# Have you heard of tetration?

It's the fourth hyperoperation.

# Have you heard of tetration?

It's the fourth hyperoperation.

$$b + n \quad \triangleq \quad b \underbrace{+1+1+\cdots+1}_{n}$$

# Have you heard of tetration?

It's the fourth hyperoperation.

$$
\begin{aligned}
b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\
b \times n &\triangleq \underbrace{b + b + \cdots + b}_{n}
\end{aligned}
$$

# Have you heard of tetration?

It's the fourth hyperoperation.

$$
\begin{aligned}
b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\
b \times n &\triangleq \underbrace{b+b+\cdots+b}_{n} \\
b^n &\triangleq \underbrace{b \times b \times \cdots \times b}_{n}
\end{aligned}
$$

# Have you heard of tetration?

It's the fourth hyperoperation.

$$\begin{aligned}
b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\
b \times n &\triangleq \underbrace{b+b+\cdots+b}_{n} \\
b^n &\triangleq \underbrace{b \times b \times \cdots \times b}_{n} \\
{}^{n}b &\triangleq \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n}
\end{aligned}$$

# Have you heard of tetration?

It's the fourth hyperoperation.

$$
\begin{aligned}
b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\
b \times n &\triangleq \underbrace{b+b+\cdots+b}_{n} \\
b^n &\triangleq \underbrace{b \times b \times \cdots \times b}_{n} \\
{}^n b &\triangleq \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n}
\end{aligned}
$$

Exponentiation *associates to the right,* so for example ${}^4 b$ means $b^{\left(b^{\left(b^b\right)}\right)}$, not $\left(\left(b^b\right)^b\right)^b$.

# Have you heard of tetration?

It's the fourth hyperoperation.

$$\begin{aligned} b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\ b \times n &\triangleq \underbrace{b+b+\cdots+b}_{n} \\ b^n &\triangleq \underbrace{b \times b \times \cdots \times b}_{n} \\ {}^{n}b &\triangleq \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n} \end{aligned}$$

Exponentiation *associates to the right,* so for example ${}^4b$ means $b^{\left(b^{\left(b^b\right)}\right)}$, not $\left(\left(b^b\right)^b\right)^b$. Why?

# Have you heard of tetration?

It's the fourth hyperoperation.

$$\begin{aligned}
b + n &\triangleq b \underbrace{+1+1+\cdots+1}_{n} \\
b \times n &\triangleq \underbrace{b+b+\cdots+b}_{n} \\
b^n &\triangleq \underbrace{b \times b \times \cdots \times b}_{n} \\
{}^{n}b &\triangleq \underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{n}
\end{aligned}$$

Exponentiation *associates to the right,* so for example ${}^{4}b$ means $b^{\left(b^{\left(b^{b}\right)}\right)}$, not $\left(\left(b^{b}\right)^{b}\right)^{b}$. Why? Which is bigger?

# Tetration FAQ

Q: How fast does it grow?

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

# Tetration FAQ

Q: How fast does it grow?     A: Real fast.

Q: Does it have an inverse?

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

# Tetration FAQ

Q: How fast does it grow?   A: Real fast.

Q: Does it have an inverse?   A: Yeah, 2—which do you want?

Q: Is one better than the other?

# Tetration FAQ

Q: How fast does it grow?  A: Real fast.

Q: Does it have an inverse?  A: Yeah, 2—which do you want?

Q: Is one better than the other?  A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | | |
| $b \times n = a$ | | |
| $b^n = a$ | | |
| ${}^n b = a$ | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
| --- | --- | --- |
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | | |
| $b^n = a$ | | |
| ${}^n b = a$ | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
| --- | --- | --- |
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | | |
| ${}^n b = a$ | | |

# Tetration FAQ

Q: How fast does it grow?     A: Real fast.

Q: Does it have an inverse?     A: Yeah, 2—which do you want?

Q: Is one better than the other?     A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}$ | $n = \log_b a$ |
| ${}^n b = a$ | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ... **,** | **and also** ... **.** |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}$ | $n = \log_b a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}$ | $n = \log_b a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^\star a$ |

# Tetration FAQ

Q: How fast does it grow?   A: Real fast.

