**Disjoint Sets**

**Disjoint-Set-Data-Structure:** maintains a collection \( \{S_1, S_2, \ldots, S_k\} \) of dynamic disjoint sets. Each set is identified by a **representative**, which is a member of the set.

Let \( x \) be an object:

**MAKE-SET(\( x \)):** makes a **singleton set** with \( x \).

**UNION(\( x, y \)):** takes two disjoint sets with representatives \( x \) and \( y \) respectively and creates their union. Usually, either \( x \) or \( y \) will be the new representative. The old sets are destroyed.

**FIND-SET(\( x \)):** returns a pointer to the representative of the unique set containing \( x \).
Disjoint Sets

Analysis of running time in terms of two parameters:
• $n$: number of MAKE-SET operations
• $m$: total number of MAKE-SET, UNION and FIND-SET operations.

Each UNION operation reduces the number of sets by 1 - recall sets are all disjoint. Maximum: $n - 1$ UNIONS.
Also $m \geq n$. Assume that the $n$ MAKE-SET operations are the first performed.

Example of use: Connected Components in an undirected graph, Min-Spanning Tree (Forest), etc.
Disjoint Sets

**CONNECTED-COMPONENTS**\((G)\)

1. for each vertex \(v \in V[G]\)
2. do **MAKE-SET**\((v)\)
3. for each edge \((u, v) \in E[G]\)
4. do if **FIND-SET**\((u) \neq **FIND-SET**\((v)\)
5. then **UNION**\((u, v)\)

**SAME-COMPONENT**\((u, v)\)

1. if **FIND-SET**\((u) = **FIND-SET**\((v)\)
2. then return **TRUE**
3. else return **FALSE**
### Disjoint Sets

(a)

(b)

<table>
<thead>
<tr>
<th>Edge processed</th>
<th>Collection of disjoint sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial sets</td>
<td>{a}</td>
</tr>
<tr>
<td>(b,d)</td>
<td>{a}</td>
</tr>
<tr>
<td>(e,g)</td>
<td>{a}</td>
</tr>
<tr>
<td>(a,c)</td>
<td>{a,c}</td>
</tr>
<tr>
<td>(h,i)</td>
<td>{a,c}</td>
</tr>
<tr>
<td>(a,b)</td>
<td>{a,b,c,d}</td>
</tr>
<tr>
<td>(e,f)</td>
<td>{a,b,c,d}</td>
</tr>
<tr>
<td>(b,c)</td>
<td>{a,b,c,d}</td>
</tr>
</tbody>
</table>
Disjoint Sets

First Implementation: Linked Lists with head and tail pointers.

• MAKE-SET would be fairly efficient, no matter what, with $O(1)$ time complexity.

• FIND-SET would be messy, leading to the introduction of a “head pointer” associated with each node: this would point to the first element of the list, which would be the representative. This would make FIND-SET an $O(1)$ operation.

• UNION would still require work: $\Omega(n)$ - more precisely $\Omega(\min(\text{cardinality of first set, cardinality of second set}))$ - see next slide for details.
Disjoint Sets
## Disjoint Sets

### Cost: $\Theta(n^2)$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of objects updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-SET($x_1$)</td>
<td>1</td>
</tr>
<tr>
<td>MAKE-SET($x_2$)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>MAKE-SET($x_n$)</td>
<td>1</td>
</tr>
<tr>
<td>UNION($x_1, x_2$)</td>
<td>1</td>
</tr>
<tr>
<td>UNION($x_2, x_3$)</td>
<td>2</td>
</tr>
<tr>
<td>UNION($x_3, x_4$)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>UNION($x_{n-1}, x_n$)</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>
Disjoint Sets

**Weighted Union Heuristic**: if each list has a length attribute, we can always add the shorter list to the longer - ties are broken arbitrarily.
A single `UNION` operation can still require $\Omega(n)$ time, if both sets have $\Omega(n)$ members.

