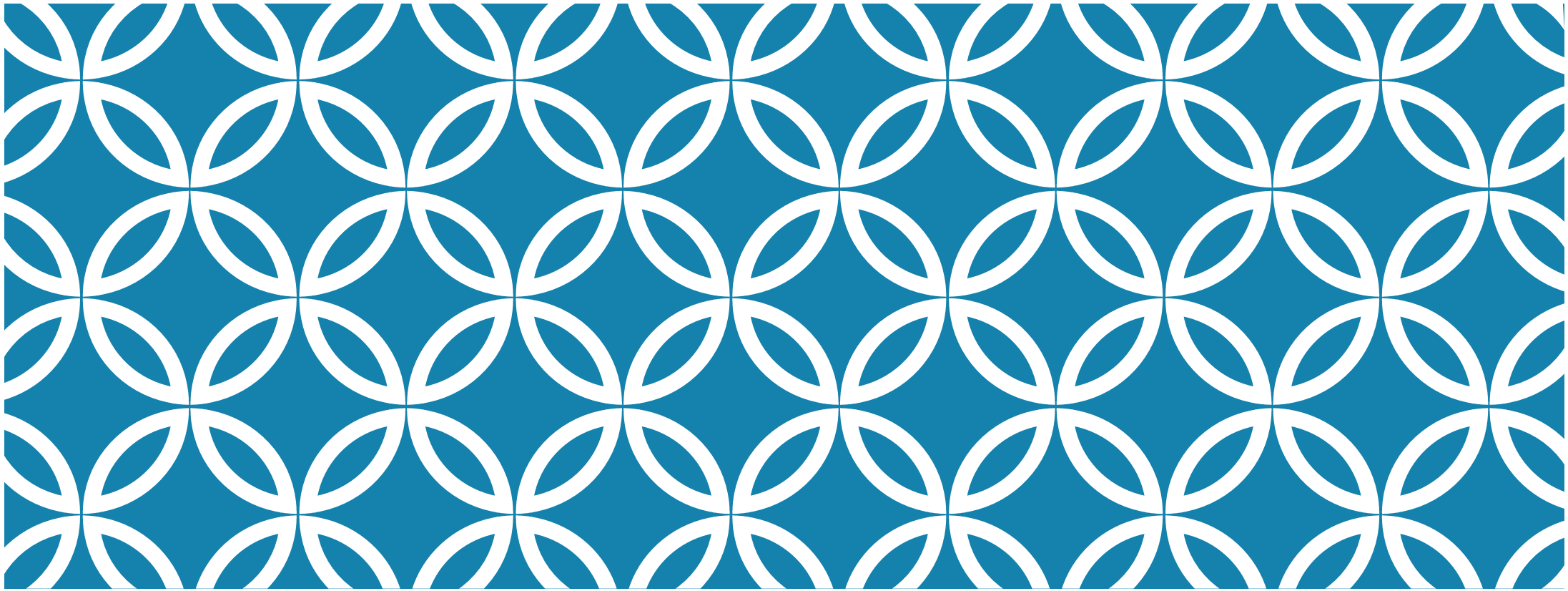


LECTURE 17: KRUSKAL'S ALGORITHM

Harold Soh
harold@comp.nus.edu.sg

ADMINISTRATIVE ISSUES: QUIZ 3

In lecture tomorrow (Wed)

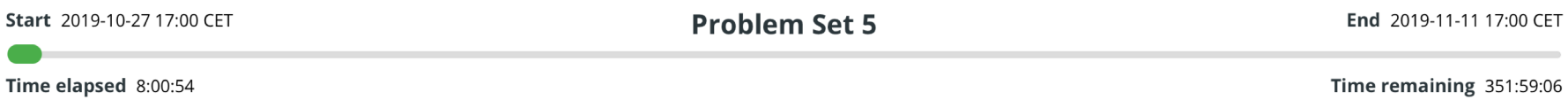
Please come on time!

- Start at 12pm.
- End at 1:30pm.

Open Book

No electronic equipment.

PROBLEM SET 5



Problem Set 5 has been released.

Last problem set! Yay! :D

Due in 2 weeks.

Plagiarism

- No Sharing of Code!

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

REVIEW: PRIM'S ALGORITHM

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

Prim's "grows" the tree one node at a time.

Analysis:

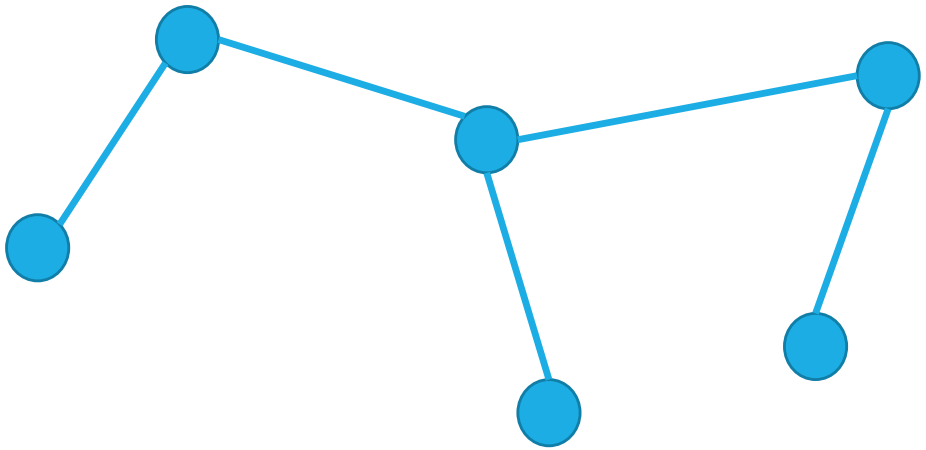
Each vertex added/removed once from the priority queue:

$$O(V \log V)$$

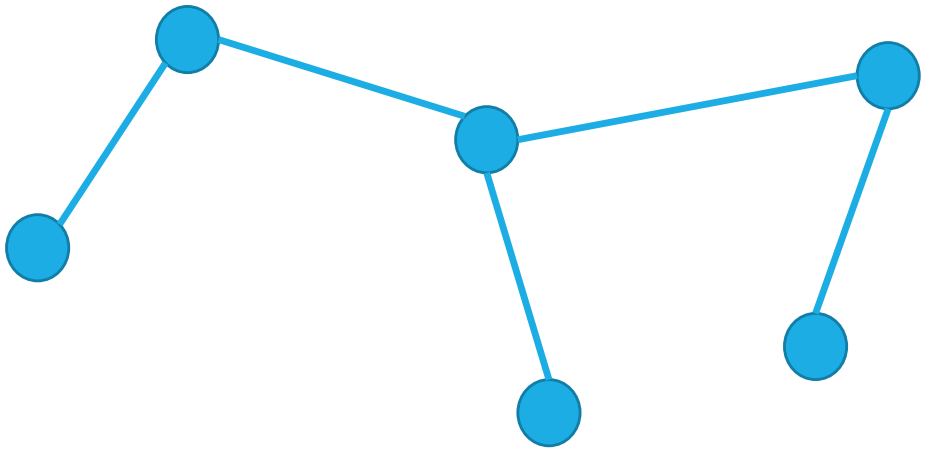
Each edge → one decreaseKey:

$$O(E \log V)$$

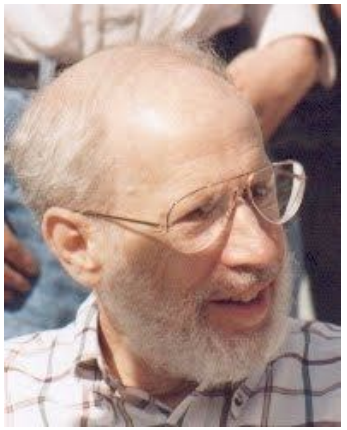
PRIMS ALGORITHM



ANOTHER STRATEGY?



KRUSKAL'S ALGORITHM (Kruskal 1956)



1925-2010

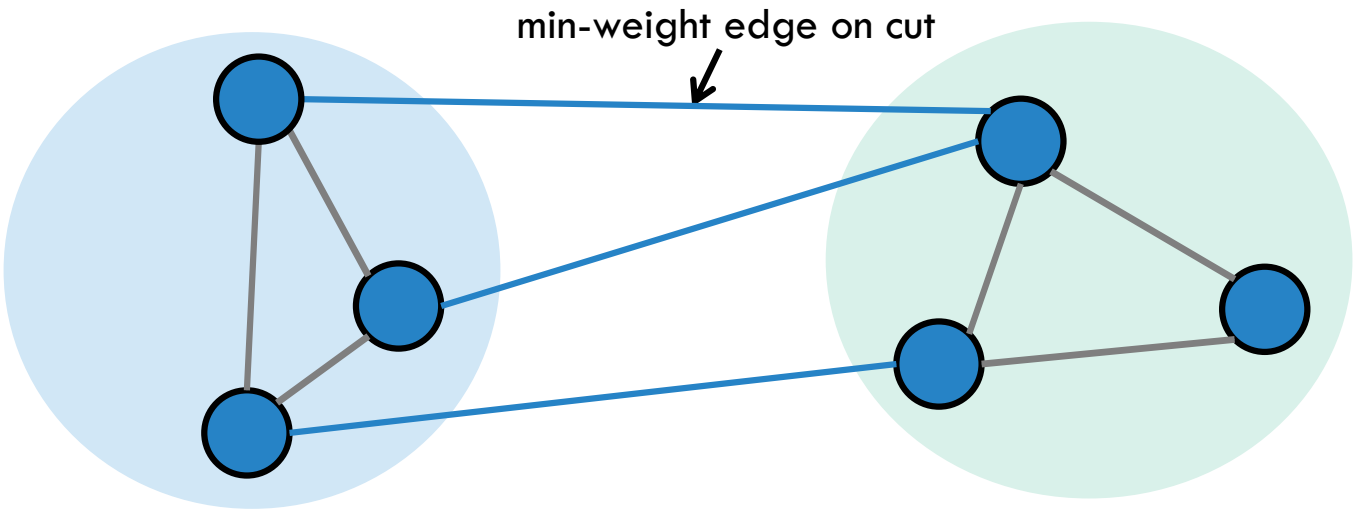
Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
  - Remove minimum weight edge from S
  - If removed edge connects two trees
    - add it to the F (combine the trees)

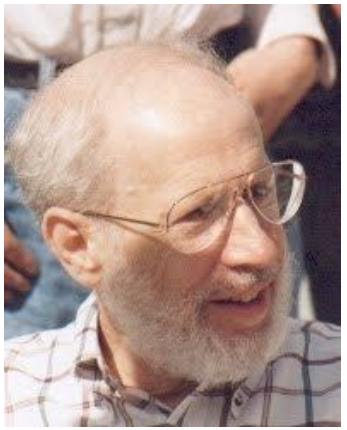
Kruskal "merges" smaller trees into a bigger tree

MST PROPERTIES

Property 4: Cut Property; for every partition of nodes, the **minimum weight edge across the cut** is in the MST.



KRUSKAL'S ALGORITHM (Kruskal 1956)



1925-2010

Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
  - Remove minimum weight edge from S
  - If removed edge connects two trees
    - add it to the F (combine the trees)

How can we quickly find out if an edge connect two trees?
How can we combine trees?

DISJOINT SET (UNION-FIND) ADT

Supports fast:

- find
- union

Idea: Keep each subtree as a set of vertices

public interface	DisjointSet<Key>	
	DisjointSet(int N)	*constructor: N objects*
boolean	find(Key p, Key q)	*are p and q in the same set?*
void	union(Key p, Key q)	*replace sets containing p and q with their union*

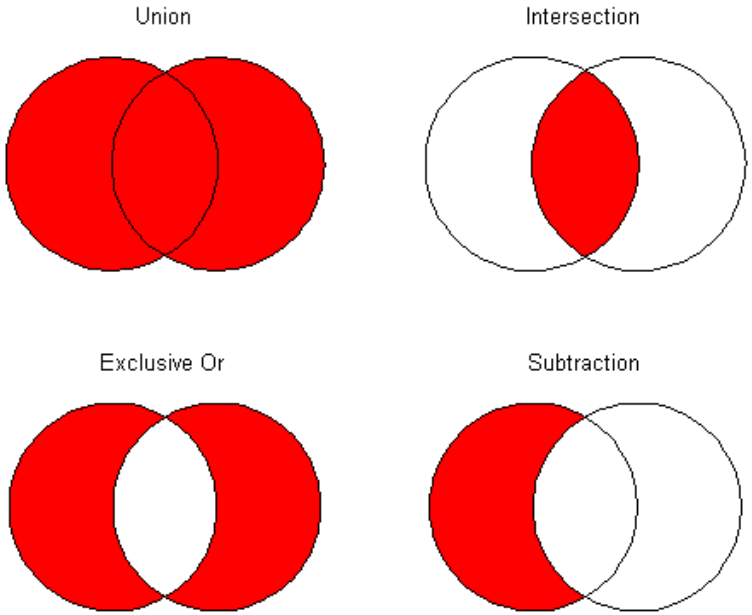
ROADMAP

Two “first-cut” methods:

- Quick find
- Quick Union

Union-Find

- Improving upon the first two approaches.



QUICK FIND

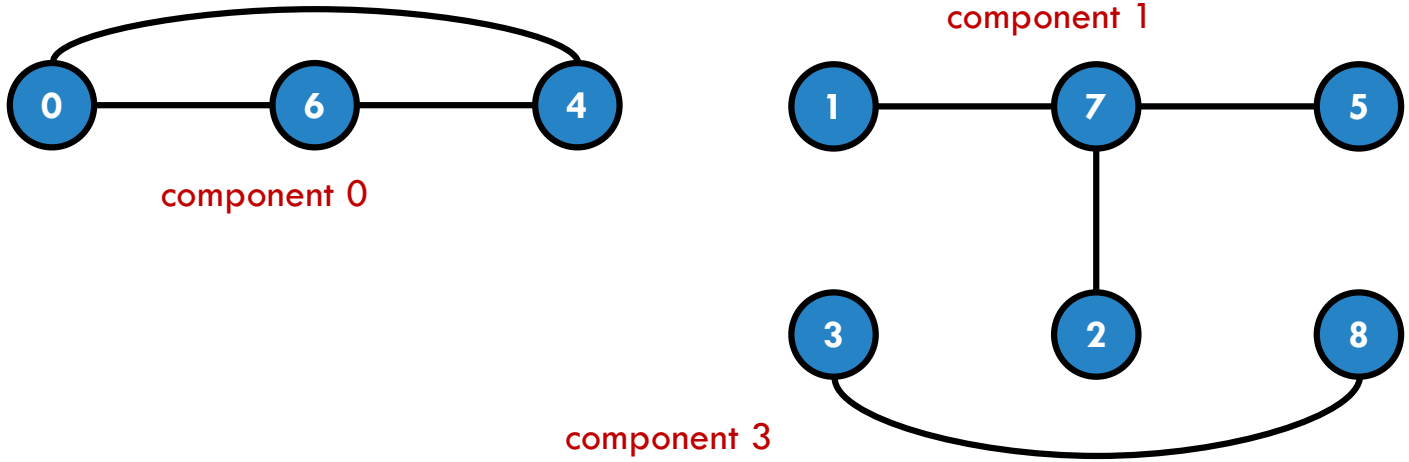
Data structure:

Integer array: `int[] componentId`

Two objects are connected if they have the same component identifier.

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	1	3	0	1	0	1	3

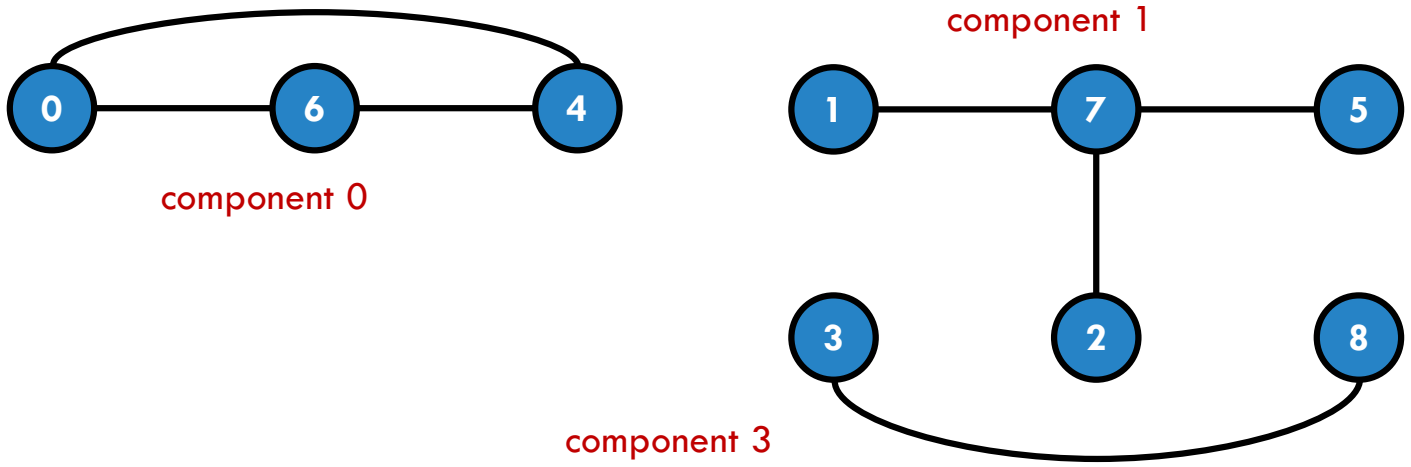
Assume objects are integers



QUICK FIND

```
find(int p, int q)
 return(componentId[p] == componentId[q]);
```

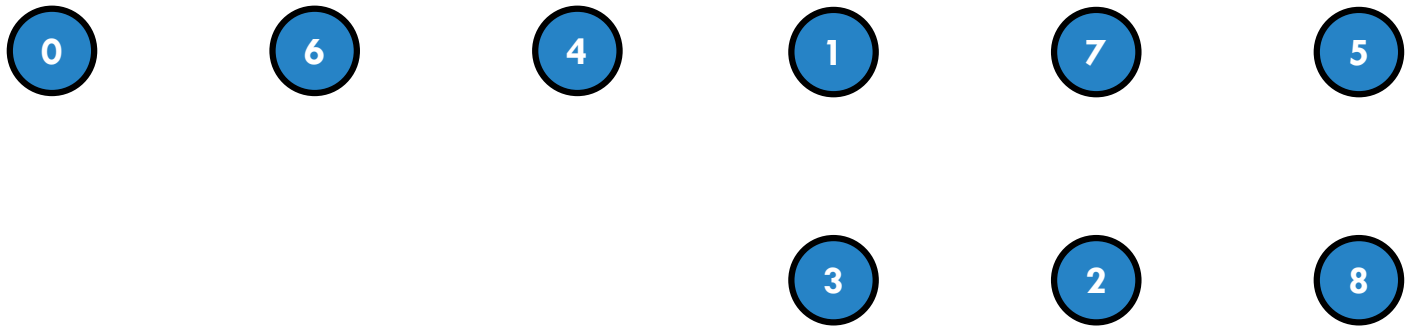
object	0	1	2	3	4	5	6	7	8
component identifier	0	1	1	3	0	1	0	1	3



QUICK FIND

Initial state of data structure:

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	2	3	4	5	6	7	8



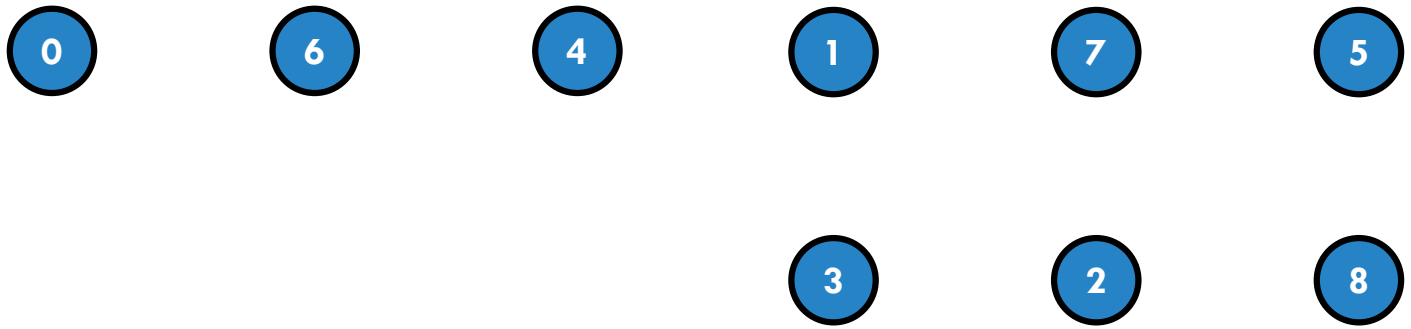
QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