Q: Does it have an inverse?   A: Yeah, 2—which do you want?

Q: Is one better than the other?   A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}_3$ | $n = \log_b a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^\star a$ |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** … | **then** …, | **and also** …. |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}_3$ | $n = \log_b^3 a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^\star a$ |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | $b = \sqrt[n]{a}_1$ | $n = \log_b^1 a$ |
| $b \times n = a$ | $b = \sqrt[n]{a}_2$ | $n = \log_b^2 a$ |
| $b^n = a$ | $b = \sqrt[n]{a}_3$ | $n = \log_b^3 a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^4 a$ |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $1 + n = a$ | $\cancel{b = \sqrt[n]{a}_0}$ | $n = \log_b^0 a$ |
| $b + n = a$ | $b = \sqrt[n]{a}_1$ | $n = \log_b^1 a$ |
| $b \times n = a$ | $b = \sqrt[n]{a}_2$ | $n = \log_b^2 a$ |
| $b^n = a$ | $b = \sqrt[n]{a}_3$ | $n = \log_b^3 a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^4 a$ |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: Maybe, you decide:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $H_0(b,n) = a$ | $\cancel{b = \sqrt[n]{a}_0}$ | $n = \log_b^0 a$ |
| $H_1(b,n) = a$ | $b = \sqrt[n]{a}_1$ | $n = \log_b^1 a$ |
| $H_2(b,n) = a$ | $b = \sqrt[n]{a}_2$ | $n = \log_b^2 a$ |
| $H_3(b,n) = a$ | $b = \sqrt[n]{a}_3$ | $n = \log_b^3 a$ |
| $H_4(b,n) = a$ | $b = \sqrt[n]{a}_4$ | $n = \log_b^4 a$ |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: One we care about:

| **If** ... | **then** ..., | **and also** .... |
|---|---|---|
| $b + n = a$ | $b = a - n$ | $n = a - b$ |
| $b \times n = a$ | $b = a/n$ | $n = a/b$ |
| $b^n = a$ | $b = \sqrt[n]{a}$ | $n = \log_b a$ |
| ${}^n b = a$ | $b = \sqrt[n]{a}_4$ | $n = \log^\star_b a$ |

$$\log^\star_b a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log^\star_b \log_b a & \text{otherwise.} \end{cases}$$

# Tetration FAQ

Q: How fast does it grow?   A: Real fast.

Q: Does it have an inverse?   A: Yeah, 2—which do you want?

Q: Is one better than the other?   A: One we care about:

$$^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a$$

$$\log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases}$$

Q: Why should we care?

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: One we care about:

$$^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a$$

$$\log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases}$$

Q: Why should we care? A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $^{n}2$ | 1 | 2 | | | | | |

# Tetration FAQ

Q: How fast does it grow?   A: Real fast.

Q: Does it have an inverse?   A: Yeah, 2—which do you want?

Q: Is one better than the other?   A: One we care about:

$$ {}^{n}b = a \quad \Rightarrow \quad n = \log^{\star}_{b} a $$

$$ \log^{\star}_{b} a = \begin{cases} 0 & \text{if } a \leq 1; \\ 1 + \log^{\star}_{b} \log_{b} a & \text{otherwise.} \end{cases} $$

Q: Why should we care?   A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ${}^{n}2$ | 1 | 2 | 4 | | | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: One we care about:

$$^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a$$

$$\log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases}$$

Q: Why should we care? A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $^{n}2$ | 1 | 2 | 4 | 16 | | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: One we care about:

$$ {}^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a $$

$$ \log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases} $$

Q: Why should we care? A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ${}^{n}2$ | 1 | 2 | 4 | 16 | 65,536 | | |

# Tetration FAQ

Q: How fast does it grow? A: Real fast.

Q: Does it have an inverse? A: Yeah, 2—which do you want?

Q: Is one better than the other? A: One we care about:

$$ {}^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a $$

$$ \log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases} $$

Q: Why should we care? A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ${}^{n}2$ | 1 | 2 | 4 | 16 | 65,536 | $2^{65,536}$ | |

# Tetration FAQ

Q: How fast does it grow?     A: Real fast.

Q: Does it have an inverse?     A: Yeah, 2—which do you want?