**Theorem 21.1.** Using the linked-list representation of disjoint sets and the weighted-union heuristic, a sequence of $m$ `MAKE-SET`, `UNION` and `FIND-SET` operations, $n$ of which are `MAKE-SET` operations, takes $O(m + n \lg n)$ time.
Disjoint Sets

**Proof.** Compute, for each object in a set of size \( n \), an upper bound on the number of times the object’s back pointer to the representative has been updated. Updates occur only when the set containing the object is the smaller of the two. The total number of times the same object can be in the smaller set is \( \lfloor \lg k \rfloor \) over \( k \) update operations, with the set having at least \( k \) members. Since the largest set has at most \( n \) members, the total number of updates for the head pointer of an object must be \( \lfloor \lg n \rfloor \) over all the \texttt{Union} operations. Updating the head and tail pointers of the list costs \( \Theta(1) \) per \texttt{Union} operation. The total time for updating the \( n \) objects is \( O(n \lg n) \).
Each `MAKE-SET` and `FIND-SET` operation takes $O(1)$ time, and there are $O(m)$ of them.

Total time: $O(m + n \lg n)$. Slightly more roughly: $O(m \lg n)$. Can we do better?
Disjoint Sets

Tree Representations.

- **MAKE-SET(x)** creates a tree with one node.
- **FIND-SET(x)** follows parent pointers from $x$ up to the root.
- **UNION(x, y)** causes the root of one tree to point to the root of the other.

(a)  
(b)
Disjoint Sets

Unless we are careful, this will not improve on the naïve implementation via linked lists: the tree could be just a chain of \( n \) nodes.

**We introduce two heuristics:**

1. **Union by Rank:** make the root of the tree with fewer nodes point to the root of the tree with more nodes. Rather than maintaining an exact node count associated with each node, we will maintain a value, called the **rank**: it will be an upper bound on the height of the node. In a **Union**, the root with the smaller rank will point at the root with the larger rank (no rank change). If the ranks are the same then the choice of root pointed to is random, and its rank increases by 1.
2. **Path Compression**: during a `FIND()` operation, compress the path along the chain of nodes so that, at the end, all the nodes in the chain point directly to the root.

Path Compression does not change any ranks.

See the next slide for a pictorial representation.
Disjoint Sets

Effect of $\text{FIND-SET}(a)$:
Disjoint Sets

Pseudocode for disjoint-set forests.

MAKE-SET(x)
1 \( x.p = x \)
2 \( x.rank = 0 \)

UNION(x, y)
1 \( \text{LINK(FIND-SET}(x), \text{FIND-SET}(y)) \)

LINK(x, y)
1 \( \text{if } x.rank > y.rank \)
2 \( \quad y.p = x \)
3 \( \text{else } x.p = y \)
4 \( \quad \text{if } x.rank == y.rank \)
5 \( \quad y.rank = y.rank + 1 \)

The FIND-SET procedure with path compression is quite simple:

FIND-SET(x)
1 \( \text{if } x \neq x.p \)
2 \( \quad x.p = \text{FIND-SET}(x.p) \)
3 \( \text{return } x.p \)
Disjoint Sets

**Union by rank.** We can show that it yields a time of $O(m \lg n)$. You would start by proving that each node has rank at most $\lfloor \lg n \rfloor$. You just need to show that the **UNION** construction increases the rank only when the size of the tree “doubles” - not too hard to set up an induction.

Can we do better? The answer is yes, where the $\lg n$ term is replaced by a function that grows as the “inverse” of the **ACKERMAN** function - so slowly that the multiplier will never be greater than 5 even if we use all the elementary particles in the universe to store bits.
Analysis of union by rank with path compression.

Define the function \((k \geq 0, j \geq 1)\):

\[
A_k(j) = \begin{cases} 
  j + 1 & \text{if } k = 0 \\
  A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1
\end{cases}
\]

Where

\[
A_{k-1}^{(0)}(j) = j \\
A_{k-1}^{(i)}(j) = A_{k-1}(A_{k-1}^{(i-1)}(j)) \text{ for } i \geq 1.
\]

**Def.:** \(k\) is the **level** of the function \(A\).

Notice that \(A_k(j)\) strictly increases with both \(j\) and \(k\).