Initial state of data structure:

union(1, 4)

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	2	3	4	5	6	7	8

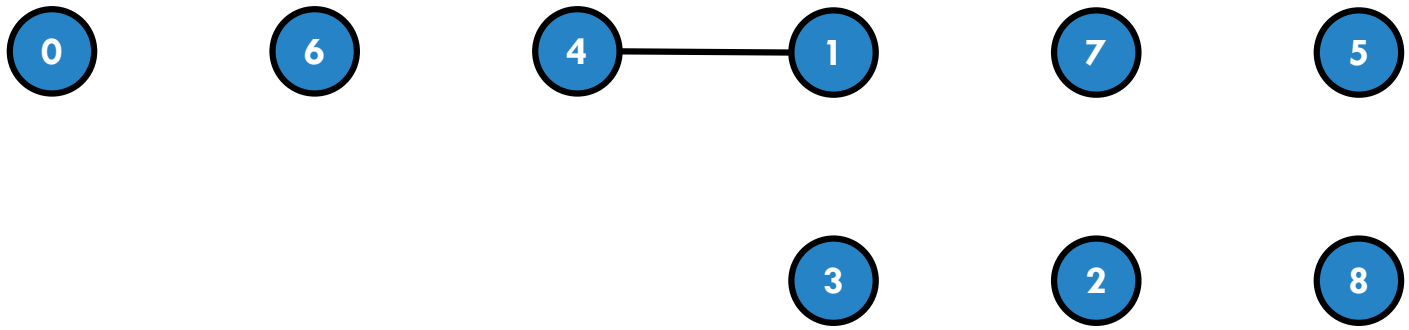


QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

union(1,4)

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	2	3	1	5	6	7	8

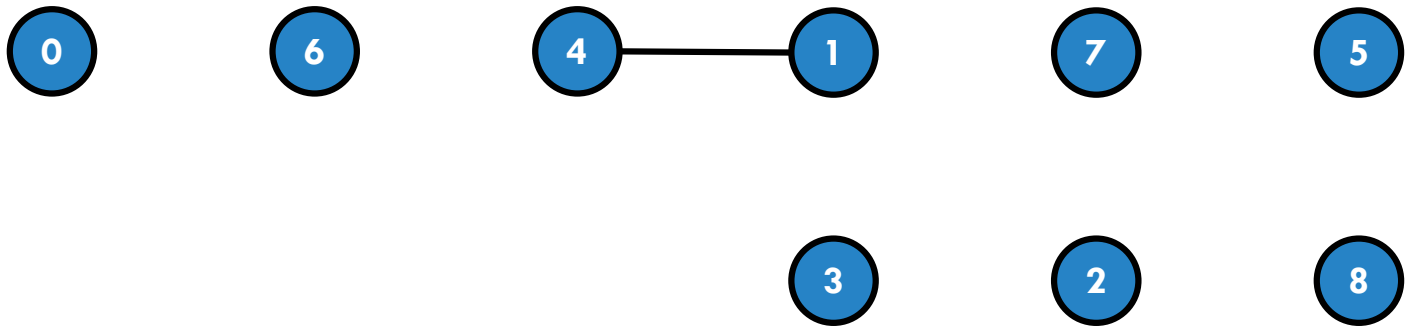


QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

union(1,3)

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	2	~~3~~ 1	1	5	6	7	8

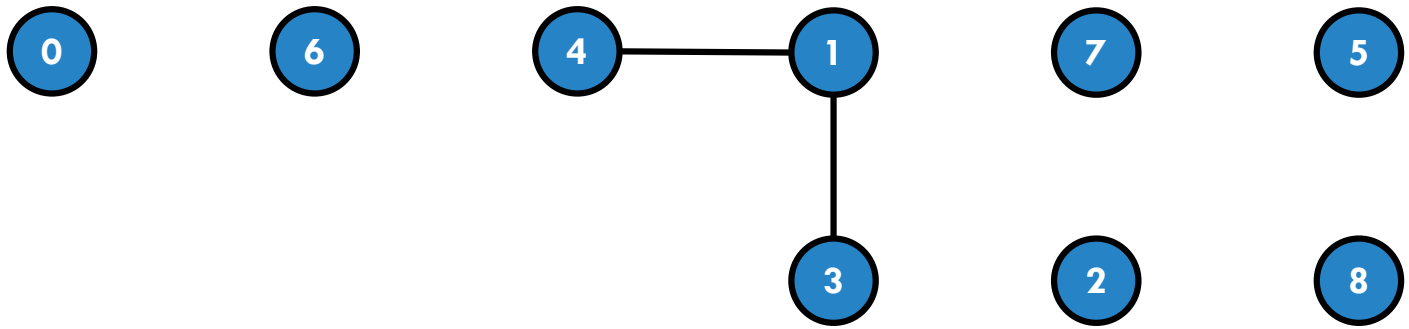


QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

union(1,3)

object	0	1	2	3	4	5	6	7	8
component identifier	0	1	2	1	1	5	6	7	8

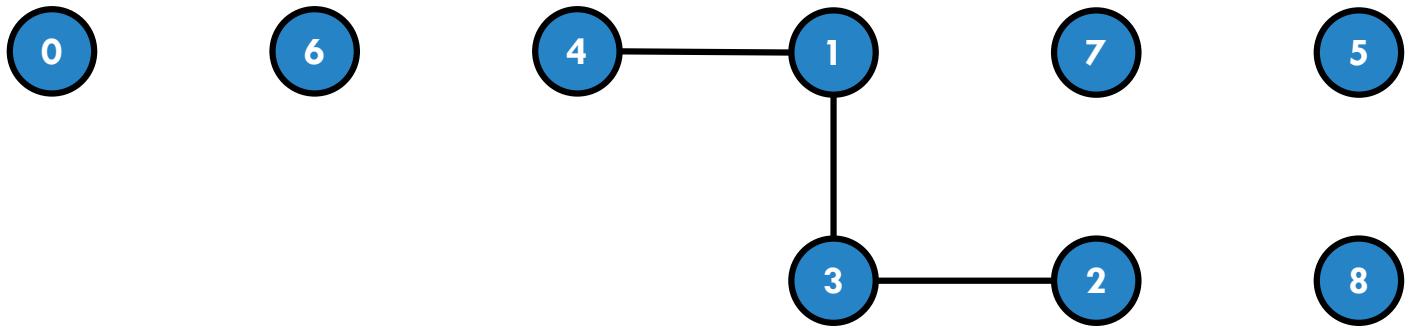


QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

union(2,1)

object	0	1	2	3	4	5	6	7	8
component identifier	0	1→?	2→?	1→?	1→?	5	6	7	8

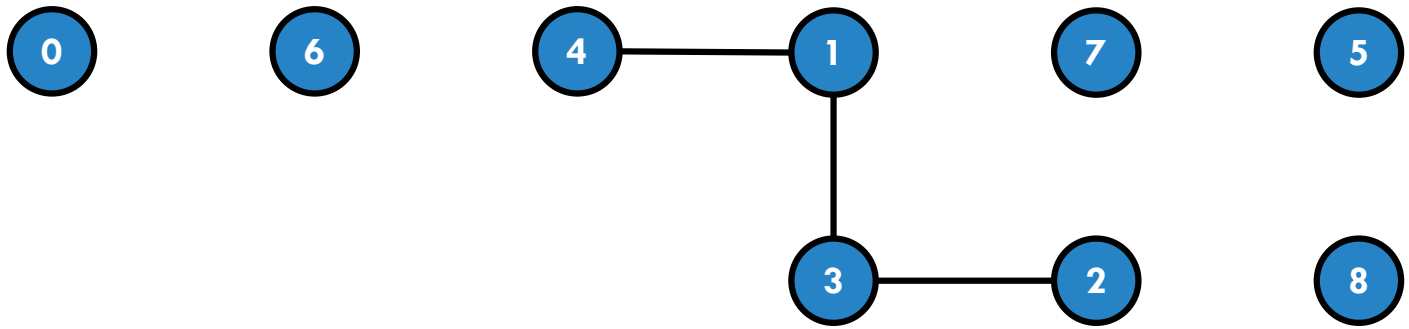


QUICK FIND

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
      if (componentId[i] ==  s)
          componentId[i] = componentId[p];
```

union (2,1)

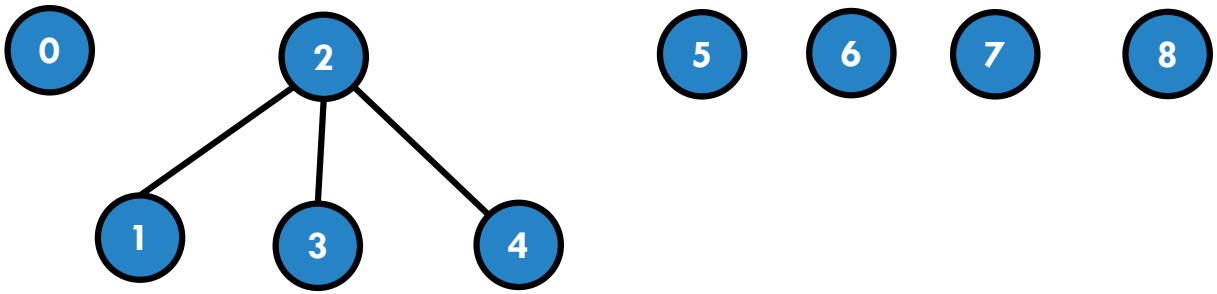
object	0	1	2	3	4	5	6	7	8
component identifier	0	2	2	2	2	5	6	7	8



QUICK FIND

object	0	1	2	3	4	5	6	7	8
component identifier	0	2	2	2	2	5	6	7	8

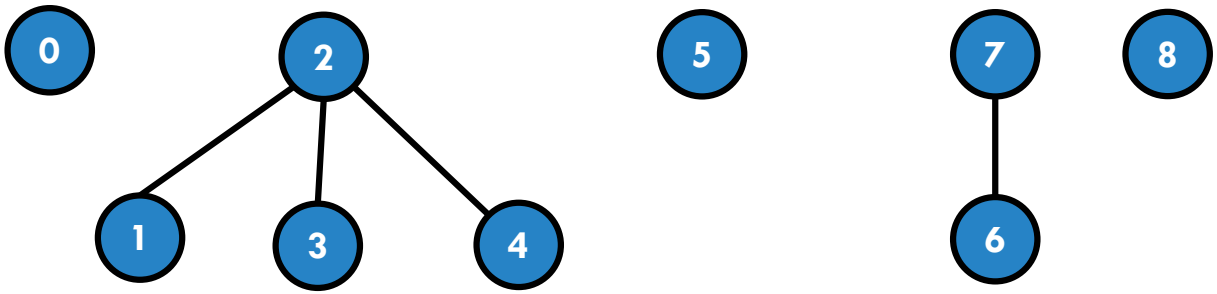
Flat trees:



QUICK FIND

object	0	1	2	3	4	5	6	7	8
component identifier	0	2	2	2	2	5	7	7	8

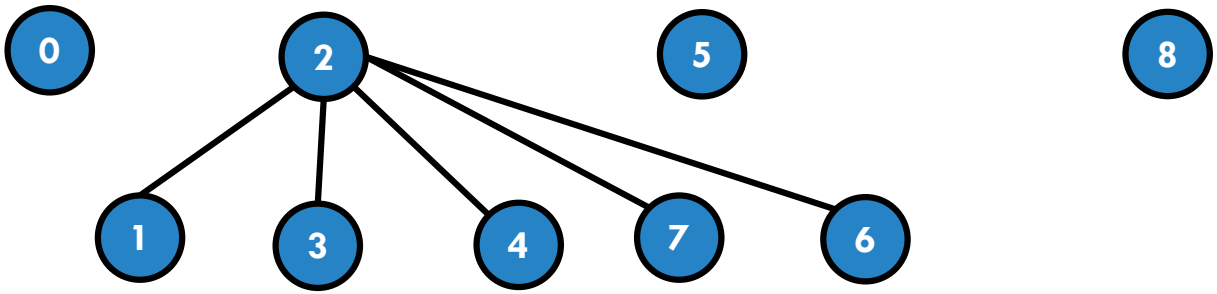
Flat trees:



QUICK FIND

object	0	1	2	3	4	5	6	7	8
component identifier	0	2	2	2	2	5	2	2	8

Flat trees:



HOW LONG DO THE OPERATIONS TAKE?

```
find(int p, int q)
 return(componentId[p] == componentId[q]);
```

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
       if (componentId[i] ==  s)
             componentId[i] = componentId[p];
```

How long does Find and Union take?

A. $O(1), O(n)$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $O(n), O(n)$
E. $O(\log n), O(\log n)$

HOW LONG DO THE OPERATIONS TAKE?

```
find(int p, int q)
 return(componentId[p] == componentId[q]);
```

```
union(int p, int q)
 int s = componentId[q]
 for (int i=0; i<componentId.length; i++)
       if (componentId[i] ==  s)
             componentId[i] = componentId[p];
```

Fast Find. Slow Union.

How long does Find and Union take?

A. $\boldsymbol{O(1), O(n)}$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $O(n), O(n)$
E. $O(\log n), O(\log n)$

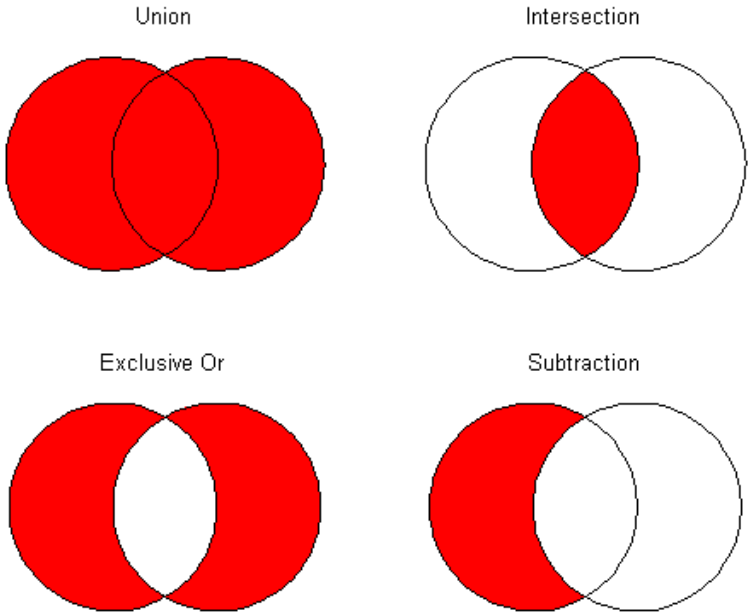
ROADMAP

Two "first-cut" methods:

- Quick find
- **Quick Union**

Union-Find

- Improving upon the first two approaches.



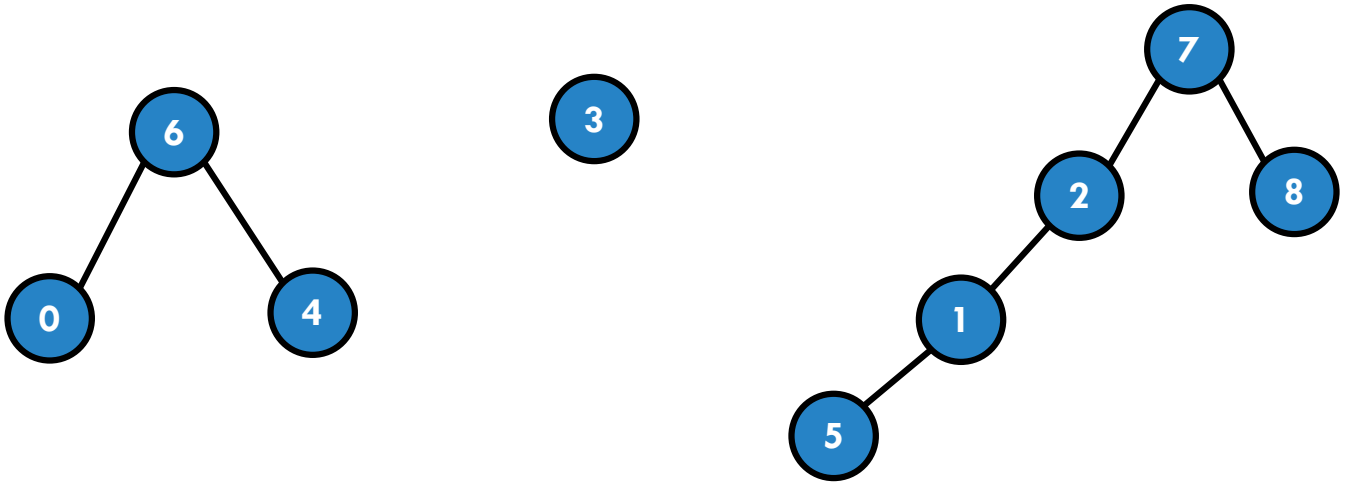
QUICK UNION

Data structure:

Integer array: `int[] parent`

Two objects are connected if they are part of the same tree

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7

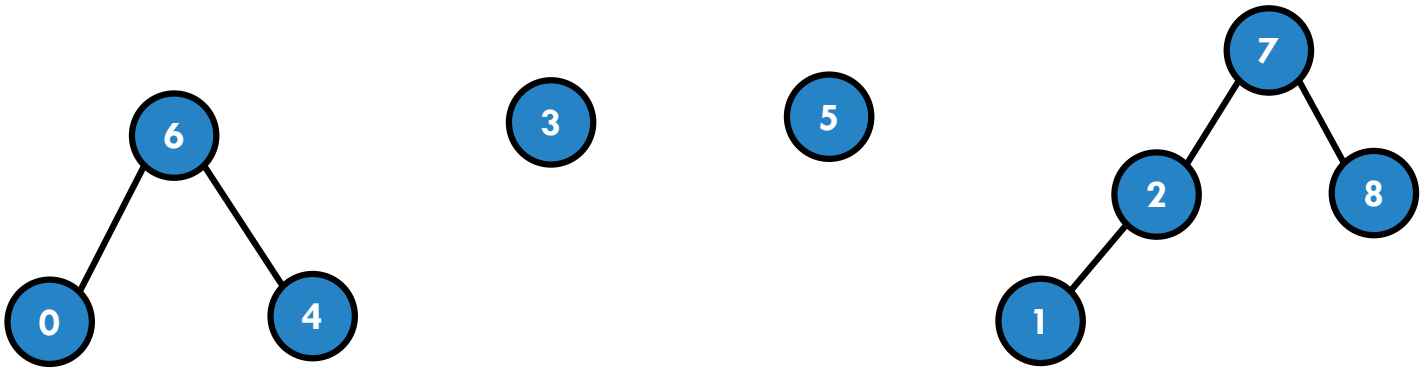


QUICK UNION: FIND

```
find(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   return (p == q);
```

Idea: Compare the root ids. Need to traverse upwards.