Q: Is one better than the other?     A: One we care about:

$$
{}^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a
$$

$$
\log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases}
$$

Q: Why should we care?     A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ${}^{n}2$ | 1 | 2 | 4 | 16 | 65,536 | $2^{65,536}$ | $\approx 2^{2.0035\times 10^{19,728}}$ |

# Tetration FAQ

Q: How fast does it grow?     A: Real fast.

Q: Does it have an inverse?     A: Yeah, 2—which do you want?

Q: Is one better than the other?     A: One we care about:

$$ {}^{n}b = a \quad \Rightarrow \quad n = \log_b^\star a $$

$$ \log_b^\star a = \begin{cases} 0 & \text{if } a \le 1; \\ 1 + \log_b^\star \log_b a & \text{otherwise.} \end{cases} $$

Q: Why should we care?     A: Its inverse grows as slow as its self grows fast, and tetration grows real fast:

| $a \le$ | 1 | 2 | 4 | 16 | 65,536 | $2^{65,536}$ | $2^{2.0035\times 10^{19,728}}$ |
|---|---|---|---|---|---|---|---|
| $\lg^\star a$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

# Union-Find

CS 214, Fall 2019

*We're going to use the chalkboard from here on, but if you want union-find slides to read on your own then I suggest these slides from Robert Sedgewick and Kevin Wayne's algorithms & data structures course at Princeton University. I've included a selection of those same union-find slides as the rest of this PDF, and the rest of their original lectures may be found here.*

## Union-find abstractions

- Objects.
- Disjoint sets of objects.
- Find queries: are two objects in the same set?
- Union commands: replace sets containing two items by their union

Goal. Design efficient data structure for union-find.

- Find queries and union commands may be intermixed.
- Number of operations M can be huge.
- Number of objects N can be huge.

# Quick-find [eager approach]

## Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected if they have the same id.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 8 | 9 |

5 and 6 are connected
2, 3, 4, and 9 are connected

# Quick-find [eager approach]

## Data structure.

- Integer array **id[]** of size **N**.
- Interpretation: **p** and **q** are connected if they have the same id.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 8 | 9 |

5 and 6 are connected
2, 3, 4, and 9 are connected

**Find.** Check if **p** and **q** have the same id.

id[3] = 9; id[6] = 6
3 and 6 not connected

**Union.** To merge components containing **p** and **q**, change all entries with `id[p]` to `id[q]`.

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 6 | 6 | 6 | 6 | 6 | 7 | 8 | 6 |

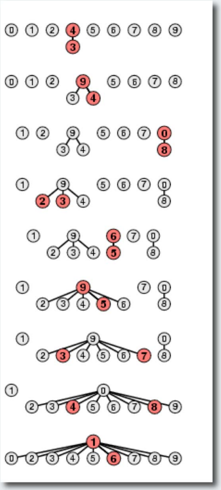
union of 3 and 6
2, 3, 4, 5, 6, and 9 are connected

problem: many values can change

# Quick-find example

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3-4 | 0 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 8 | 9 |
| 4-9 | 0 | 1 | 2 | 9 | 9 | 5 | 6 | 7 | 8 | 9 |
| 8-0 | 0 | 1 | 2 | 9 | 9 | 5 | 6 | 7 | 0 | 9 |
| 2-3 | 0 | 1 | 9 | 9 | 9 | 5 | 6 | 7 | 0 | 9 |
| 5-6 | 0 | 1 | 9 | 9 | 9 | 6 | 6 | 7 | 0 | 9 |
| 5-9 | 0 | 1 | 9 | 9 | 9 | 9 | 9 | 7 | 0 | 9 |
| 7-3 | 0 | 1 | 9 | 9 | 9 | 9 | 9 | 9 | 0 | 9 |
| 4-8 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6-1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

problem: many values can change



# Quick-find is too slow

Quick-find algorithm may take ~MN steps to process M union commands on N objects

Rough standard (for now).

- $10^9$ operations per second.
- $10^9$ words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly) since 1950 !

Ex. Huge problem for quick-find.

- $10^{10}$ edges connecting $10^9$ nodes.
- Quick-find takes more than $10^{19}$ operations.
- 300+ years of computer time!