To get an idea of how fast the function increases, we will look at some low levels of \(k\).
Lemma 21.2. For any integer $j \geq 1$, we have
\[ A_1(j) = 2j + 1 \]
Proof. ----

Lemma 21.3. For any integer $j \geq 1$, we have
\[ A_2(j) = 2^{j+1}(j + 1) - 1 \]
Proof. ----

We can compute:
\[ A_3(1) = 2047 \]
\[ A_4(1) = A_3(2)(1) = A_2(2048)(2047) \gg A_2(2047) = 16^{512} \gg 10^{80}. \]
And $A_5(1)$ would meet the size conditions claimed earlier.
We define the “inverse” of the function $A_k(n)$ for any integer $n \geq 0$, by $\alpha(n) = \min\{k : A_k(1) \geq n\}$, which is the lowest level $k$ for which $A_k(1)$ is at least $n$. From the lemmas and the computations we can see:

$$\alpha(n) = \begin{cases} 0 & \text{for } 0 \leq n \leq 2, \\ 1 & \text{for } n = 3, \\ 2 & \text{for } 4 \leq n \leq 7, \\ 3 & \text{for } 8 \leq n \leq 2047, \\ 4 & \text{for } 2048 \leq n \leq A_4(1). \end{cases}$$

We will show that a sequence of $m$ \textsc{Make-Set}, \textsc{Find-Set} and \textsc{Union} operations, of which $n$ are \textsc{Make-Set}, will have a cost $O(m \alpha(n))$. 
Properties of Ranks.

Lemma 21.4. For all nodes $x$, $x.rank \leq x.p.rank$, with strict inequality if $x \neq x.p$. $x.rank$ is initially 0, and increases through time until $x \neq x.p$; from then on $x.rank$ does not change. The value of $x.p.rank$ monotonically increases over time.

Proof. Induction on the number of operations using the implementations given. Ex. 21.4-1.

Corollary 21.5. As we follow the path from any node to the root, the node ranks strictly increase.

Proof. Obvious.
Lemma 21.6. Every node has rank at most $n - 1$.

Proof. Ranks start at 0 and increase only through \texttt{Link} operations. There are at most $n - 1$ \texttt{Union} operations, and thus at most $n - 1$ \texttt{Link} operations. Since each \texttt{Link} either leaves ranks alone or increases a rank by 1, the result follows.

Note: this is a very weak bound. It will be enough for our purposes... One can prove a bound of $[\lg n]$. 

Disjoint Sets
Disjoint Sets

We will find it useful to replace \textsc{Union} by its constituents: two calls to \textsc{Find-Set} and one to \textsc{Link}.

\textbf{Lemma 21.7.} Convert a sequence $S'$ of $m'$ \textsc{Make-Set}, \textsc{Union} and \textsc{Find-Set} operations into a sequence $S$ of $m$ \textsc{Make-Set}, \textsc{Link} and \textsc{Find-Set} operations. If $S$ runs in $O(m \alpha(n))$ time, then $S'$ runs in $O(m \alpha(n))$ time.

\textit{Proof}. Just observe that $m' \leq m \leq 3m'$.

The rest of the proof will be based on the use of a particular potential function.
Disjoint Sets

Let \( q \) denote the number of operations performed; let \( x \) denote a node in the disjoint-set forest.

We define a potential function \( \phi_q(x) \), a function of nodes and number of operations performed. We assume we start with an empty system, so all trees are empty and \( \phi_0(x) = 0 \).

We shall see that \( \phi_q(x) \) is well-defined: it is not obvious that it is...
For the entire forest, we define the function \( \Phi_q = \sum_x \phi_q(x) \), summing over all the nodes of the forest = the potential of the forest after \( q \) operations. The forest is initially empty.

\[
\Phi_0 = 0, \quad \Phi_q \geq 0 \quad \forall q \geq 0.
\]

We introduce two complementary cases:

**Case 1:** after the \( q^{th} \) operation, \( x \) is a tree root, or \( x.rank = 0 \). In either instance, we define \( \phi_q(x) = \alpha(n) \cdot x.rank \). (recall \( n = \) number of MAKE-SET operations).