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



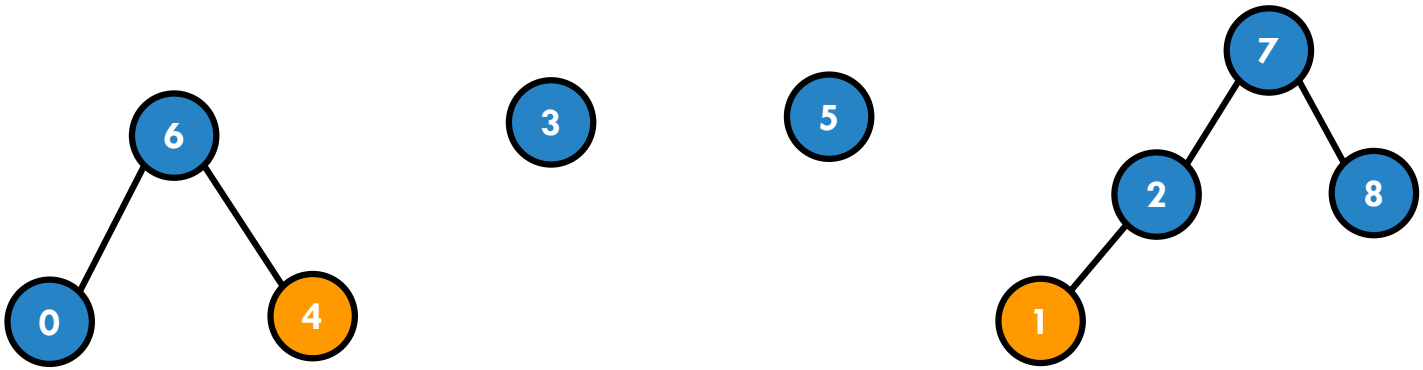
QUICK UNION: FIND

Example: `find(4, 1)`

4 → 6 → 6

```
find(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   return (p == q);
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



QUICK UNION: FIND

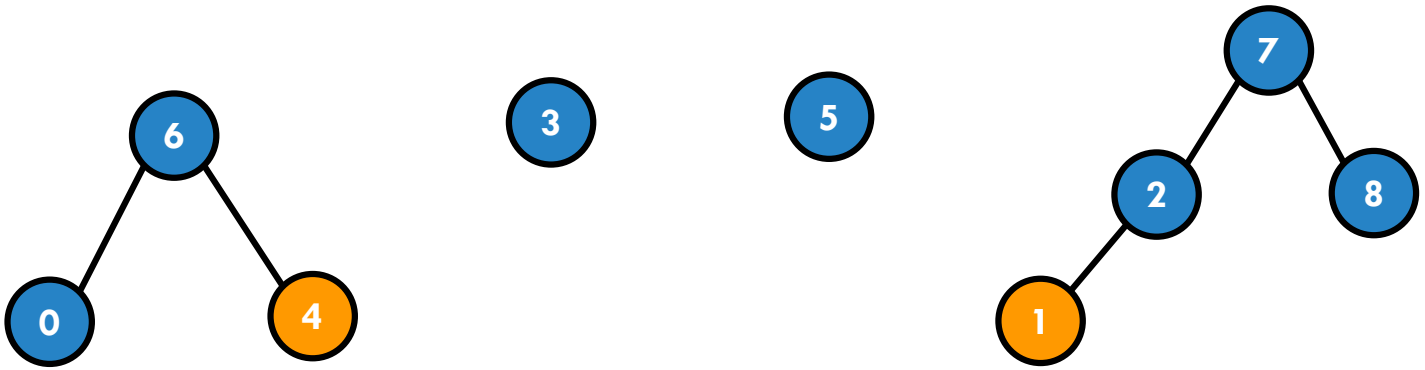
Example: `find(4, 1)`

4 → 6 → 6

1 → 2 → 7 → 7

```
find(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   return (p == q);
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



QUICK UNION: FIND

Example: `find(4, 1)`

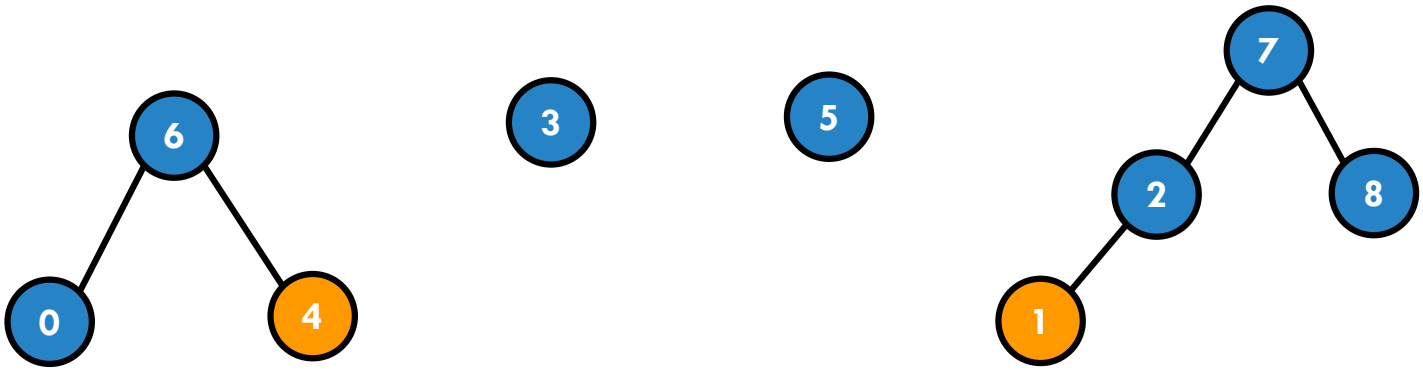
4 → 6 → 6

1 → 2 → 7 → 7

`return (6 == 7) // false`

```
find(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   return (p == q);
```

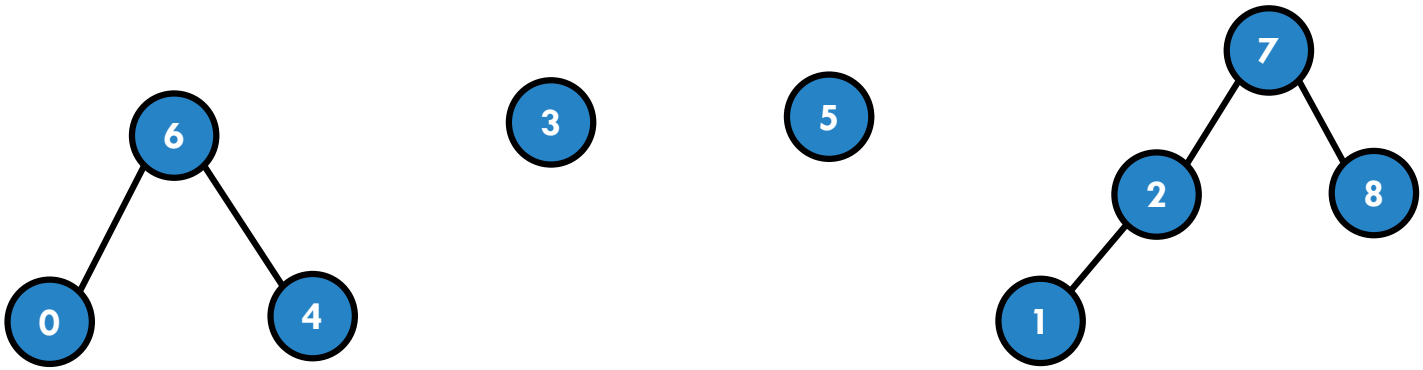
object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



QUICK UNION: UNION

```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



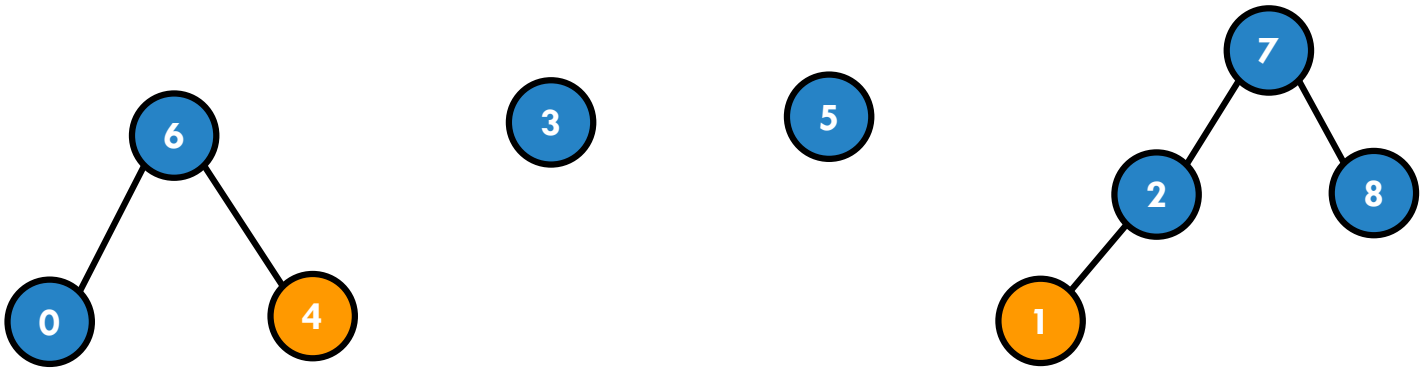
QUICK UNION: UNION

Example: union(1, 4)

1 → 2 → 7 → 7

```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



QUICK UNION: UNION

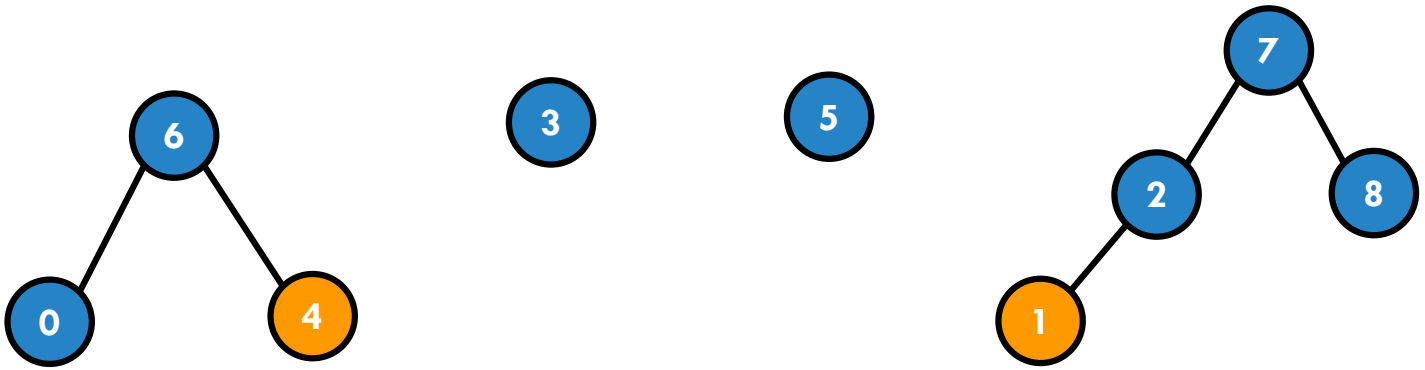
Example: union(1, 4)

1 → 2 → 7 → 7

4 → 6 → 6

```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	7	7



QUICK UNION: UNION

Example: union(1, 4)

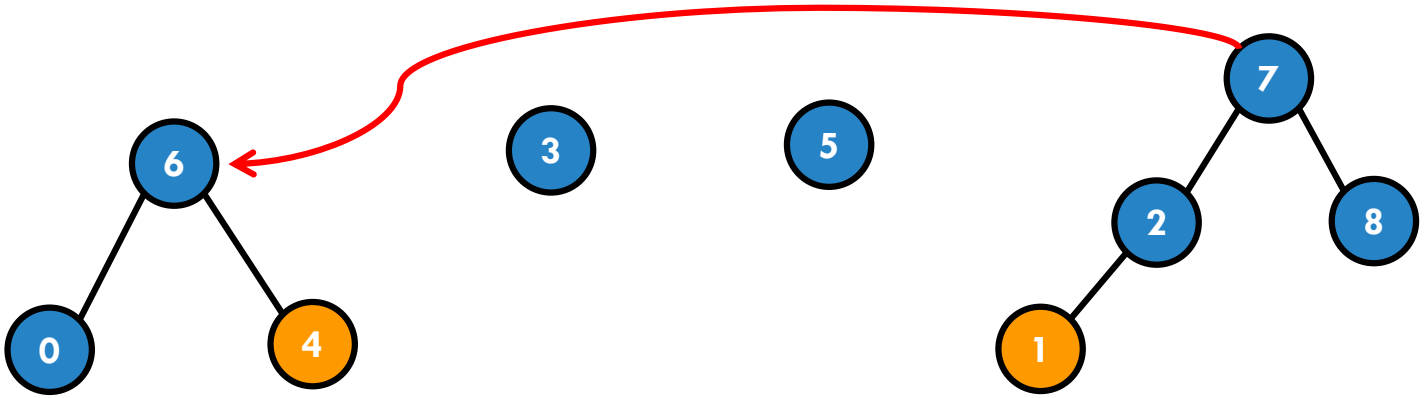
1 → 2 → 7 → 7

4 → 6 → 6

parent[7] = 6;

```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	6	7



QUICK UNION: UNION

```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

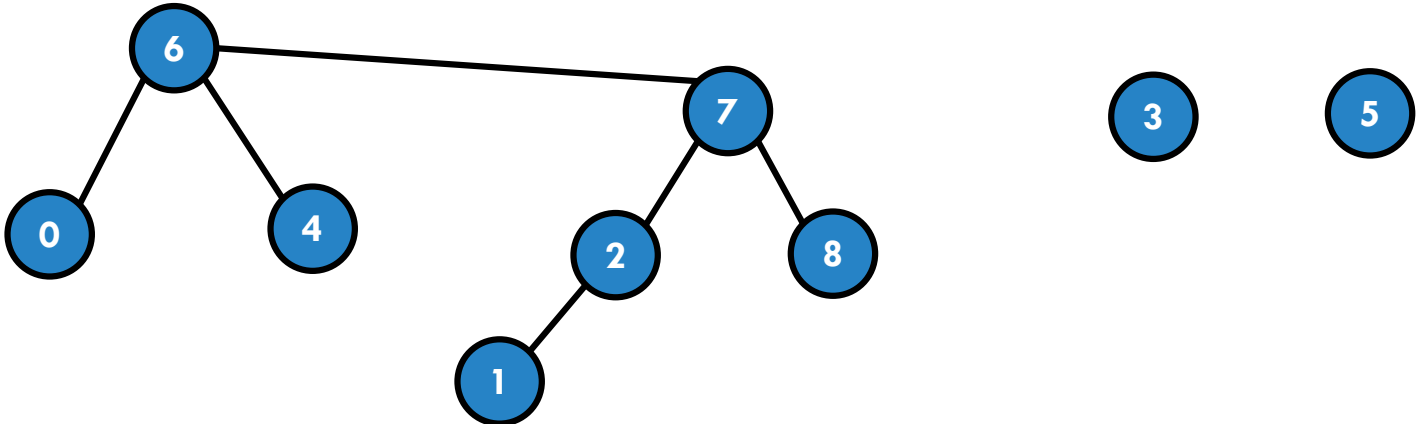
Example: union(1, 4)

1 → 2 → 7 → 7

4 → 6 → 6

parent[7] = 6;

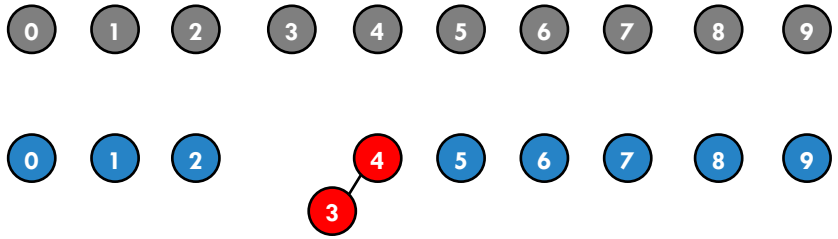
object	0	1	2	3	4	5	6	7	8
parent	6	2	7	3	6	1	6	6	7



P	0	1	2	3	4	5	6	7	8	9

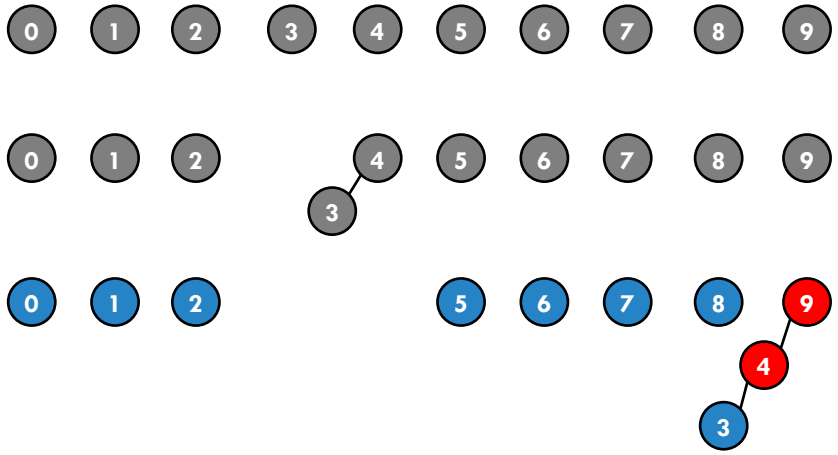

```
union(int p, int q)
     while (parent[p] != p) p = parent[p];
     while (parent[q] != q) q = parent[q];
     parent[p] = q;
```

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



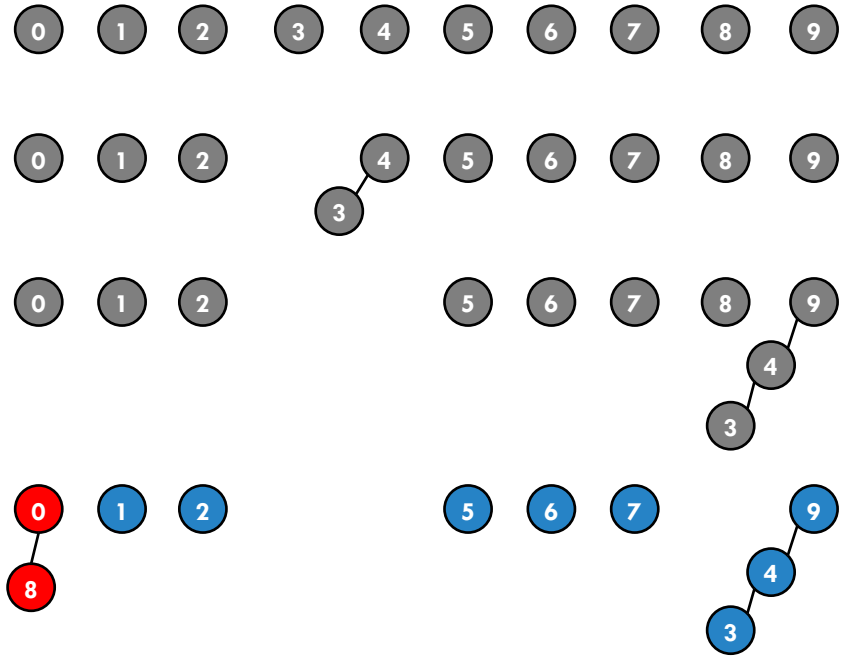

```
union(int p, int q)
     while (parent[p] != p) p = parent[p];
     while (parent[q] != q) q = parent[q];
     parent[p] = q;
```

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



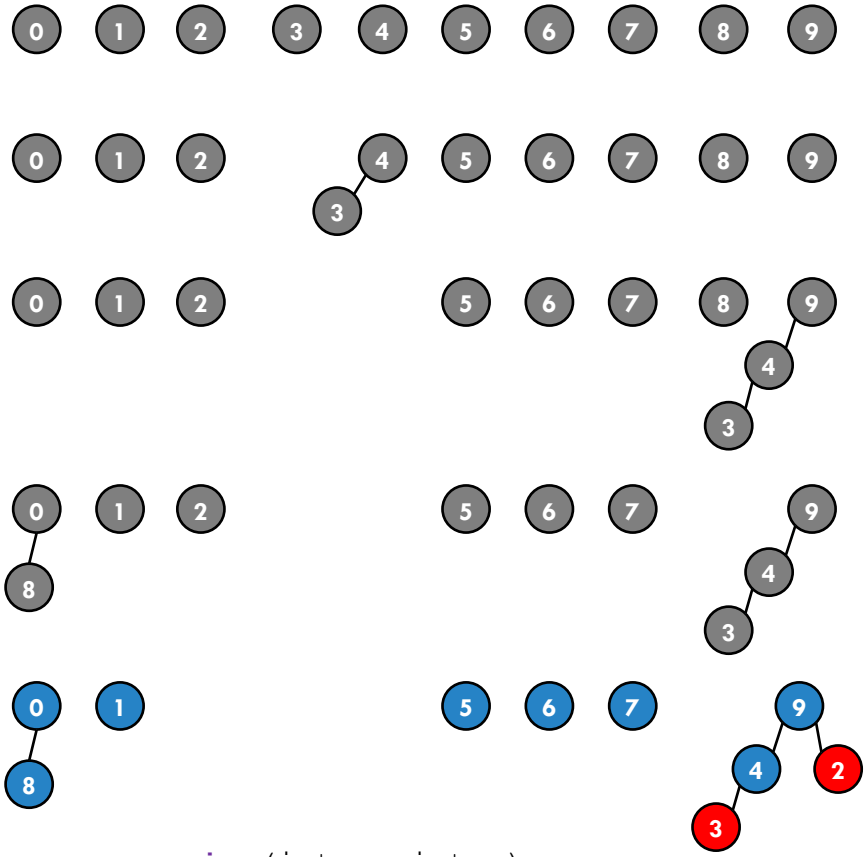