Paradoxically, quadratic algorithms get worse with newer equipment.

- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!

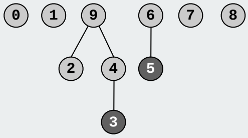
# Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |




3's root is 9; 5's root is 6

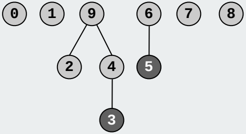
# Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 8 | 9 |



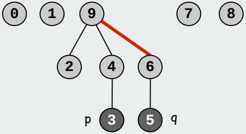

3's root is 9; 5's root is 6
3 and 5 are not connected

Find. Check if p and q have the same root.

Union. Set the id of q's root to the id of p's root.

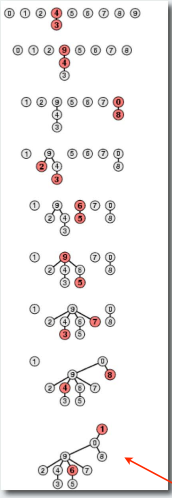
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| id[i] | 0 | 1 | 9 | 4 | 9 | 6 | 9 | 7 | 8 | 9 |

only one value changes



# Quick-union example

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3-4 | 0 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 8 | 9 |
| 4-9 | 0 | 1 | 2 | 4 | 9 | 5 | 6 | 7 | 8 | 9 |
| 8-0 | 0 | 1 | 2 | 4 | 9 | 5 | 6 | 7 | 0 | 9 |
| 2-3 | 0 | 1 | 9 | 4 | 9 | 5 | 6 | 7 | 0 | 9 |
| 5-6 | 0 | 1 | 9 | 4 | 9 | 6 | 6 | 7 | 0 | 9 |
| 5-9 | 0 | 1 | 9 | 4 | 9 | 6 | 9 | 7 | 0 | 9 |
| 7-3 | 0 | 1 | 9 | 4 | 9 | 6 | 9 | 9 | 0 | 9 |
| 4-8 | 0 | 1 | 9 | 4 | 9 | 6 | 9 | 9 | 0 | 0 |
| 6-1 | 1 | 1 | 9 | 4 | 9 | 6 | 9 | 9 | 0 | 0 |



problem: trees can get tall

# Quick-union is also too slow

## Quick-find defect.

- Union too expensive (N steps).
- Trees are flat, but too expensive to keep them flat.

## Quick-union defect.

- Trees can get tall.
- Find too expensive (could be N steps)
- Need to do find to do union

| algorithm | union | find |
|---|---|---|
| Quick-find | N | 1 |
| Quick-union | N* | N ← worst case |

* includes cost of find

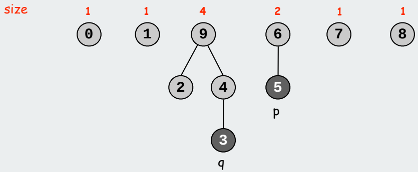
# Improvement 1: Weighting

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each component.
- Balance by linking small tree below large one.

Ex. Union of `5` and `3`.

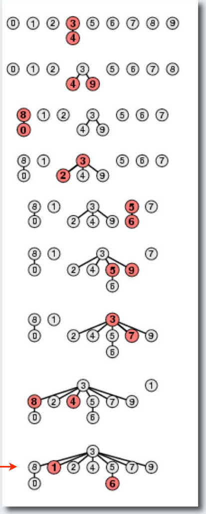
- Quick union: link `9` to `6`.
- Weighted quick union: link `6` to `9`.



# Weighted quick-union example

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3-4 | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 9 |
| 4-9 | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 8-0 | 8 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 2-3 | 8 | 1 | 3 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 5-6 | 8 | 1 | 3 | 3 | 3 | 5 | 5 | 7 | 8 | 3 |
| 5-9 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 7 | 8 | 3 |
| 7-3 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 3 | 8 | 3 |
| 4-8 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 3 | 3 | 3 |
| 6-1 | 8 | 3 | 3 | 3 | 3 | 3 | 5 | 3 | 3 | 3 |

no problem: trees stay flat



# Weighted quick-union: Java implementation

Java implementation.

- Almost identical to quick-union.
- Maintain extra array `sz[]` to count number of elements in the tree rooted at i.