**Case 2:** after the \( q^{th} \) operation, \( x \) is not a root and \( x.rank \geq 1 \).

We will define two auxiliary functions on \( x \), before defining \( \phi_q(x) \).
Disjoint Sets

**Def.:** \( \text{level}(x) = \max\{k: x.p.rank \geq A_k(x.rank)\} \).

**Claim:** \( 0 \leq \text{level}(x) < \alpha(n) \).

**Proof.** \( x.p.rank \geq x.rank + 1 \), by Lemma 21.4

\[ = A_0(x.rank), \text{ by def. of } A_0(j). \]

implies that \( \text{level}(x) \geq 0 \). For the other inequality

\[ A_{\alpha(n)}(x.rank) \geq A_{\alpha(n)}(1), \text{ by } A_k(j) \text{ strictly increasing in } j \]

\[ \geq n, \text{ by def. of } \alpha(n) \]

\[ > x.p.rank, \text{ by Lemma 21.6} \]

And this implies that \( \text{level}(x) < \alpha(n) \).

**Note:** \( x.p.rank \) increases monotonically over time \( \Rightarrow \)
\( \text{level}(x) \) does too.
Def.: \( \text{iter}(x) = \max\{i : x.p.rank \geq A_{\text{level}(x)}(i)(x.rank)\} \).

\( \text{iter}(x) \) is the largest number of times we can iteratively apply \( A_{\text{level}(x)} \), applied initially to \( x \)'s rank, before we get a value greater than \( x \)'s parent's rank.

Claim: \( 1 \leq \text{iter}(x) \leq x.rank \).

\textbf{Proof.} We have:

\[
x.p.rank \geq A_{\text{level}(x)}(x.rank), \text{ by def. of level}(x) \\
= A_{\text{level}(x)}^{(1)}(x.rank), \text{ by def. of functional iteration}
\]

Thus \( \text{iter}(x) \geq 1 \). For the other half of the inequality:

\[
A_{\text{level}(x)}^{(x.rank + 1)}(x.rank) = A_{\text{level}(x)+1}(x.rank), \text{ by def. of } A_k(j) \geq x.p.rank, \text{ by def. of level}(x)
\]
**Disjoint Sets**

**Note:** because \( x.p.rank \) monotonically increases over time, in order for \( \text{iter}(x) \) to decrease, \( \text{level}(x) \) must increase. As long as \( \text{level}(x) \) remains unchanged, \( \text{iter}(x) \) must either increase or remain unchanged.

We can now proceed with our definition:

\[
\phi_q(x) = \begin{cases} 
\alpha(n) \cdot n.rank & \text{x a root or } n.rank = 0 \\
(\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) & \text{x not a root and } x.rank \geq 1 
\end{cases}
\]

Notice that the definitions of both \( \phi_q(x) \) and \( \Phi_q \) depend only on the conditions of the nodes in the forest at a specific time, and neither on the kinds of operations performed, nor the specific sequence.
Lemma 21.8. For every node $x$ and for all operation counts $q$, we have $0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$.

Proof. The right-hand inequality follows by definition, if $x$ is a root or $x.rank = 0$, and and by the fact that $\text{level}(x)$ and $\text{iter}(x)$ are both non-negative otherwise.

The left-hand inequality also follows by definition if $x$ is a root or $x.rank = 0$. We have some work to do in the other case.

If $x$ is not a root and $x.rank \geq 1$, we maximize $\text{level}(x)$ and $\text{iter}(x)$. Since we already proven $\text{level}(x) < \alpha(n)$ and $\text{iter}(x) \leq x.rank$, we have

$$
\phi_q(x) = (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x)
\geq (\alpha(n) - (\alpha(n) - 1)) \cdot x.rank - x.rank = x.rank - x.rank = 0.
$$
And we have the left-hand inequality. For the right-hand one, signs and inequalities for \(\text{level}(x)\) and \(\text{iter}(x)\) give

\[
\phi_q(x) \leq (\alpha(n) - 0) \cdot x.rank - 1 = \alpha(n) \cdot x.rank - 1
= \alpha(n) \cdot x.rank.
\]
Disjoint Sets