```
union(int p, int q)
     while (parent[p] != p) p = parent[p];
     while (parent[q] != q) q = parent[q];
     parent[p] = q;
```

P	0	1	2	3	4	5	6	7	8	9
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



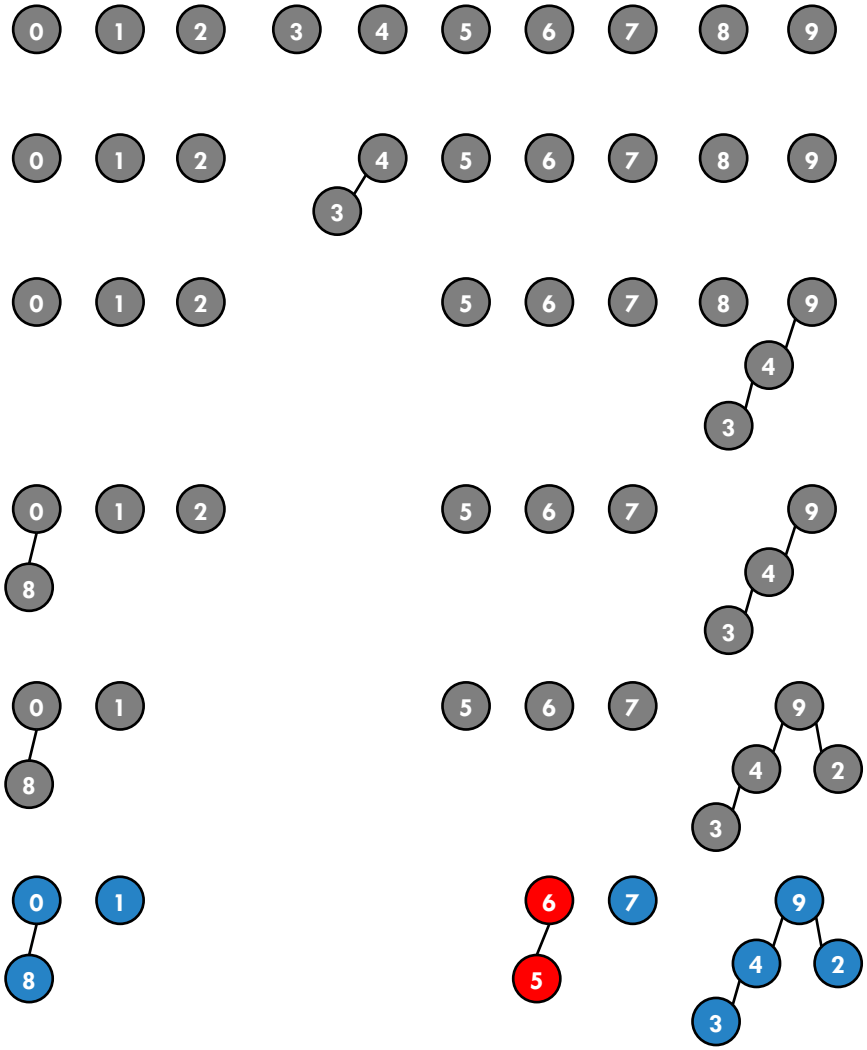

```
union(int p, int q)
    while (parent[p] != p) p = parent[p];
    while (parent[q] != q) q = parent[q];
    parent[p] = q;
```

P	0	1	2	3	4	5	6	7	8	9
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

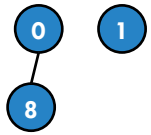


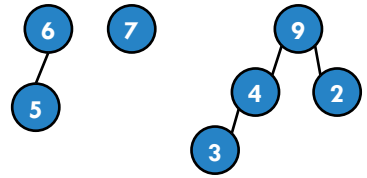
```
union(int p, int q)
    while (parent[p] != p) p = parent[p];
    while (parent[q] != q) q = parent[q];
    parent[p] = q;
```

P	0	1	2	3	4	5	6	7	8	9
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



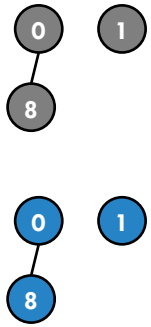
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	9	5	6	7	8	9
8-0	0	1	2	4	9	5	6	7	0	9
2-3	0	1	9	4	9	5	6	7	0	9
5-6	0	1	9	4	9	6	6	7	0	9

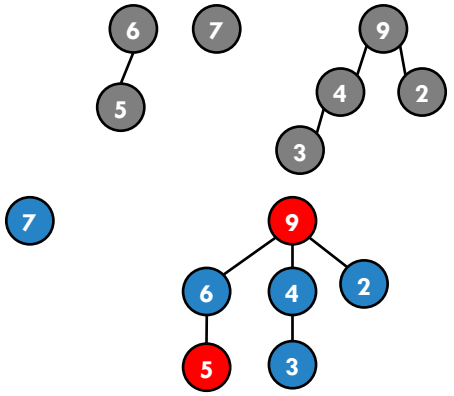





```
union(int p, int q)
     while (parent[p] != p) p = parent[p];
     while (parent[q] != q) q = parent[q];
     parent[p] = q;
```

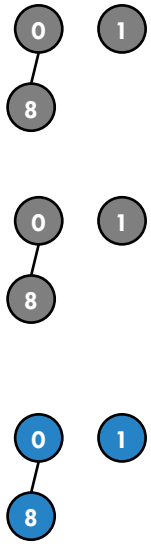
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	9	5	6	7	8	9
8-0	0	1	2	4	9	5	6	7	0	9
2-3	0	1	9	4	9	5	6	7	0	9
5-6	0	1	9	4	9	6	6	7	0	9
5-9	0	1	9	4	9	6	9	7	0	9

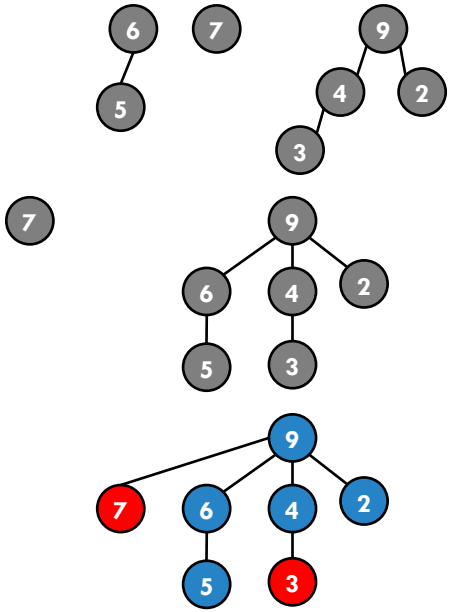





```
union(int p, int q)
     while (parent[p] != p) p = parent[p];
     while (parent[q] != q) q = parent[q];
     parent[p] = q;
```

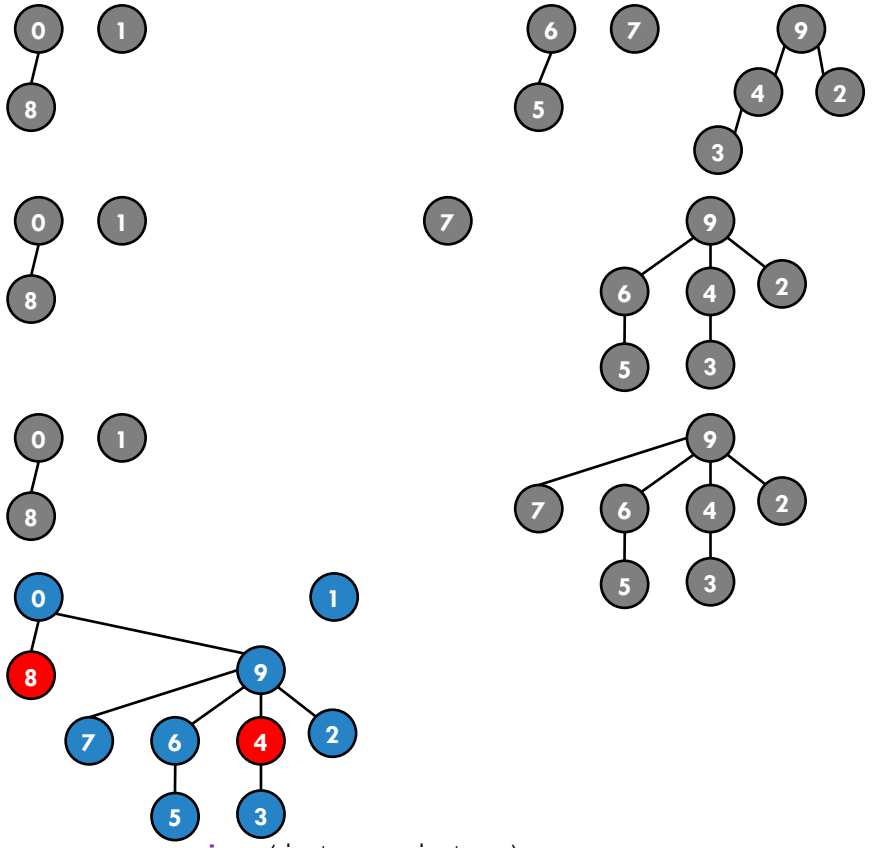
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	9	5	6	7	8	9
8-0	0	1	2	4	9	5	6	7	0	9
2-3	0	1	9	4	9	5	6	7	0	9
5-6	0	1	9	4	9	6	6	7	0	9
5-9	0	1	9	4	9	6	9	7	0	9
7-3	0	1	9	4	9	6	9	9	0	9





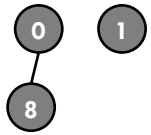

```
union(int p, int q)
    while (parent[p] != p) p = parent[p];
    while (parent[q] != q) q = parent[q];
    parent[p] = q;
```

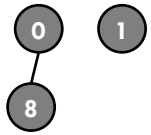
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	9	5	6	7	8	9
8-0	0	1	2	4	9	5	6	7	0	9
2-3	0	1	9	4	9	5	6	7	0	9
5-6	0	1	9	4	9	6	6	7	0	9
5-9	0	1	9	4	9	6	9	7	0	9
7-3	0	1	9	4	9	6	9	9	0	9
4-8	0	1	9	4	9	6	9	9	0	0

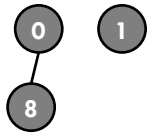


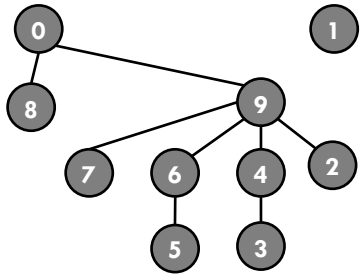
```
union(int p, int q)
    while (parent[p] != p) p = parent[p];
    while (parent[q] != q) q = parent[q];
    parent[p] = q;
```

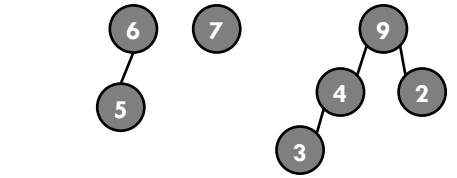
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	9	5	6	7	8	9
8-0	0	1	2	4	9	5	6	7	0	9
2-3	0	1	9	4	9	5	6	7	0	9
5-6	0	1	9	4	9	6	6	7	0	9
5-9	0	1	9	4	9	6	9	7	0	9
7-3	0	1	9	4	9	6	9	9	0	9
4-8	0	1	9	4	9	6	9	9	0	0
6-1	1	1	9	4	9	6	9	9	0	0

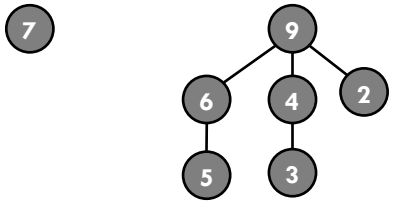


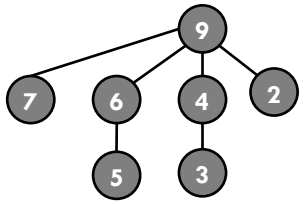


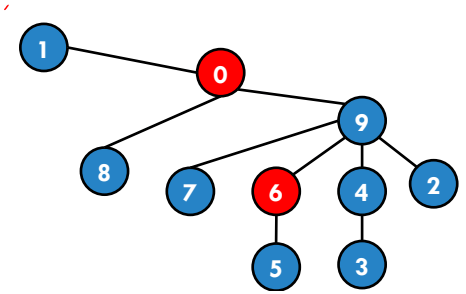












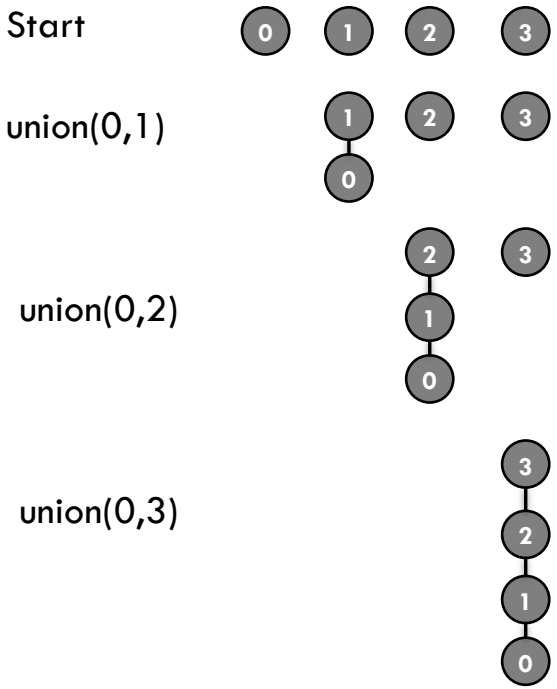
QUICK UNION: HOW LONG DO THE OPERATIONS TAKE?

```
find(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   return (p == q);


union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q = parent[q];
   parent[p] = q;
```

How long does Find and Union take?

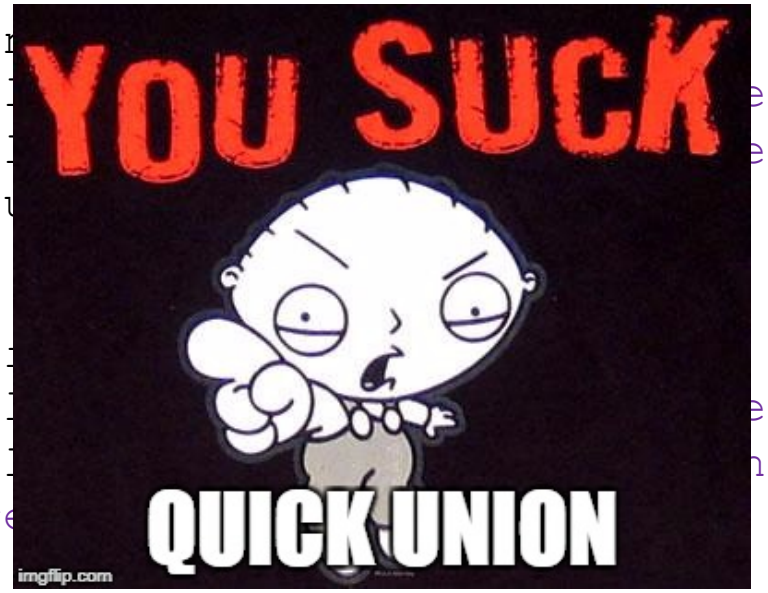
A. $O(1), O(n)$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $O(n), O(n)$
E. $O(\log n), O(\log n)$

QUICK UNION WORST CASE




```
union(int p, int q)
   while (parent[p] != p) p = parent[p];
   while (parent[q] != q) q= parent[q];
   parent[p] = q;
```

QUICK UNION: HOW LONG DO THE OPERATIONS TAKE?



Slow Find. Slow Union!

How long does Find and Union take?

A. $O(1), O(n)$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $\boldsymbol{O(n), O(n)}$
E. $O(\log n), O(\log n)$

UNION-FIND: STORY SO FAR

Quick-find:

- Union is slow
- Tree is flat

Quick-union:

- Union *and* find are slow.
- Trees too tall (i.e., unbalanced)

	find	union
quick-find	O(1)	O(n)
quick-union	O(n)	O(n)

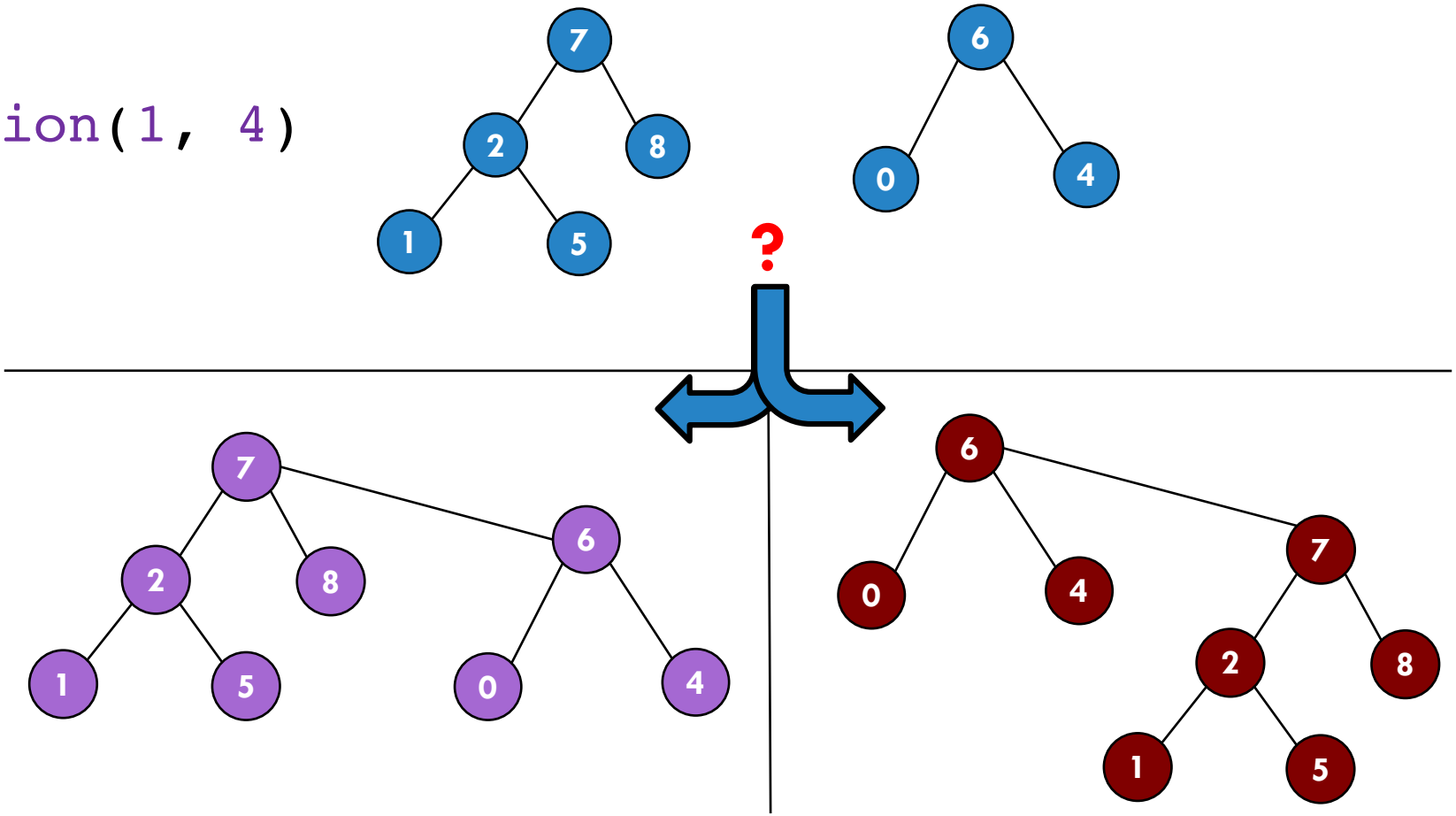
Can we make Quick Union better?

Maybe… make the trees less tall?

WEIGHTED UNION

Question: which tree should you make the root?

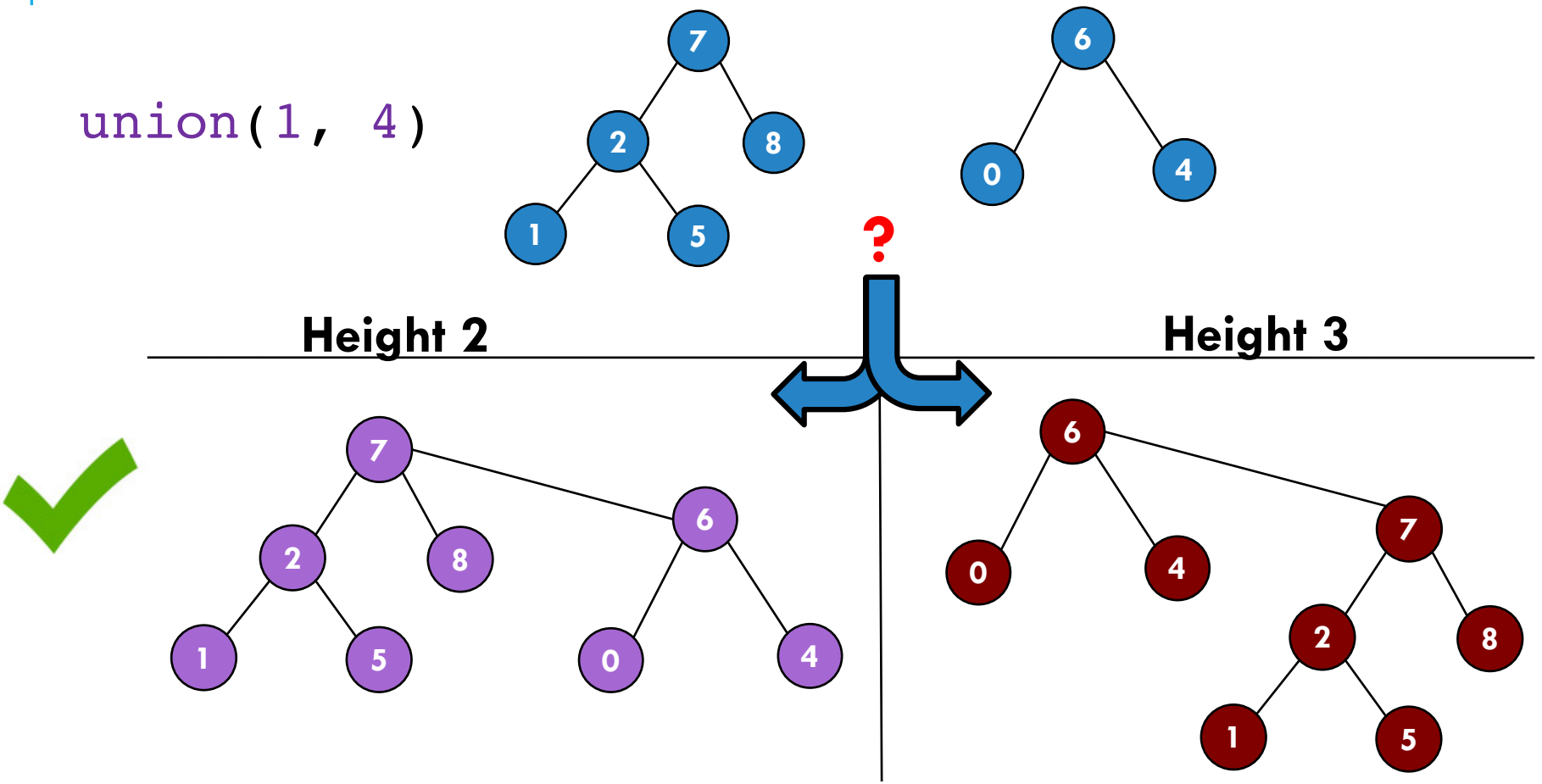
```
union(1, 4)
```



WEIGHTED UNION

Question: which tree should you make the root?

