Examples:
- Given prime \( p \), \( a \equiv b \mod p \)
  
  \( \text{Eg. } p=5 \); Partition: \{0,5,10...,11,16,...\} \{2,7,12,...\}...
- Given graph \( G \), vertices \( u,v \),
  
  \( u \equiv v \) if in same connected component

Partition: \{2,8,10\}, \{1,3,5,9\}, \{4\}, \{6,7\}

Union-Find:
Given set \( S = \{1,2,...,n\} \) maintain a partition supporting ops:
Init: Each element in its own set \{13, 23, ..., \[n\}\}
Union(s,t): Merge two sets \( s \cup t \) and replace with their union
Find(x): Return the set containing \( x \)

Example: Suppose: \{1,5\}, \{2,6,8\}, \{3,4,7\}
\( S_1, S_2, S_3 \)

Union \( S_1, S_2 \) \( \rightarrow \{1,3,4,5,7\}, \{2,6,8\} \)
Find \( (s) \) \( \rightarrow \{s\} \)

Eg. \( \{3,7,10\} \) \( \{1,3,4,7\} \{2,5,6,8,11\} \{4,9\} \)

Equivalence Relation:
Binary relation over set \( S \) such that \( \forall a, b, c \in S \):
  - reflexive: \( a \equiv a \)
  - symmetric: \( a \equiv b \Rightarrow b \equiv a \)
  - transitive: \( a \equiv b \Rightarrow b \equiv c \Rightarrow a \equiv c \)

Any equivalence relation defines a partition over \( S \).

A simple approach to finding is to trace the path to the root:
Set \( \text{Simple-Find} (\text{Element } x) \{
  \text{Set } \text{Element } \equiv \text{int 1} \leq x \leq n
  \}
  \text{while } \left( \text{parent}[x] \neq \text{null} \right)
  \text{x } \leftarrow \text{parent}[x]
  \text{return x}

Set Identifiers are indices of root nodes

Eg. \( \text{Find}(7)=3 \)
\( \text{Find}(10)=3 \)
\( \text{Find}(5)=2 \)

Union-Find I

Inverted-Tree Approach:
- Store elements of each set in tree with links to parent
- Root node is set identifier

Array-Based Implementation:
Assume: \( S = \{1,2,...,n\} \)
parent \( [1..n] \), where parent[\( i \)] is is parent index or \( O=\text{null} \) if root
**Set Union (Set s, Set t)**

if (rank[s] > rank[t])

- Just link one tree under the other

- How to maintain low ranks?

**Lemma:** Assuming rank-based merging of a tree of height h has at least $2^h$ nodes.

**Proof:** By induction on num. of unions

**Basis:** Single node. $h = 0$, $2^0 = 1$ node

**Step:** Consider the last of series of unions. Let $T' + T''$ be trees to merge. Heights: $h' + h''$

Sizes: $n' + n''$

By induction: $n' \geq 2^{h'}$ $n'' > 2^{h''}$

**Cases:**

1. $h' = h''$
2. $h' < h''$
3. $h' > h''$ (symmetrical)

**Example:**

- Set 1: $\{1, 3, 7, 10\}$
- Set 2: $\{13, 2, 5, 6, 8, 11\}$
- Set 3: $\{4, 9\}$

**Disjoint Set Union - Find II**

**Running Time?**

- **Init:** $O(n)$ - set a parents to null + ranks to 0
- **Union:** $O(1)$ - constant time
- **Find:** $O(\text{tree height})$

We'll show tree height $= \mathcal{O}(\log n)$

**Final tree height:** $h = h' + 1 = h'' + 1$

**Final size:**

$\begin{align*}
    n &= n' + n'' \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h-1} = 2^h
\end{align*}$

**Case 2:** $h' < h''$

Final height: $h = h''$

Final size:

$\begin{align*}
    n &= n' + n'' \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h-1} = 2^h
\end{align*}$

**Case 3:** $h' > h''$ (symmetrical)

Final height: $h = h'$

Final size:

$\begin{align*}
    n &= n' + n'' \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h'} + 2^{h''} \\
    &= 2^{h-1} = 2^h
\end{align*}$
Path Compression:
- Whenever we perform find, short-cut the links so they point directly to root
- This does not increase running time more than constant, but can speed up later finds

Simple Union-Find performs a sequence of m Unions + Finds on set of size n in \( O(m \log m) \) time.
\[ \Rightarrow \text{Amortized time (average per op)} = \mathcal{O}(\log m) \]
- Not bad - But can we do better?

Example:

```
3
2
1
```

Disjoint Set Union - Find

Disregression: Ackerman's Function (1926)

\[
A(i,j) = \begin{cases} 
  j+1 & \text{if } i = 0 \\
  A(i-1,1) & \text{if } i > 0, j = 0 \\
  A(i-1,A(i,j-1)) & \text{otherwise}
\end{cases}
\]


Thorem: (Tarjan 1975) After init.
any seq of m Union-Finds (with path compression) takes total time \( O(m^\alpha(m,n)) \).
\[ \Rightarrow \text{Amortized time} = \mathcal{O}(\alpha(m,n)) \]
[For all practical purposes, this is constant time.

From super big to super small

Inverse of Ackerman:

\[
\alpha(m,n) = \min \{ i \geq 1 \mid A(i, \lceil m/n \rceil) > \log m \}
\]

Obs: \( \alpha(m,n) \leq 4 \) for any imaginable values of \( m, n \) (\( m \geq n \))

Digression: Ackerman's Function

\[ A(i,0) = i \]
\[ A(i,1) = 2^i - 1 \]
\[ A(i,j) = A(i-1,A(i,j-1)) \]