Lemma 21.10. Let $x$ be a node that is not a root, and suppose the $q^{th}$ operation is either a \texttt{LINK} or a \texttt{FIND-SET}. Then, after the $q^{th}$ operation, $\phi_q(x) \leq \phi_{q-1}(x)$. Moreover, if $x.rank \geq 1$ and either $\text{level}(x)$ or $\text{iter}(x)$ changes due to the $q^{th}$ operation, then $\phi_q(x) \leq \phi_{q-1}(x) - 1$. Thus $x$’s potential cannot increase, and, if it has positive rank and either $\text{level}(x)$ or $\text{iter}(x)$ changes, then $x$’s potential drops by at least 1.

Proof. An assumption has been that the $n$ \texttt{MAKE-SET} operations occur at the beginning of the sequence, and we examine the behavior after these are complete: $q > n$. 
Disjoint Sets

Since $x$ is not a root, and $q > n$, neither $x.rank$ nor $\alpha(n)$ change. These two components of the potential formula remain the same. If $x.rank = 0$, we must have $\phi_q(x) = \phi_{q-1}(x) = 0$.

Assume $x.rank \geq 1$. Recall that $\text{level}(x)$ monotonically increases over time. If the $q^{th}$ operation leaves $\text{level}(x)$ unchanged, then $\text{iter}(x)$ either increases or remains unchanged.

1. $\text{level}(x)$, $\text{iter}(x)$ unchanged: $\phi_q(x) = \phi_{q-1}(x)$.
2. $\text{level}(x)$ unchanged, $\text{iter}(x)$ increases: $\text{iter}(x)$ must increase by at least 1, so $\phi_q(x) \leq \phi_{q-1}(x) - 1$.
3. $\text{level}(x)$ increases (by at least 1): the value of $(\alpha(n) - \text{level}(x)) \cdot x.rank$ drops by at least $x.rank$. 
Disjoint Sets

Because $\text{level}(x)$ increased, $\text{iter}(x)$ might drop, but, since $1 \leq \text{iter}(x) \leq x.\text{rank}$, the drop can be no worse than $x.\text{rank} - 1$. When we throw all this into the potential formula we see that, in all cases, $\phi_q(x) \leq \phi_{q-1}(x) - 1$.

We now move to finding the amortized costs for $\text{MAKE-SET}$, $\text{LINK}$ and $\text{FIND-SET}$.

**Lemma 21.11.** The amortized cost of each $\text{MAKE-SET}$ is $O(1)$.

**Proof.** Obvious, since the actual cost is $O(1)$ and $x.\text{rank}$ remains 0 for each $x$, giving us 0 potential change.
Lemma 21.12. The amortized cost of each LINK operation is $O(\alpha(n))$.

Proof. LINK($x, y$). The actual cost is $O(1)$ - from the pseudo-code. Suppose LINK makes $y$ the parent of $x$. The only nodes whose potential may change are $x$, $y$ and the children of $y$ just prior to the LINK. Since the rank of $x$ does not change, neither the rank, nor the level, nor the iter of its children can change. Since $x$ is no longer a root, its level and iter, which depend on the rank of $y = x.p$, could change; similarly for the immediate children of $y$ whose parent may have changed rank. We shall show that the only node whose potential can increase is $y$, and by at most $\alpha(n)$. 

Disjoint Sets
Disjoint Sets

• By Lemma 21.10, any node which is a child of $y$ just before the $\text{LINK}$ cannot have an increase in potential.

• Since $x$ was a root before the $q^{th}$ operation,

\[ \phi_{q-1}(x) = \alpha(n) \cdot x.rank. \]

If $x.rank = 0$, then $\phi_q(x) = \phi_{q-1}(x) = 0$. Otherwise,

\[ \phi_q(x) = (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) < \alpha(n) \cdot x.rank = \phi_{q-1}(x). \]

A decrease.