```
union(1, 4)
```



WEIGHTED UNION

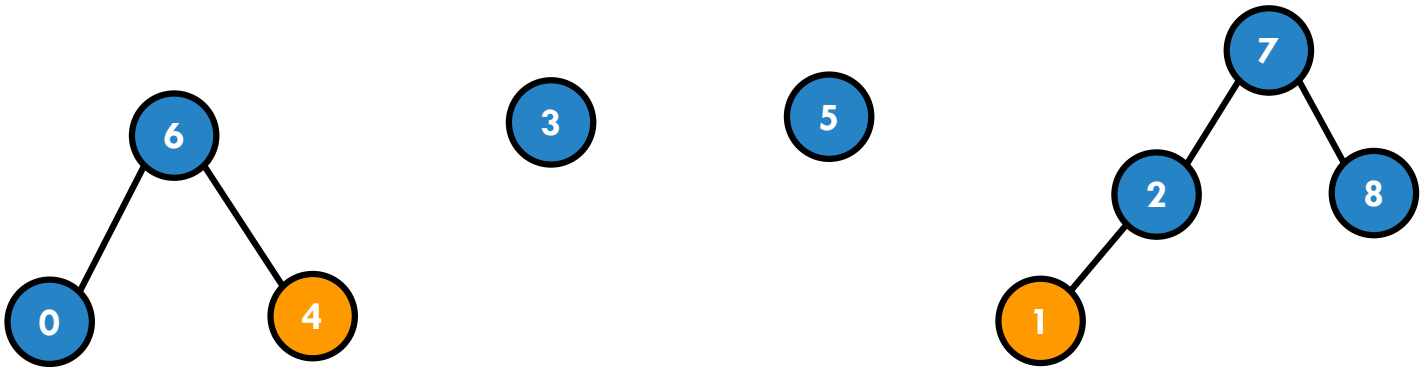
```
union(int p, int q)
 while (parent[p] !=p) p = parent[p];
 while (parent[q] !=q) q = parent[q];
 if (size[p] > size[q]) {
      parent[q] = p;   // Link q to p
      size[p] = size[p] + size[q];
 }
 else {
      parent[p] = q; // Link p to q
      size[q] = size[p] + size[q];
 }
```

use size instead of heights

WEIGHTED UNION

object	0	1	2	3	4	5	6	7	8
size	1	1	2	1	1	1	3	4	1
parent	6	2	7	3	6	1	6	7	7

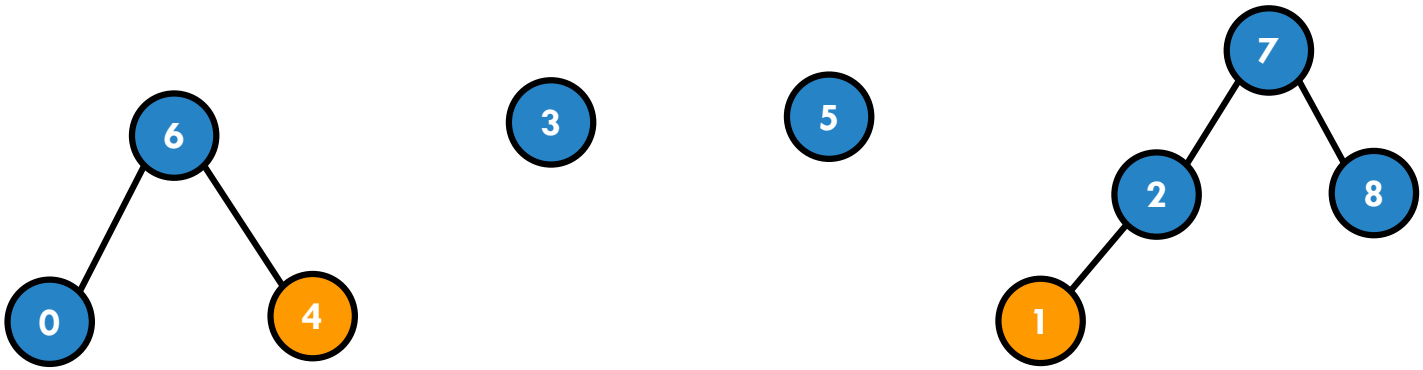
`union(1, 4)`



WEIGHTED UNION

object	0	1	2	3	4	5	6	7	8
size	1	1	2	1	1	1	3	4	1
parent	6	2	7	3	6	1	6	7	7

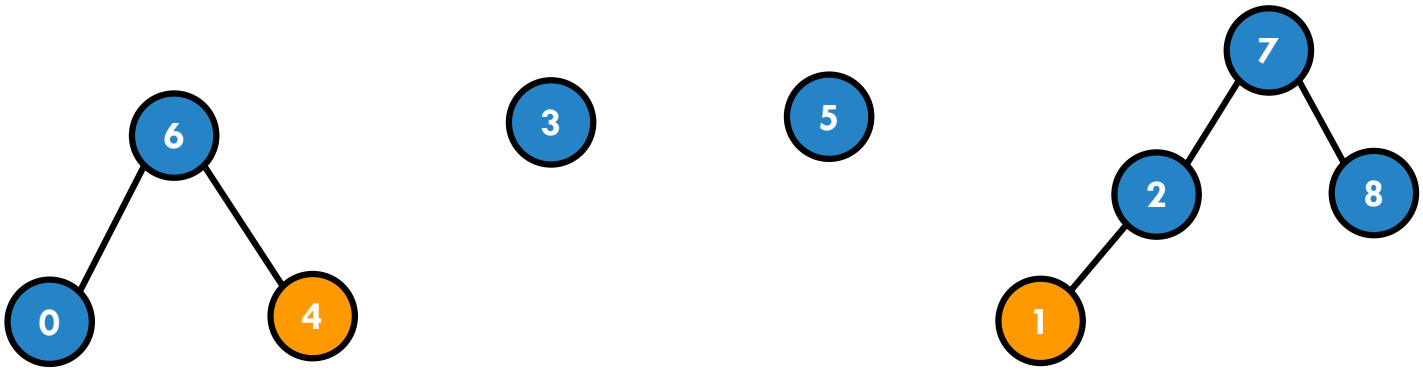
`union(1, 4)`



WEIGHTED UNION

object	0	1	2	3	4	5	6	7	8
size	1	1	2	1	1	1	3	4	1
parent	6	2	7	3	6	1	6	7	7

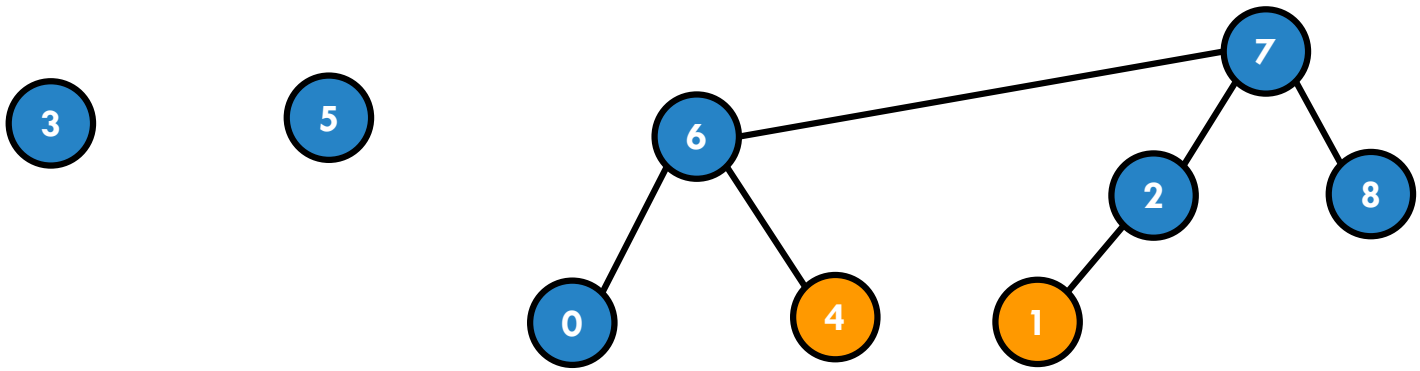
`union(1, 4)`



WEIGHTED UNION

object	0	1	2	3	4	5	6	7	8
size	1	1	2	1	1	1	3	7	1
parent	6	2	7	3	6	1	6	7	7

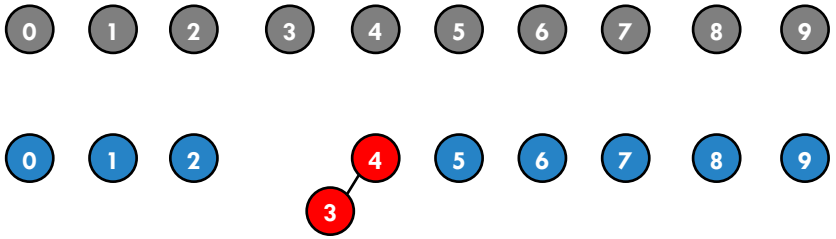
`union(1, 4)`



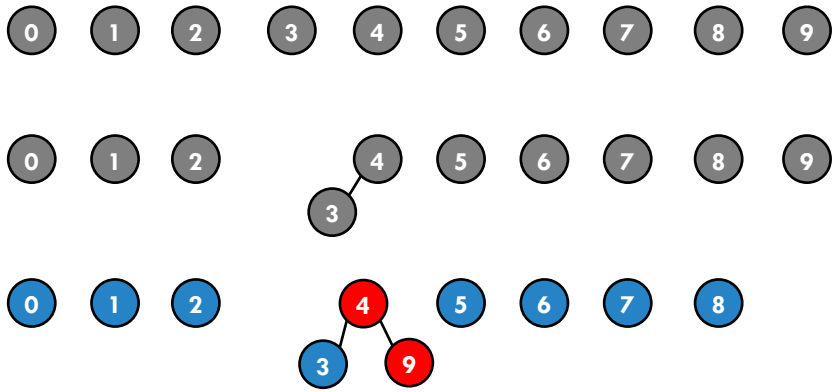

P 0 1 2 3 4 5 6 7 8 9


0 1 2 3 4 5 6 7 8 9

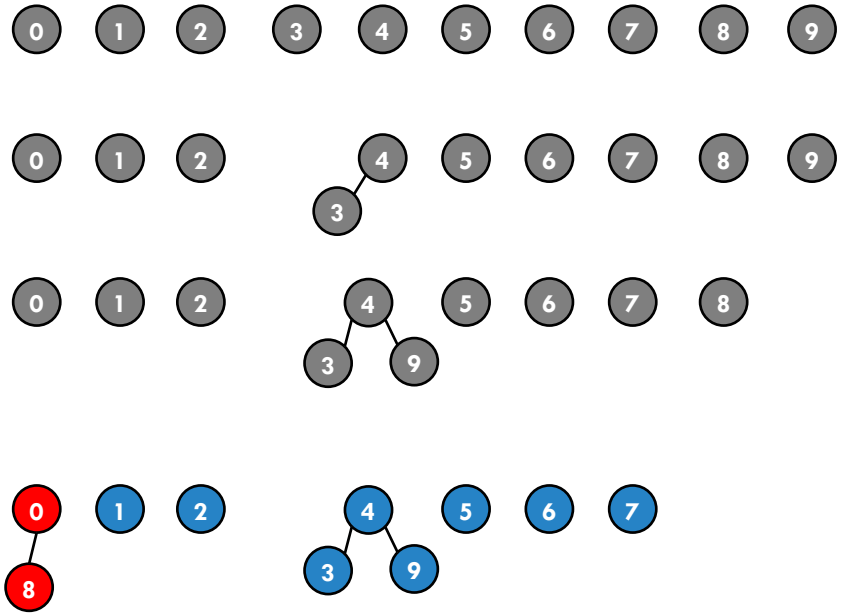
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9



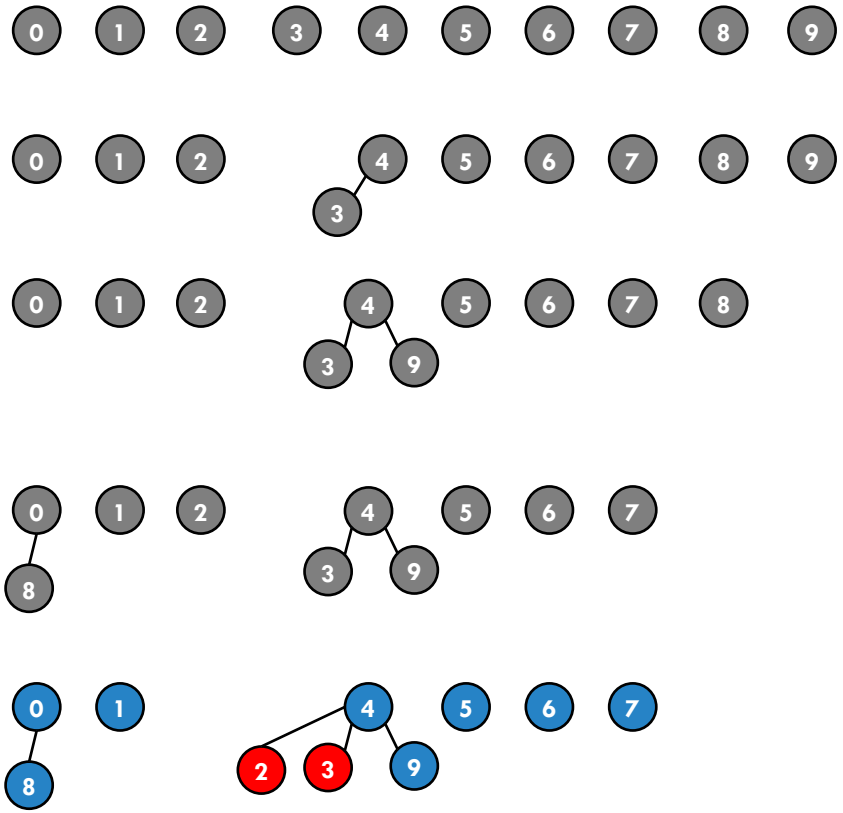
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4



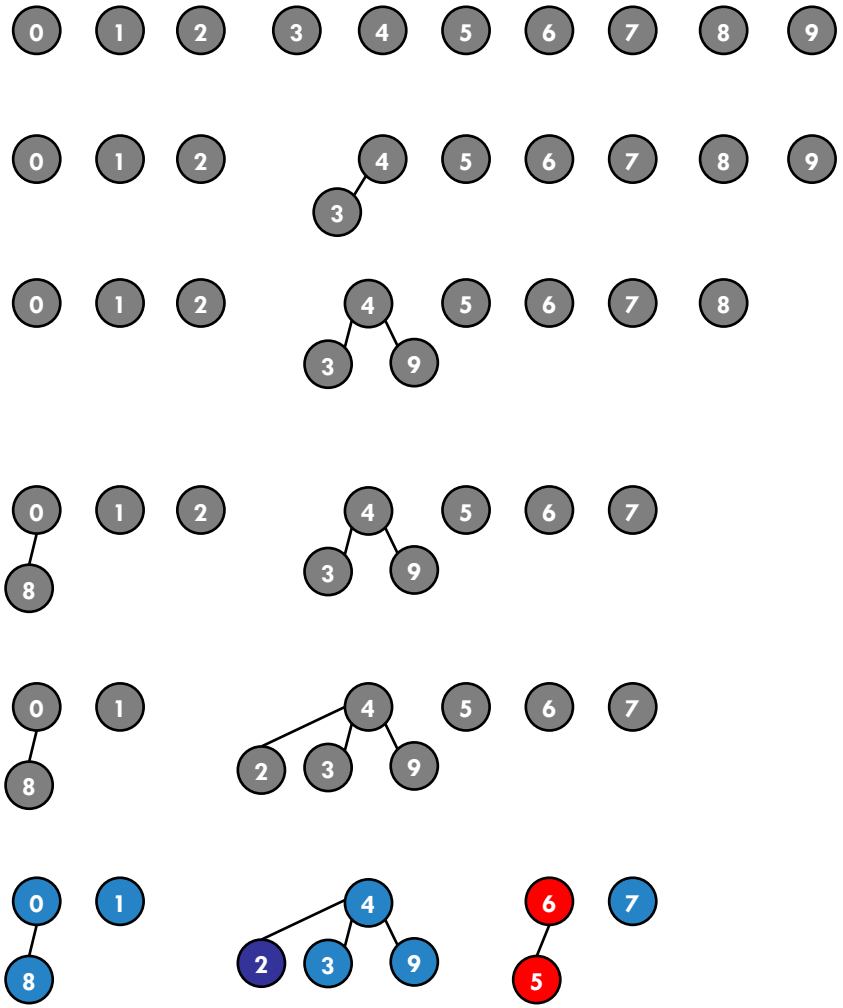
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4



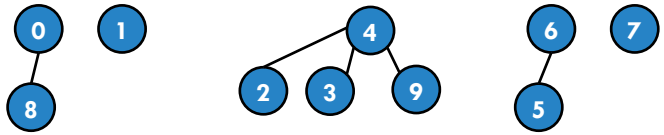
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4



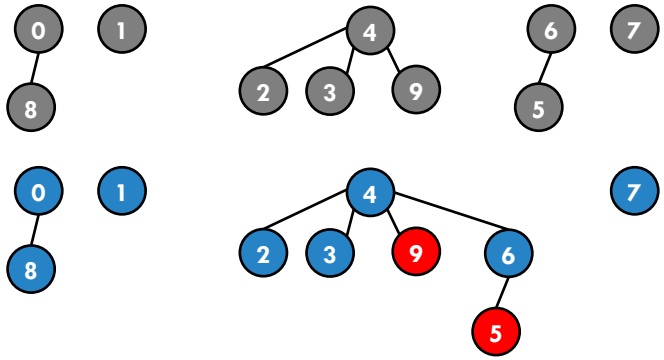
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4



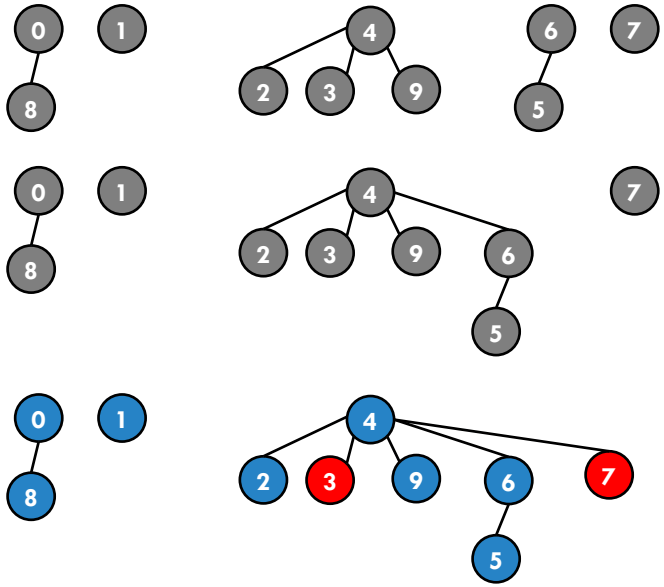
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4



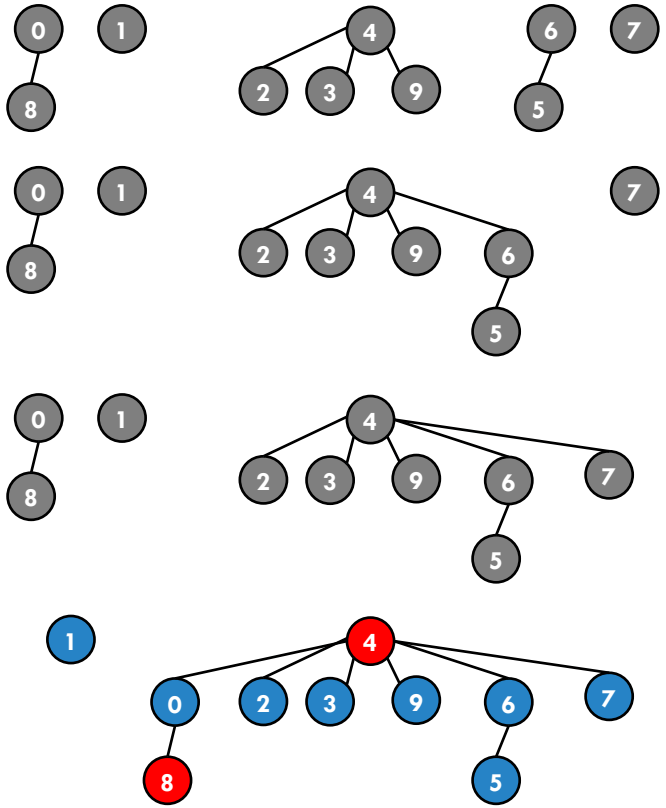
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4
5-9	0	1	4	4	4	6	4	7	0	4



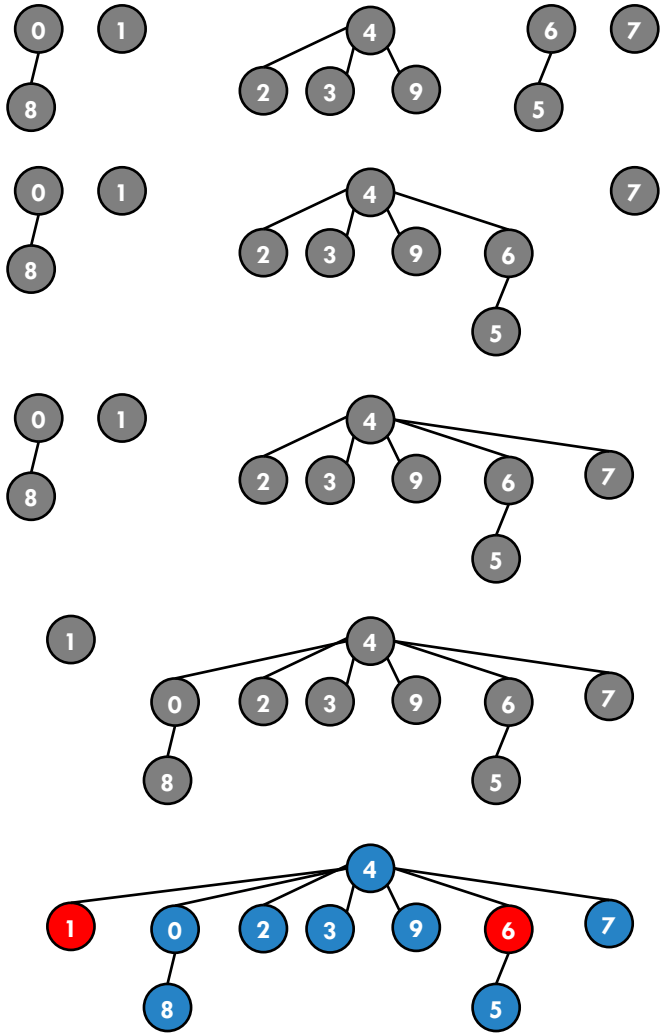
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4
5-9	0	1	4	4	4	6	4	7	0	4
7-3	0	1	4	4	4	6	4	4	0	4



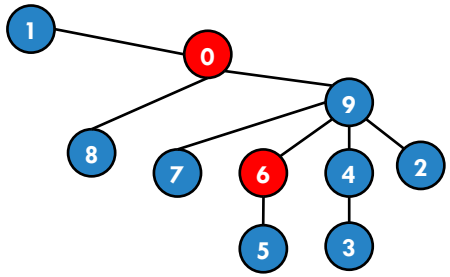
P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4
5-9	0	1	4	4	4	6	4	7	0	4
7-3	0	1	4	4	4	6	4	4	0	4
4-8	4	1	4	4	4	6	4	4	0	4



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

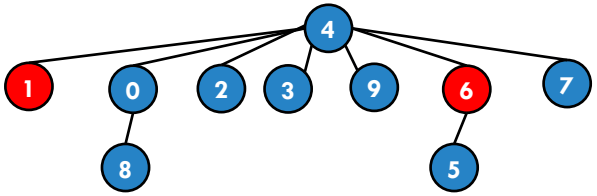


P	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	4	4	5	6	7	8	9
4-9	0	1	2	4	4	5	6	7	8	4
8-0	0	1	2	4	4	5	6	7	0	4
2-3	0	1	4	4	4	5	6	7	0	4
5-6	0	1	4	4	4	6	6	7	0	4
5-9	0	1	4	4	4	6	4	7	0	4
7-3	0	1	4	4	4	6	4	4	0	4
4-8	4	1	4	4	4	6	4	4	0	4
6-1	4	4	4	4	4	6	4	4	0	4



Quick Union

V.S.



Weighted Union

WEIGHTED UNION

