)} = \Theta(\lg n)$

Hence $O(\lg n)$ per op. (amortized)

Heuristics to improve running time:

1. Union-by-rank:

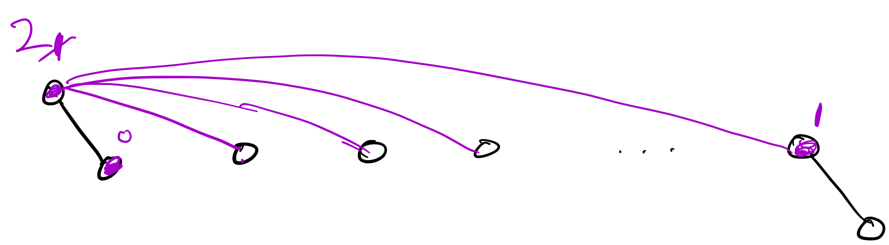
"rank" of x is approximately

$\approx \log_2(\text{size of tree rooted at } x)$

$\leq$ height of subtree rooted at x

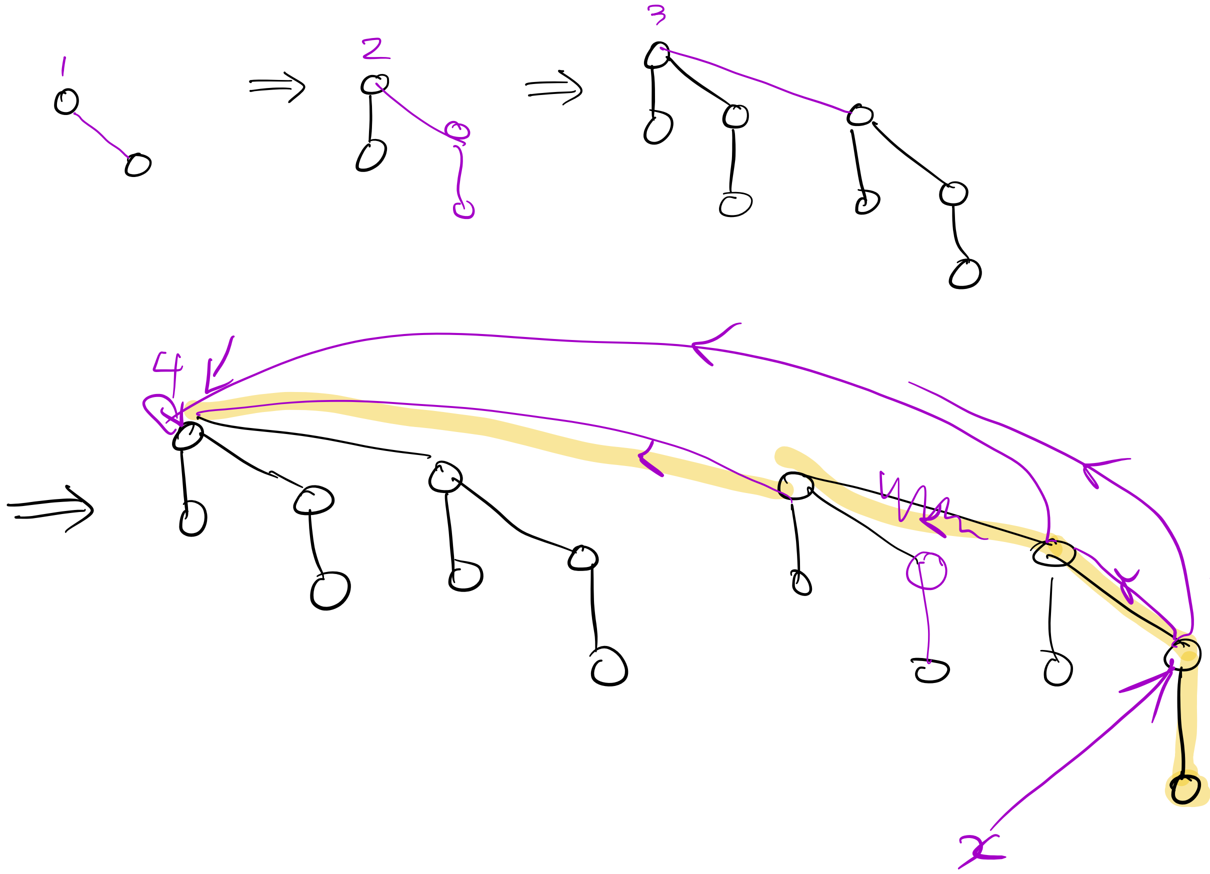
note: we do not keep exact record of height or number of nodes

Here's what we do instead:



we are parsimonious with rank — we only

increase it if we union two trees of equal rank.



2. Path Compression.

Let's do some helpful house-keeping while we climb the tree during a FIND.

Find (x)
  while (parent[x] != x)
    x = parent[x]
  return x
	Find (x)
	  if (parent [x] != x)
	    parent [x] = Find (parent [x])
	  return parent [x]

Worst case running time of Quick-Union Union Find, using Path compression and union by rank:

In the scope of this course, we will be able to show that the running time is $\lg^* n$ amortized time (per operation)

$\lg^* n$ = number of times you take the lg before getting to ≤ 1

$\lg^* 1 = 0$

$\lg^* 2 = 1$

$\lg^* 3 \; .. \; \lg^* 4 = 2$

$\ldots \lg^* 16 = 3$ $\quad 2^4 = 16, \; 2^2 = 4, \; 2^1 = 2$

$\lg^* 65{,}536 = 4$ $\quad 2^{16} = 65{,}536, \; 2^4 = 16, \; 2^2 = 4,$

$\lg^* 2^{65{,}536} = 5$

Super-super slow growing function.

$\lg^*(\#\text{ atoms in universe}) \leq 5$

$\lg^* n$ essentially acts like a constant in most conceivable practical circumstances.

Re Assignment 1:

what happens when increase n by 1?
by n (double it)?

if running time is

d) $\Theta(n^3)$

inc by 1

eg $\underline{c \cdot n^3} \Rightarrow c \cdot (n+1)^3$

$(n+1)(n+1)(n+1)$

$(n^2+2n+1)(n+1)$

ie $c(n^3 + \boxed{3n^2 + 3n + 1})$

what if n is doubled?

$(2n)^3 = 8n^3$