• Since $y$ was a root before the $q^{th}$ operation,

\[ \phi_{q-1}(y) = \alpha(n) \cdot y.rank. \]

Link leaves $y$ a root. And it either leaves $y.rank$ unchanged or increases it by 1. So, either $\phi_q(y) = \phi_{q-1}(y)$ or $\phi_q(y) = \phi_{q-1}(y) + \alpha(n)$.

Increase: at most $\alpha(n)$; amortized cost: $O(\alpha(n))$. 

9/14/10
Lemma 21.13. The amortized cost of each FIND-SET operation is $O(\alpha(n))$.

Proof. Assume the $q^{th}$ operation is a FIND-SET and the find path has $s$ nodes. Actual cost: $O(s)$. For the amortized cost, we must now find a bound for the change in potential. We shall show that no node’s potential increases due to the FIND-SET, and that at least $\max(0, s - (\alpha(n) + 2))$ nodes on the FIND-SET path have their potential decrease by at least 1.

Lemma 21.10 showed that the potential of nodes other than the root cannot increase. If $x$ is the root, its potential is $\alpha(n) \cdot x.\text{rank}$, which does not change.
Disjoint Sets

Claim: at least $\max(0, s - (\alpha(n) + 2))$ nodes have their potential decrease by at least $1$.

Pf. Let $x$ be a node on the find path s.t.

a. $x.rank > 0$;

b. $x$ is followed somewhere on the FIND path by a node $y$ that is not a root, with $\text{level}(x) = \text{level}(y)$ right before the FIND-SET.

All but at most $\alpha(n) + 2$ nodes $x$ on the FIND-SET path satisfy such conditions. Those that do not satisfy them are the first node on the path (if it has rank $0$), the last node on the path (the root of the tree), and the last node $w$ on the path for which $\text{level}(w) = k$ for each $k = 0, 1, 2, \ldots, \alpha(n) - 1$. 
Disjoint Sets

Fix such a node $x$. Let $k = \text{level}(x) = \text{level}(y)$. Just prior to the path compression of the $\text{FIND-SET}$,

$$x.p.rank \geq A_k^{(\text{iter}(x))}(x.rank) \quad \text{by def. of } \text{iter}(x)$$

$$y.p.rank \geq A_k(y.rank) \quad \text{by def. of } \text{level}(y)$$

$$y.rank \geq x.p.rank \quad \text{by Cor. 21.5 and path pos.}$$

Let $i = \text{iter}(x)$ before path compression. Use the inequalities:

$$y.p.rank \geq A_k(y.rank)$$

$$\geq A_k(x.p.rank) \quad \text{strictly increasing in param.}$$

$$\geq A_k(A_k^{(\text{iter}(x))}(x.rank))$$

$$\geq A_k^{(i+1)}(x.rank).$$
Disjoint Sets

Path compression will make $x.p = y.p$. Therefore, we get $x.p.rank = y.p.rank$. Path compression will not decrease $y.p.rank$, nor will it change $x.rank$. We must have:

$$x.p.rank \geq A_k^{(i+1)}(x.rank).$$

Path compression will cause either $\text{iter}(x)$ to increase (to at least $i + 1$) or $\text{level}(x)$ to increase (if $\text{iter}(x)$ increases to at least $x.rank + 1$). In either case Lemma 21.10 implies that $\phi_q(x) \leq \phi_{q-1}(x) - 1$.

$x$'s potential thus decreases by at least 1.

Since the amortized cost is the actual plus the potential change, we have: actual cost $O(s)$; potential decrease at least $\max(0, s - (\alpha(n) + 2))$.

$$\hat{c} \leq O(s) - (s - (\alpha(n) + 2)) = O(s) - s + O(\alpha(n)) = O(\alpha(n))$$

by judicious use of a scale factor in the potential.
**Theorem 21.14.** A sequence of \( m \) **MAKE-SET**, **UNION** and **FIND-SET** operations, \( n \) of which are **MAKE-SET** operations, can be performed on a disjoint-set forest with union by rank and path compression in worst case time \( O(m \alpha(n)) \).

**Proof.** The previous series of Lemmas.

**Note:** nearly linear, but not quite. Couldn’t get much closer though...

**Question:** could this lead to faster sorting algorithms? Why or why not?