```
union(int p, int q)
 while (parent[p] !=p) p = parent[p];
 while (parent[q] !=q) q = parent[q];
 if (size[p] > size[q] {
       parent[q] = p;   // Link q to p
       size[p] = size[p] + size[q];
 }
 else {
       parent[p] = q; // Link p to q
       size[q] = size[p] + size[q];
 }
```

What is the max height of a tree built using weighted union?

A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(\sqrt{n})$
E.

WEIGHTED UNION

```
union(int p, int q)
 while (parent[p] !=p) p = parent[p];
 while (parent[q] !=q) q = parent[q];
 if (size[p] > size[q] {
       parent[q] = p;   // Link q to p
       size[p] = size[p] + size[q];
 }
 else {
       parent[p] = q; // Link p to q
       size[q] = size[p] + size[q];
 }
```

What is the max height of a tree built using weighted union?

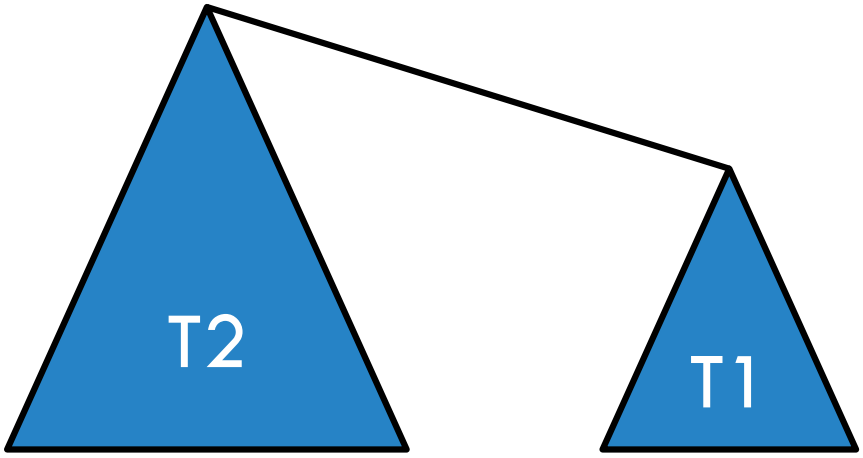
A. $O(1)$
B. $O(n)$
C. $O(\log n)$
D. $O(\sqrt{n})$
E.

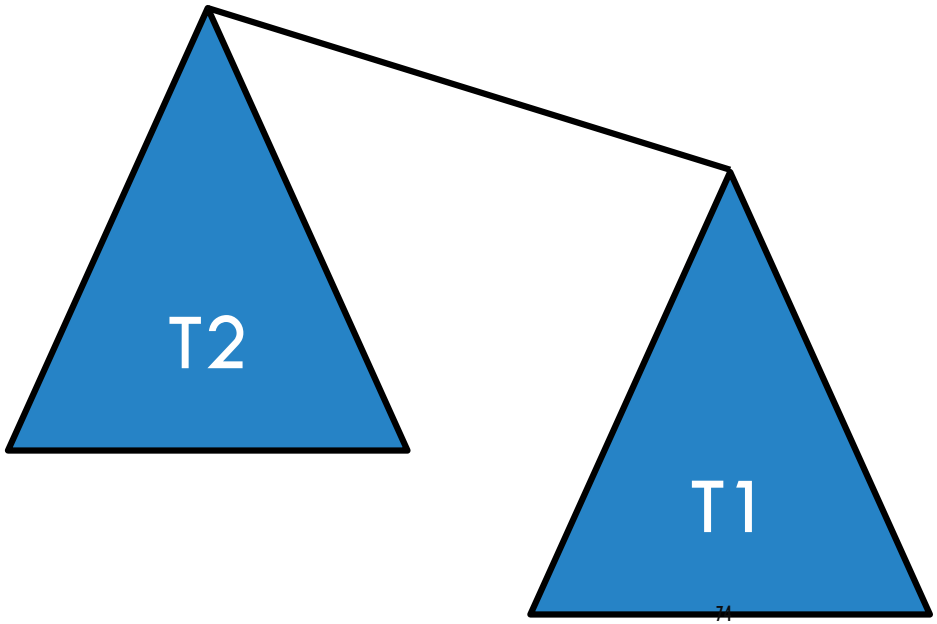
Why?

ANALYSIS

- Tree T1 is merged with Tree T2.
- When is the height of merged tree larger than T1 and T2?

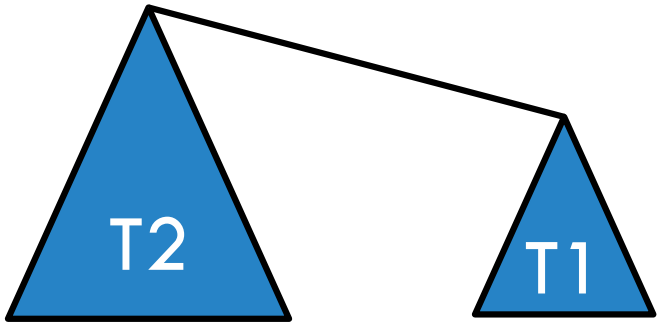
Only if: height(T2) = height(T1)





ANALYSIS: INDUCTION

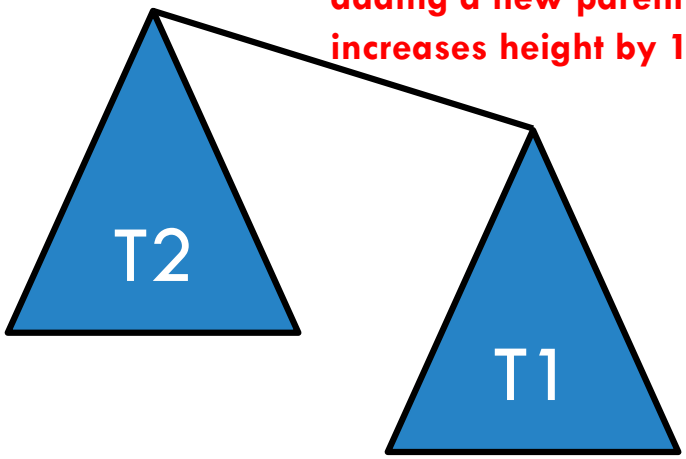
Claim: The height of a tree built via weighted union is at most $\log\ n$ where n is the size of the tree.



Proof Sketch:

- **Base case:**
  - tree of height 0 contains 1 object.
- **Inductive (strong) hypothesis:**
  - assume tree of size i has height at most $\log i$ for all $i \leq\ k$
- **Inductive Step:**
  - Combine tree of size i with j where $i \leq j$
  - Tree size now $s\ =\ i + j$
  - **Note that:** $1\ + \log i = \log(2i) = \log(i + i) \leq\ \log(i + j)\ =\ \log s$
- **Conclusion:** $\log_2 2$
  - Each tree of size n is of height at most $\log n$ ■

adding a new parent increases height by 1



WEIGHTED UNION: HOW LONG DO THE OPERATIONS TAKE?

```
union(int p, int q)
 while (parent[p] !=p) p = parent[p];
 while (parent[q] !=q) q = parent[q];
 if (size[p] > size[q] {
       parent[q] = p;   // Link q to p
       size[p] = size[p] + size[q];
 }
 else {
       parent[p] = q; // Link p to q
       size[q] = size[p] + size[q];
 }
```

How long does Find and Union take?

A. $O(1), O(n)$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $O(n), O(n)$
E. $O(\log n), O(\log n)$

WEIGHTED UNION: HOW LONG DO THE OPERATIONS TAKE?

```
union(int p, int q)
 while (parent[p] !=p) p = parent[p];
 while (parent[q] !=q) q = parent[q];
 if (size[p] > size[q] {
       parent[q] = p;   // Link q to p
       size[p] = size[p] + size[q];
 }
 else {
       parent[p] = q; // Link p to q
       size[q] = size[p] + size[q];
 }
```

How long does Find and Union take?

A. $O(1), O(n)$
B. $O(n), O(1)$
C. $O(1), O(1)$
D. $O(n), O(n)$
E. $O(\log n), O(\log n)$

UNION-FIND: STORY SO FAR

Quick-find and Quick-union are slow:

- Union and/or find is expensive
- Quick-union: tree is too deep

Weighted-union is faster:

- Trees flatter: O(log n)
- Union *and* find are O(log n)

	find	union
quick-find	O(1)	O(n)
quick-union	O(n)	O(n)
weighted-union	O(log n)	O(log n)

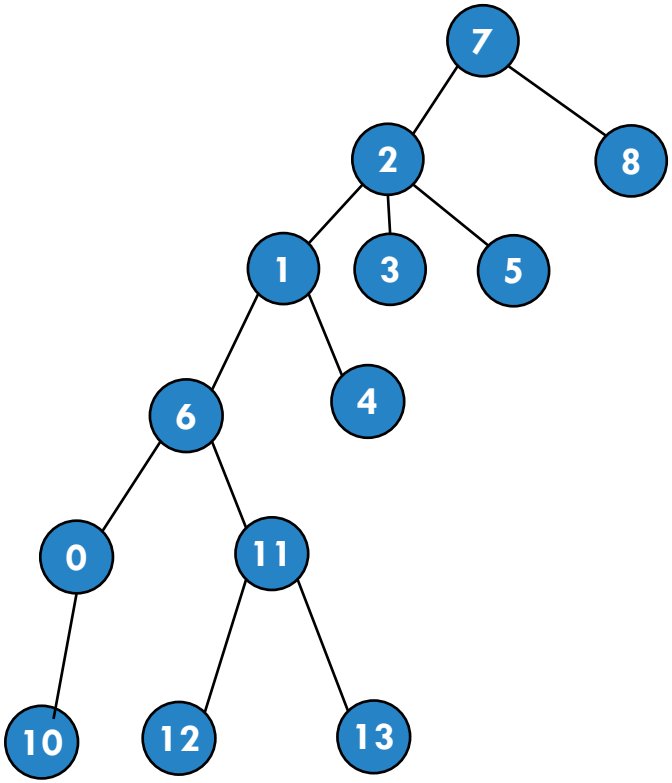
Can we do even better?

Yes! Path Compression!

PATH COMPRESSION

Idea: Flatten the tree after find

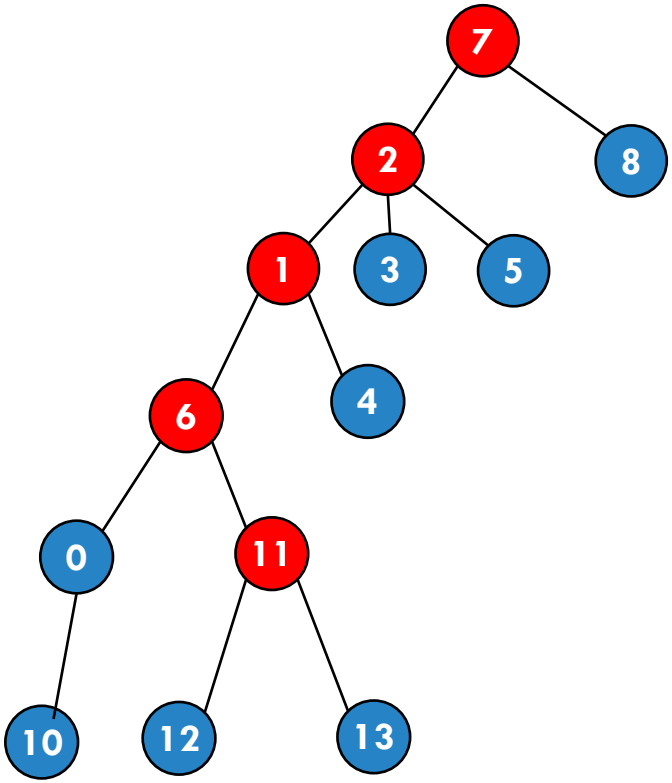
After finding the root, set the parent of each traversed node to the root.



PATH COMPRESSION

Idea: Flatten the tree after find

After finding the root, set the parent of each traversed node to the root.

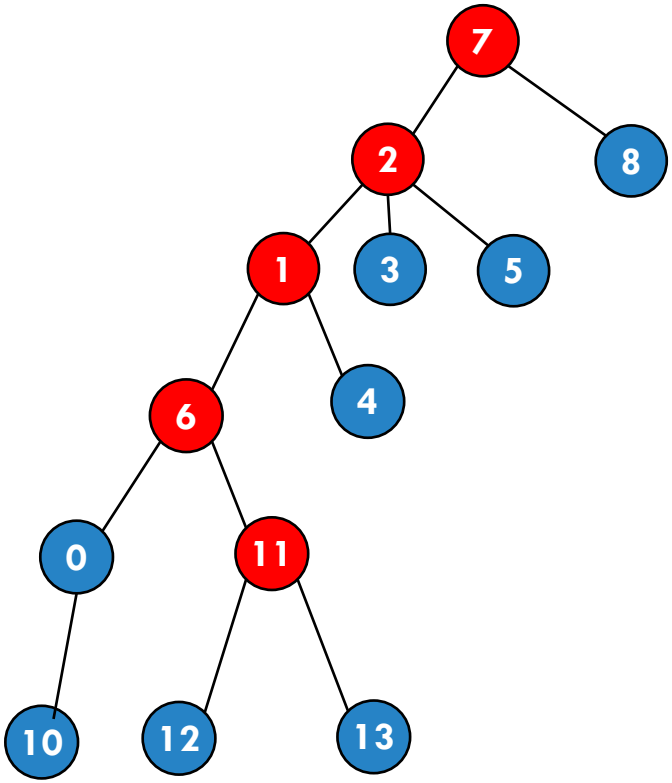


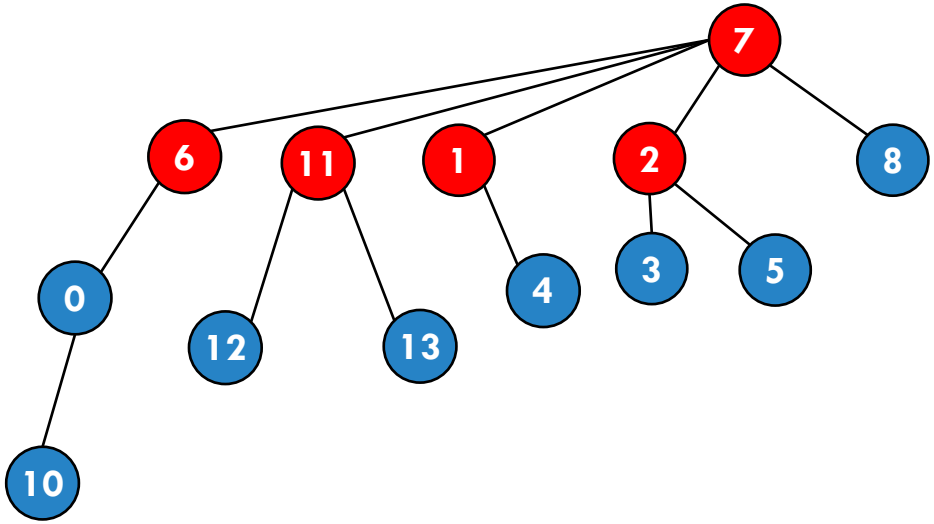