Find. Identical to quick-union.

Union. Modify quick-union to

- merge smaller tree into larger tree
- update the `sz[]` array.

```
if (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else sz[i] < sz[j] { id[j] = i; sz[i] += sz[j]; }
```

# Weighted quick-union analysis

Analysis.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.
- Fact: depth is at most lg N. [needs proof]

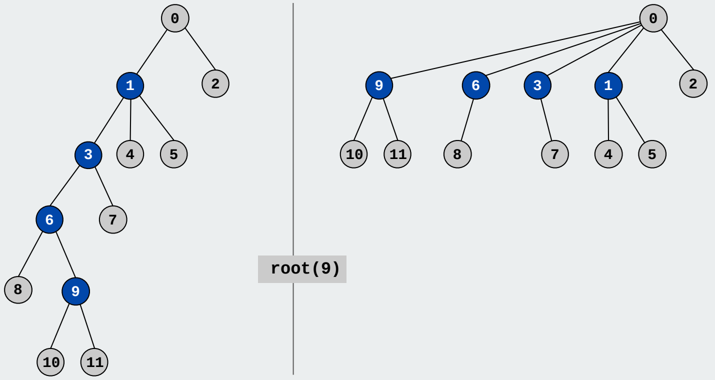
| Data Structure | Union | Find |
|---|---|---|
| Quick-find | N | 1 |
| Quick-union | N * | N |
| Weighted QU | lg N * | lg N |

* includes cost of find

Stop at guaranteed acceptable performance? No, easy to improve further.

## Improvement 2: Path compression

Path compression. Just after computing the root of **`i`**, set the **`id`** of each examined node to **`root(i)`**.



# Weighted quick-union with path compression

## Path compression.

- Standard implementation: add second loop to `root()` to set the id of each examined node to the root.
- Simpler one-pass variant: make every other node in path point to its grandparent.

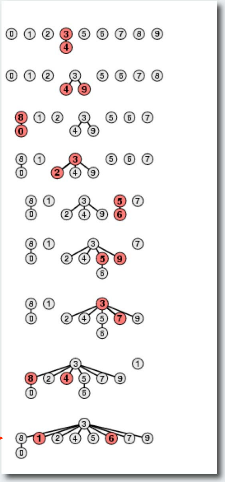
```
public int root(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

In practice. No reason not to! Keeps tree almost completely flat.

# Weighted quick-union with path compression

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3-4 | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 9 |
| 4-9 | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 8-0 | 8 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 2-3 | 8 | 1 | 3 | 3 | 3 | 5 | 6 | 7 | 8 | 3 |
| 5-6 | 8 | 1 | 3 | 3 | 3 | 5 | 5 | 7 | 8 | 3 |
| 5-9 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 7 | 8 | 3 |
| 7-3 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 3 | 8 | 3 |
| 4-8 | 8 | 1 | 3 | 3 | 3 | 3 | 5 | 3 | 3 | 3 |
| 6-1 | 8 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |



no problem: trees stay VERY flat

## WQUPC performance

Theorem. Starting from an empty data structure, any sequence of M union and find operations on N objects takes O(N + M lg* N) time.

- Proof is very difficult.
- But the algorithm is still simple!

number of times needed to take the lg of a number until reaching 1

Linear algorithm?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

because lg* N is a constant in this universe

| N | lg* N |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 4 | 2 |
| 16 | 3 |
| 65536 | 4 |
| 265536 | 5 |

Amazing fact:

- In theory, no linear linking strategy exists

# Summary

| Algorithm | Worst-case time |
|---|---|
| Quick-find | M N |
| Quick-union | M N |
| Weighted QU | N + M log N |
| Path compression | N + M log N |
| Weighted + path | (M + N) lg* N |

M union-find ops on a set of N objects

Ex. Huge practical problem.

- $10^{10}$ edges connecting $10^9$ nodes.
- WQUPC reduces time from 3,000 years to 1 minute.
- Supercomputer won't help much.
- Good algorithm makes solution possible.

WQUPC on Java cell phone beats QF on supercomputer!

Bottom line.

WQUPC makes it possible to solve problems that could not otherwise be addressed