```
find(11, 32)
```

PATH COMPRESSION

Idea: Flatten the tree after find

After finding the root, set the parent of each traversed node to the root.





`find(11, 32)`

PATH COMPRESSION

```
findRoot(int p) {
 root = p;
 while (parent[root] != root) root = parent[root];
 return root;
}
```

PATH COMPRESSION

```
findRoot(int p) {
 root = p;
 while (parent[root] != root) root = parent[root];
 // change the parents to the root
 while (parent[p] != p) {
      temp = parent[p];
      parent[p] = root;
      p = temp;
 }
 return root;
}
```

assign a temp to its parent

change its parent to the root

assign p to become its parent

ALTERNATIVE PATH COMPRESSION

```
findRoot(int p) {
    root = p;
    while (parent[root] != root) {
      parent[root] = parent[parent[root]];
      root = parent[root];
    }
    return root;
}
```

Idea: Make every other node in the path point to its grandparent!

- Simple and Works as well!

WEIGHTED UNION WITH PATH COMPRESSION

Theorem [Tarjan 1975]

Any sequence of m union/find operations on n objects takes: $O(n + m\alpha(m, n))$ time.

Robert Tarjan
Professor
Princeton University

n	α(n, n)
4	0
8	1
32	2
8,192	3
2^{65533}	4

Inverse Ackermann function: always ≤ 5 in this universe.

WEIGHTED UNION WITH PATH COMPRESSION

Theorem [Tarjan 1975]

Any sequence of m union/find operations on n objects takes: $O(n + m\alpha(m, n))$ time.

Proof

Very difficult (beyond CS2040S)

Also, proof shows linear time impossible.

Robert Tarjan
Professor
Princeton University

UNION-FIND: SUMMARY

Weighted-union is faster:

- Trees are flat: $O(\log n)$
- Union *and* find are $O(\log n)$

Weighted Union + Path Compression is very fast:

- Trees very flat.
- On average, almost linear performance per operation.

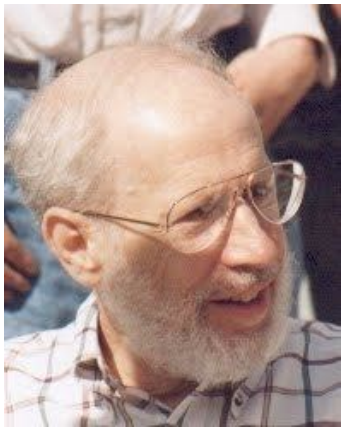
	find	union
quick-find	$O(1)$	$O(n)$
quick-union	$O(n)$	$O(n)$
weighted-union	$O(\log n)$	$O(\log n)$
path compression	$O(\log n)$	$O(\log n)$
weighted-union with path-compression	$\alpha(m, n)$	$\alpha(m, n)$

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

KRUSKAL'S ALGORITHM (Kruskal 1956)



1925-2010

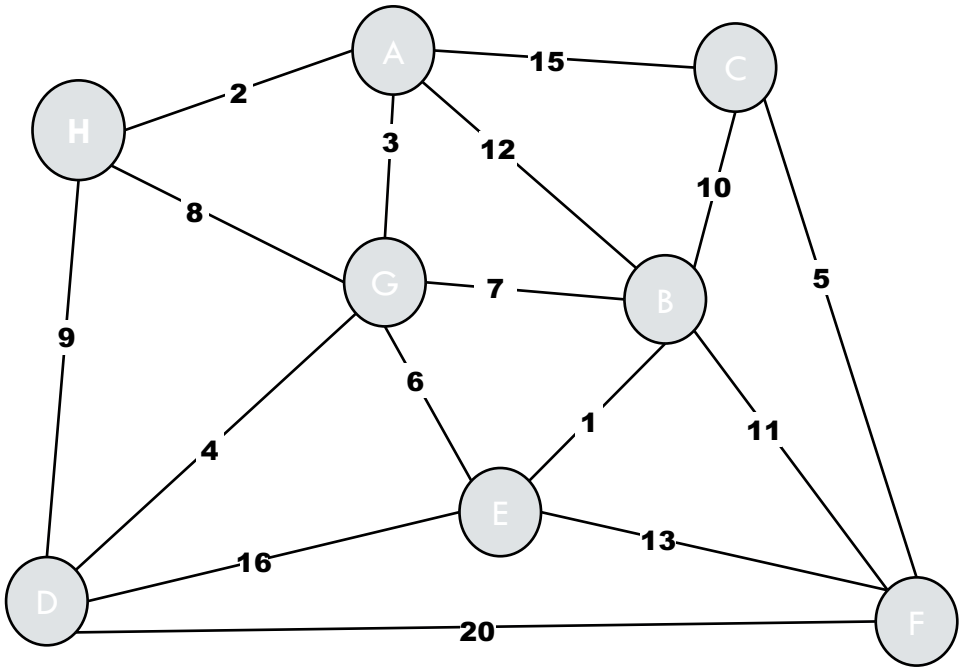
Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
  - Remove minimum weight edge from S
  - If removed edge connects two trees
    - add it to the F (combine the trees)

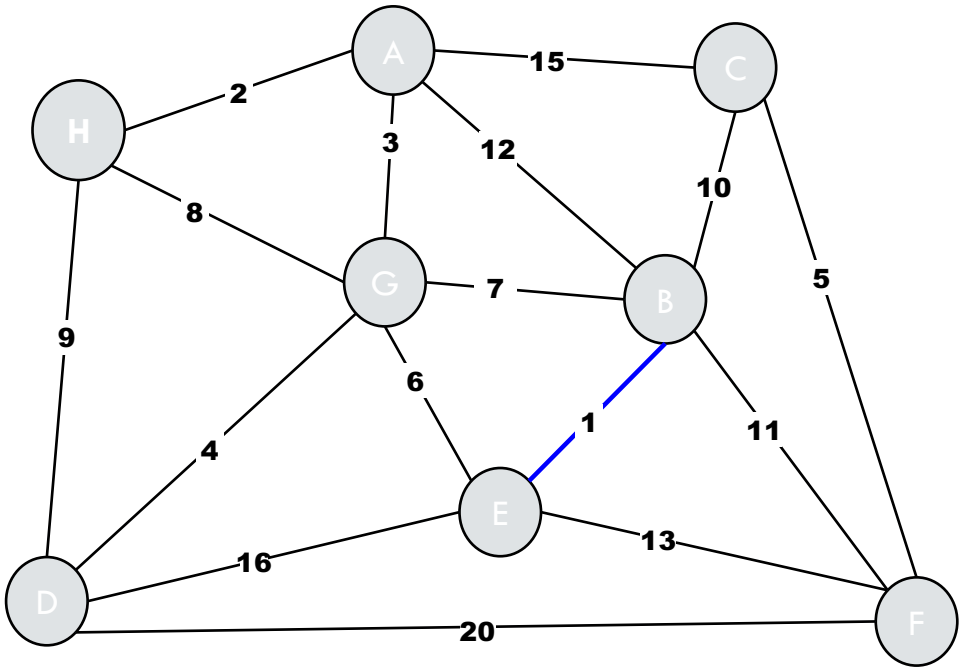
How can we quickly find out if an edge connect two trees?
How can we combine trees?

KRUSKAL'S ALGORITHM

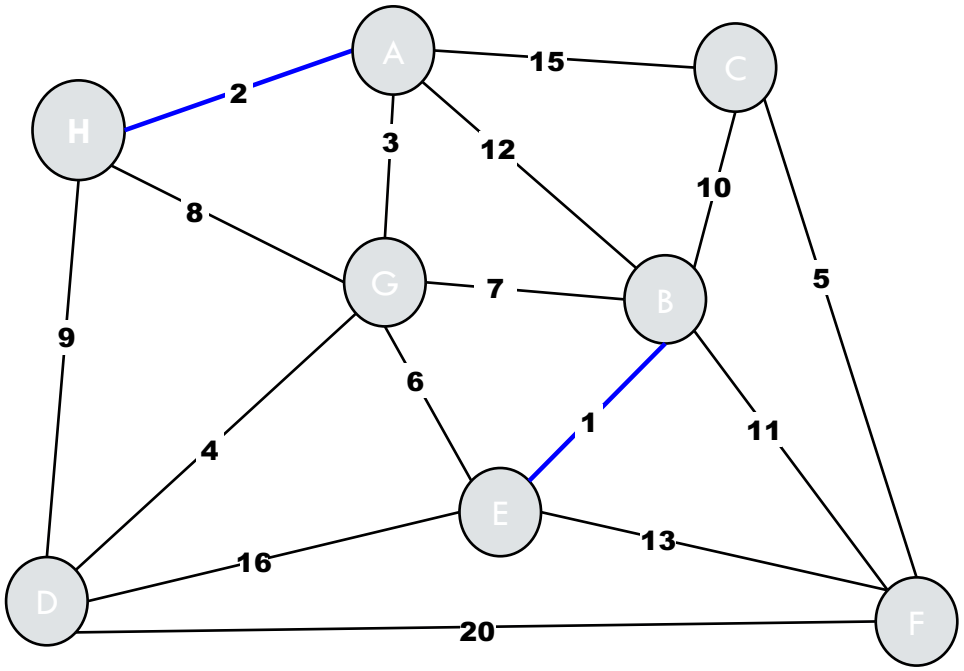
```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());

// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
    Edge e = sortedEdges[i]; // get edge
    Node v = e.one(); // get node endpoints
    Node w = e.two();

    if (!uf.find(v,w)) { // in the same tree?
       mstEdges.add(e); // save edge
       uf.union(v,w); // combine trees
    }
}
```



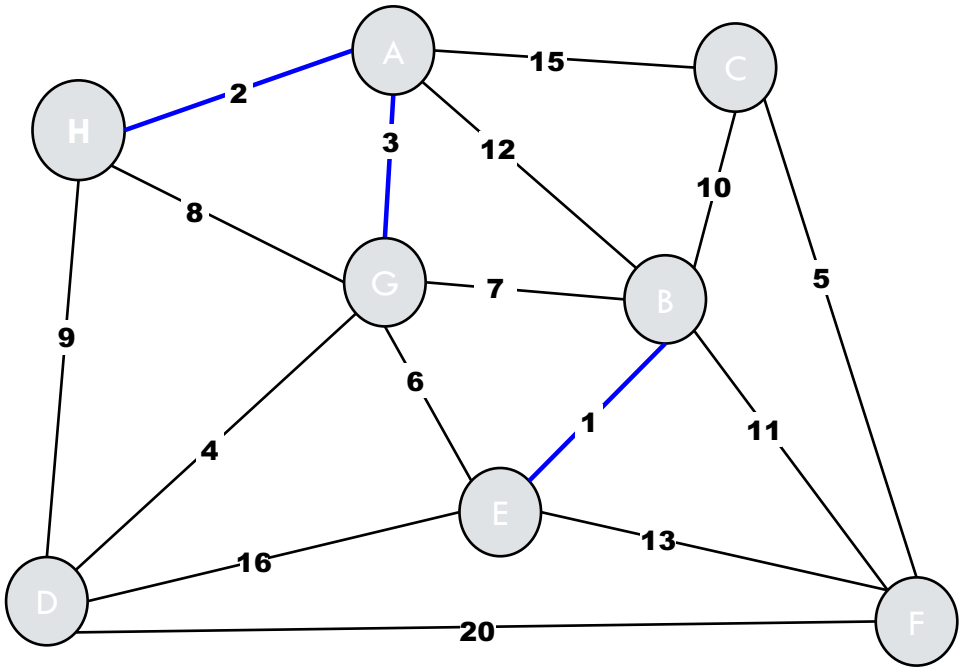
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



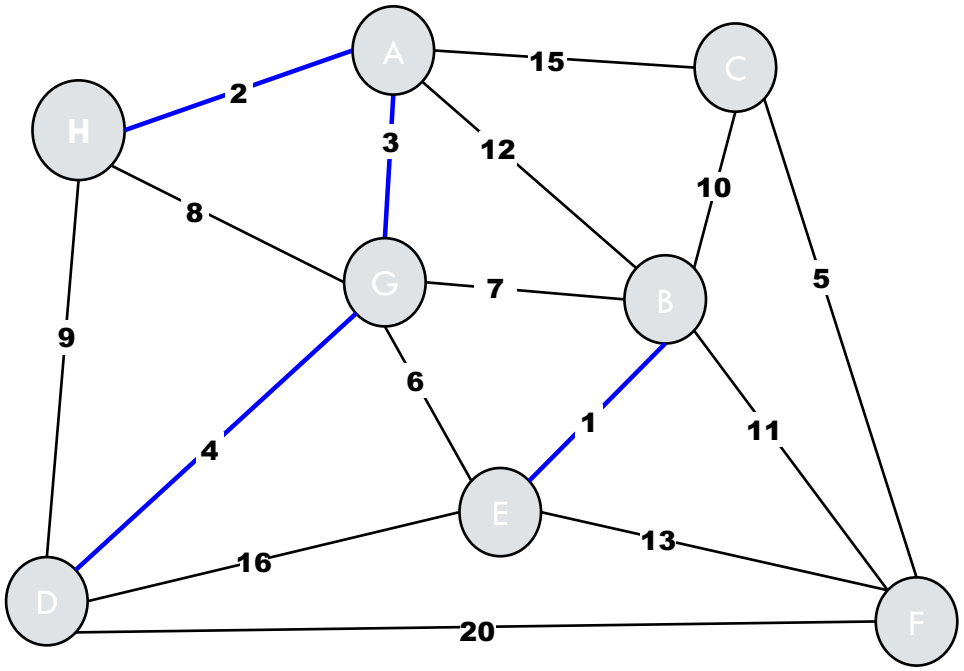
Weight	Edge
1	**(E,B)**
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



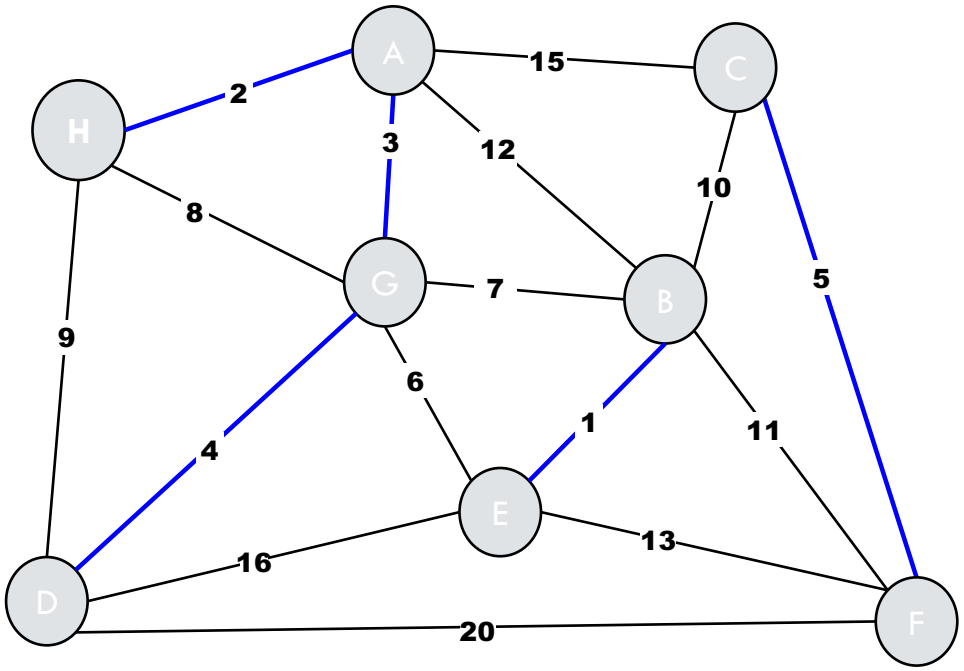
Weight	Edge
1	(E,B)
2	**(A,H)**
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



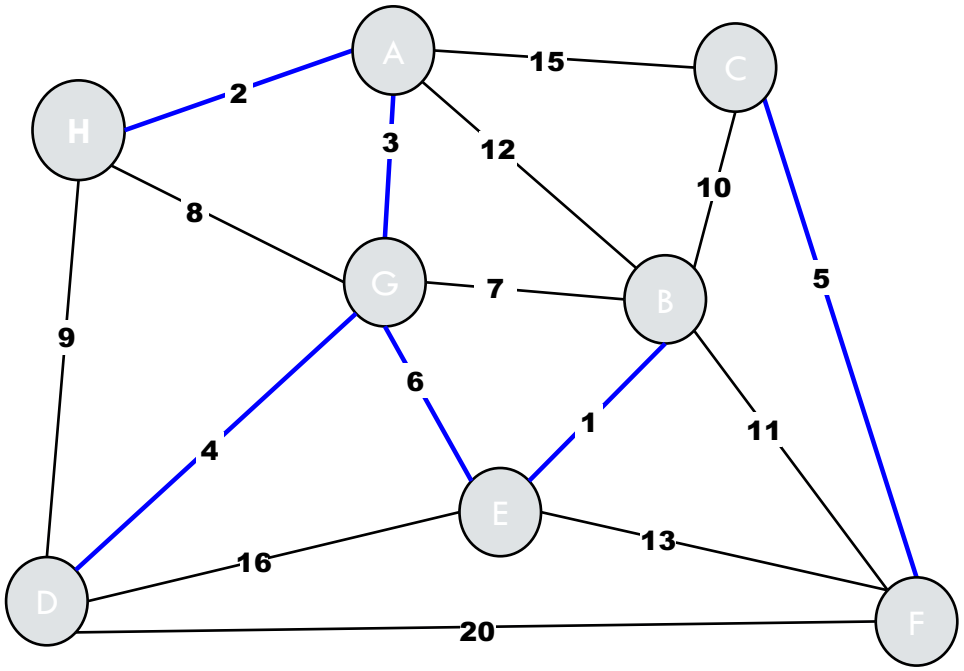
Weight	Edge
1	(E,B)
2	(A,H)
3	**(A,G)**
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



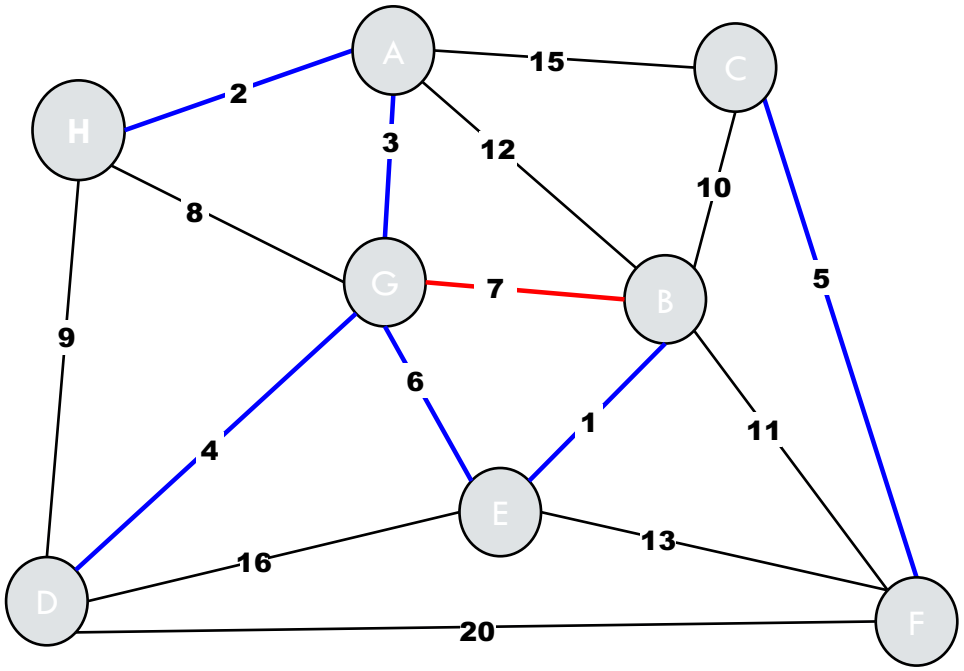
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	**(D,G)**
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



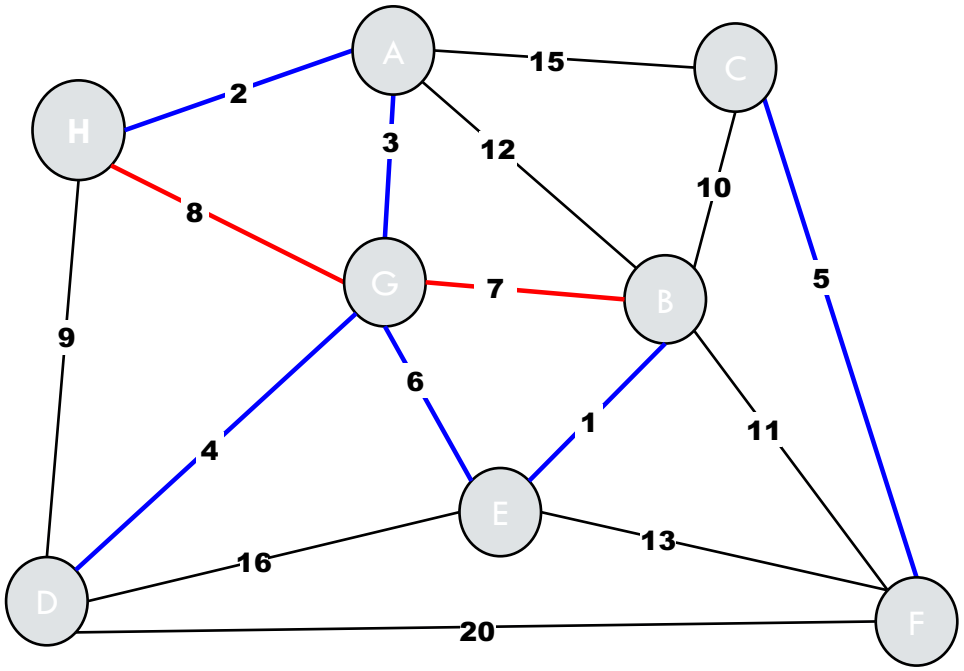
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	**(C,F)**
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



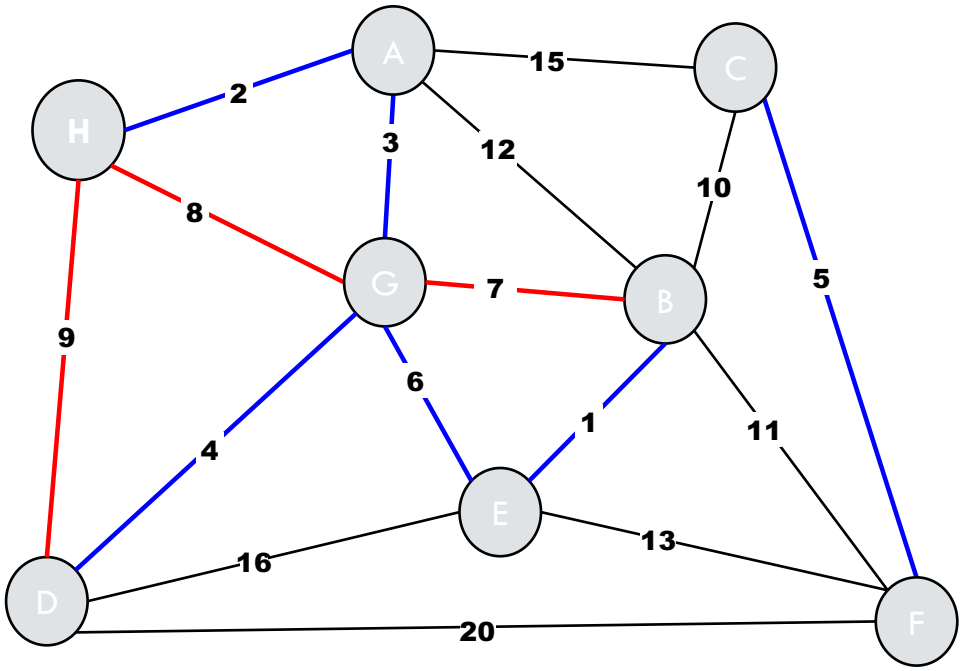
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	**(E,G)**
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



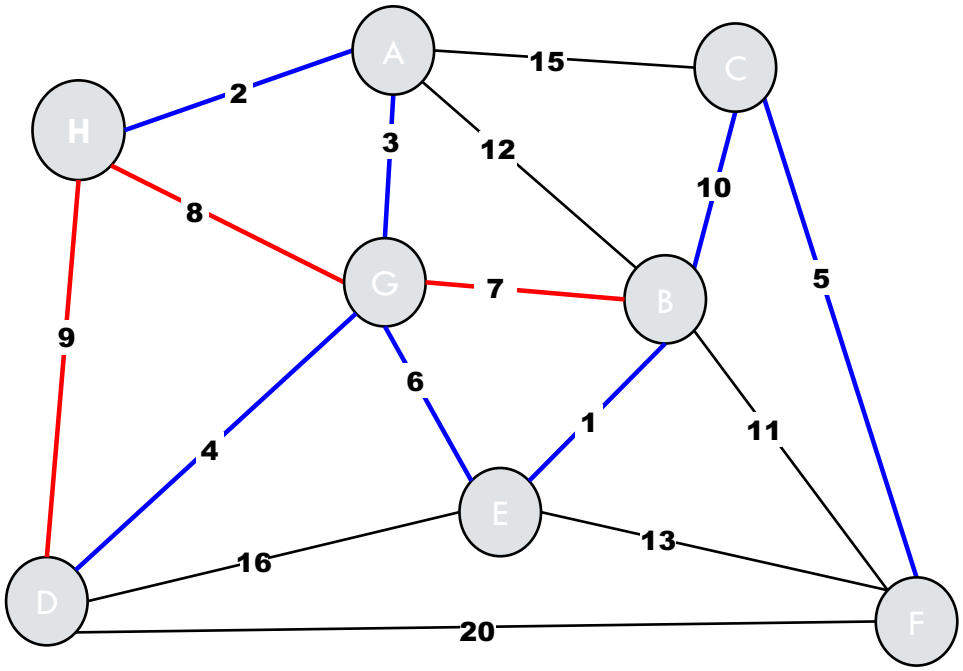
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	**(B,G)**
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



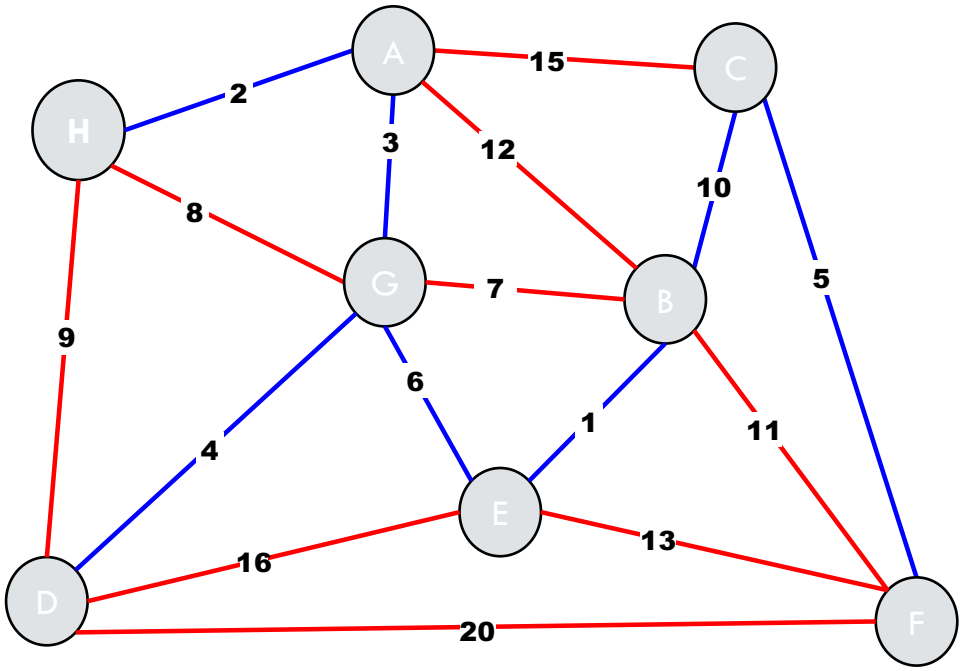
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	**(G,H)**
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	**(D,H)**
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

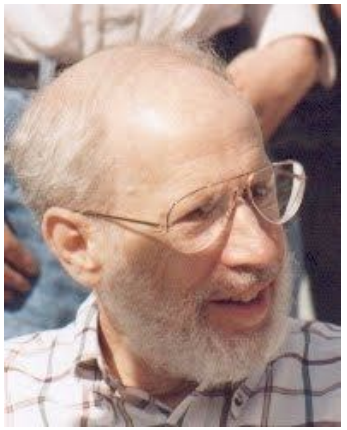


Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

KRUSKAL'S ALGORITHM (Kruskal 1956)



1925-2010

Basic idea:

- Graph F : a set of trees (initially each vertex is a separate tree)
- Set of edges $S = \{e \in E\}$
- While S is nonempty and F is not spanning:
  - Remove minimum weight edge from S
  - If removed edge connects two trees
    - add it to the F (combine the trees)

Proof Sketch:

Each added edge crosses a cut.

Each edge is the lightest edge across the cut: all other lighter edges across the cut have already been considered.

KRUSKAL'S ALGORITHM

```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());
// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
      Edge e = sortedEdges[i]; // get edge
      Node v = e.one(); // get node endpoints
      Node w = e.two();
      if (!uf.find(v,w)) { // in the same tree?
         mstEdges.add(e); // save edge
         uf.union(v,w); // combine trees
      }
}
```

What is the time complexity for Kruskal's algorithm?

A. $O(E)$
B. $O(E \log E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(E\ \alpha)$
F. $O(V\ \alpha)$
G. I really have no idea.

KRUSKAL'S ALGORITHM

```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());
// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
      Edge e = sortedEdges[i]; // get edge
      Node v = e.one(); // get node endpoints
      Node w = e.two();
      if (!uf.find(v,w)) { // in the same tree?
         mstEdges.add(e); // save edge
         uf.union(v,w); // combine trees
      }
}
```

What is the time complexity for Kruskal's algorithm?

A. $O(E)$
B. $O(E \log E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(E\ \alpha)$
F. $O(V\ \alpha)$
G. I really have no idea.

KRUSKAL'S ALGORITHM

```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());
// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
      Edge e = sortedEdges[i]; // get edge
      Node v = e.one(); // get node endpoints
      Node w = e.two();
      if (!uf.find(v,w)) { // in the same tree?
         mstEdges.add(e); // save edge
         uf.union(v,w); // combine trees
      }
}
```

Sort takes $O(E \log E)$

$$O(E \log E) = O(E \log V^2) = O(E \log V)$$

For E edges, find/union take $O(E\alpha)$

Total: $O(E \log V)$

MST PERFORMANCE

Classic greedy algorithms: $O(E \log V)$

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known: $O(E\ \alpha(E, V))$

- Chazelle (1997, 1999,2000)

DIRECTED MST

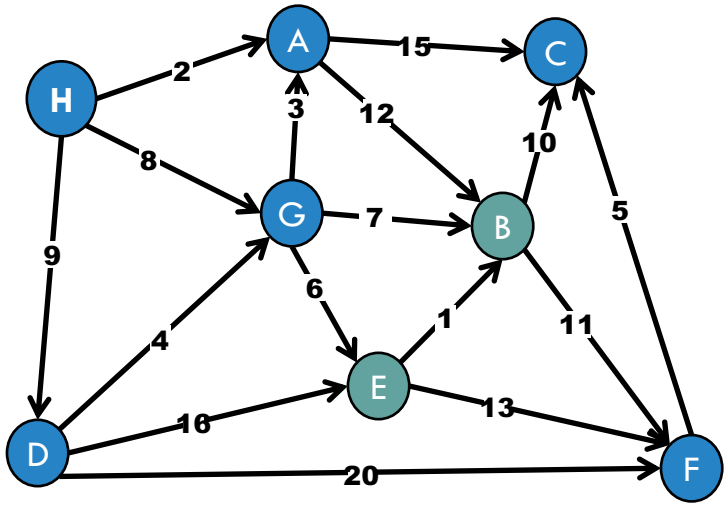
What if the edges were *directed*?

Harder problem:

- Cut property does not hold.
- Cycle property does not hold.

Prim's / Kruskal's do not work.

See CS3230 / CS5234 for more details…



QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

MST VARIATIONS

What if all the edges have the same weight?

Poll Everywhere

https://bit.ly/2LvG9bq

how fast can you find the MST if all edges have equal weight?

A. $O(V + E)$
B. $O(E \log E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(E\ \alpha)$
F. $O(V\ \alpha)$
G. What is going on here?

MST VARIATIONS

Poll Everywhere

https://bit.ly/2LvG9bq

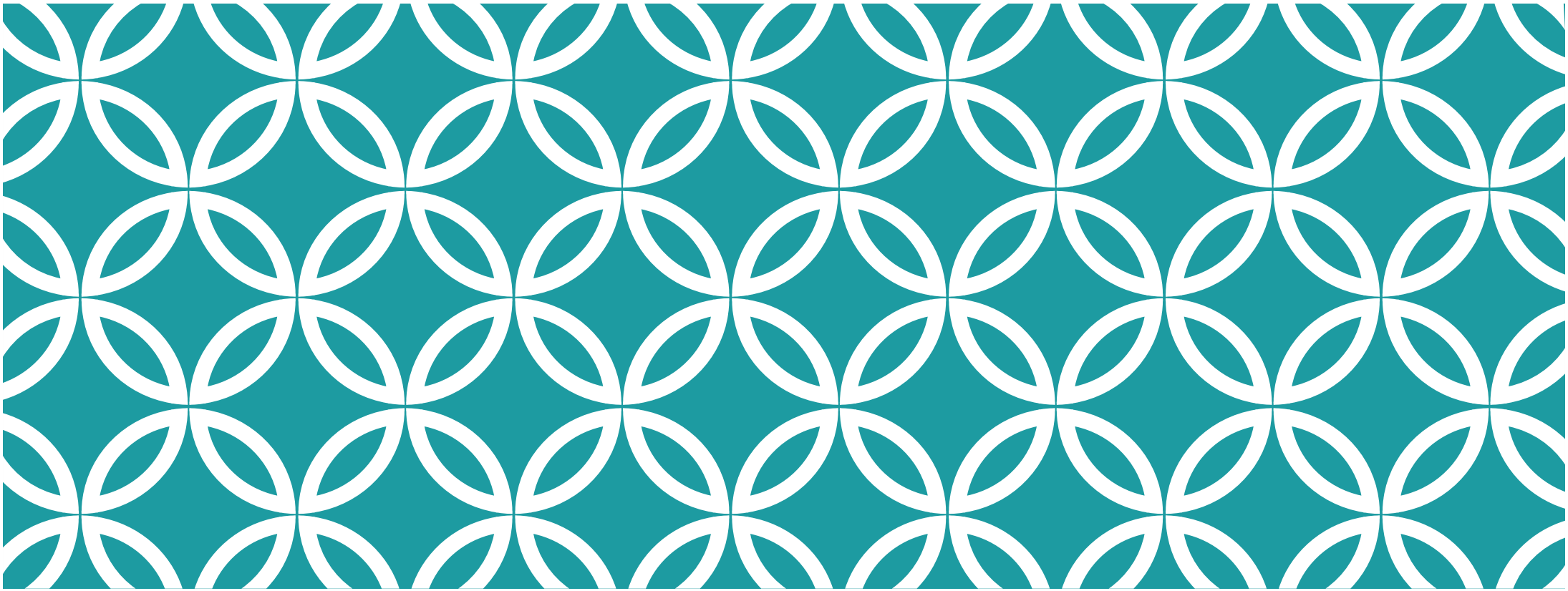
What if all the edges have the same weight?

Use DFS or BFS!

An MST contains exactly $(V-1)$ edges.
Every spanning tree contains $(V-1)$ edges!
Thus, any spanning tree you find with DFS/BFS is a MST.

how fast can you find the MST if all edges have equal weight?

A. $O(V+E)$
B. $O(E \log E)$
C. $O(E \log V)$
D. $O(V \log E)$
E. $O(E\ \alpha)$
F. $O(V\ \alpha)$
G. What is going on here?



SAMPLE PROBLEM: LIMITED EDGE WEIGHTS

Harold Soh
harold@comp.nus.edu.sg

MST: ALL EDGES HAVE INTEGER WEIGHTS

What if all the edges have integer weights from {1..10}?

How fast can you find a MST?

MST: ALL EDGES HAVE INTEGER WEIGHTS

What if all the edges have integer weights from {1..10}?

How fast can you find a MST?

Most straightforward approach:

- Use Prim's or Kruskal's algorithm

Time Complexity:

- $O(E \log V)$

Can we do better?

KRUSKAL'S ALGORITHM

```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());
// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
      Edge e = sortedEdges[i]; // get edge
      Node v = e.one(); // get node endpoints
      Node w = e.two();
      if (!uf.find(v,w)) { // in the same tree?
         mstEdges.add(e); // save edge
         uf.union(v,w); // combine trees
      }
}
```

Sort takes $O(E \log E)$

$$O(E \log E) = O(E \log V^2) = O(E \log V)$$

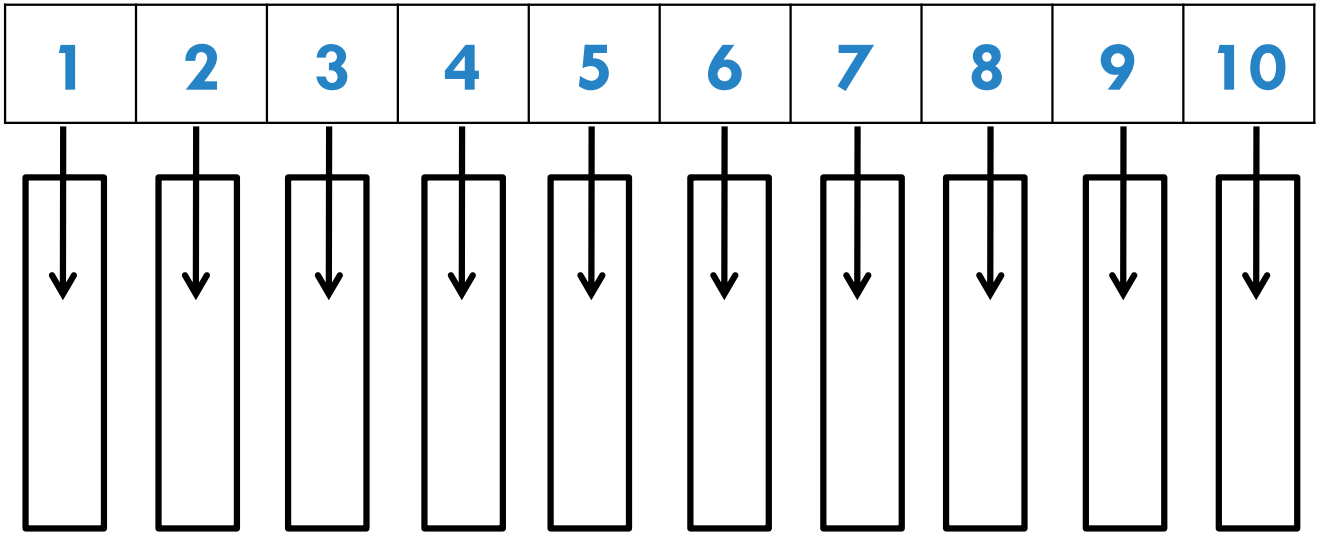
For E edges, find/union take $O(E\alpha)$

Total: $O(E \log V)$

Hmm.. sorting is expensive.
Do we need to sort?

MST: ALL EDGES HAVE INTEGER WEIGHTS

Idea: Use an array of size 10 instead of sorting



slot `A[j]` holds a linked list of edges of weight `j`

KRUSKAL'S ALGORITHM

```
// Sort edges and initialize
Edge[] sortedEdges = sortWithList(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());
// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
      Edge e = sortedEdges[i]; // get edge
      Node v = e.one(); // get node endpoints
      Node w = e.two();
      if (!uf.find(v,w)) { // in the same tree?
         mstEdges.add(e); // save edge
         uf.union(v,w); // combine trees
      }
}
```

"Sort" takes $O(E)$

For E edges, find/union take $O(E\alpha)$

Total: $O(E\alpha)$

PRIM'S ALGORITHM

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

Analysis:

Each vertex added/removed once from the priority queue:

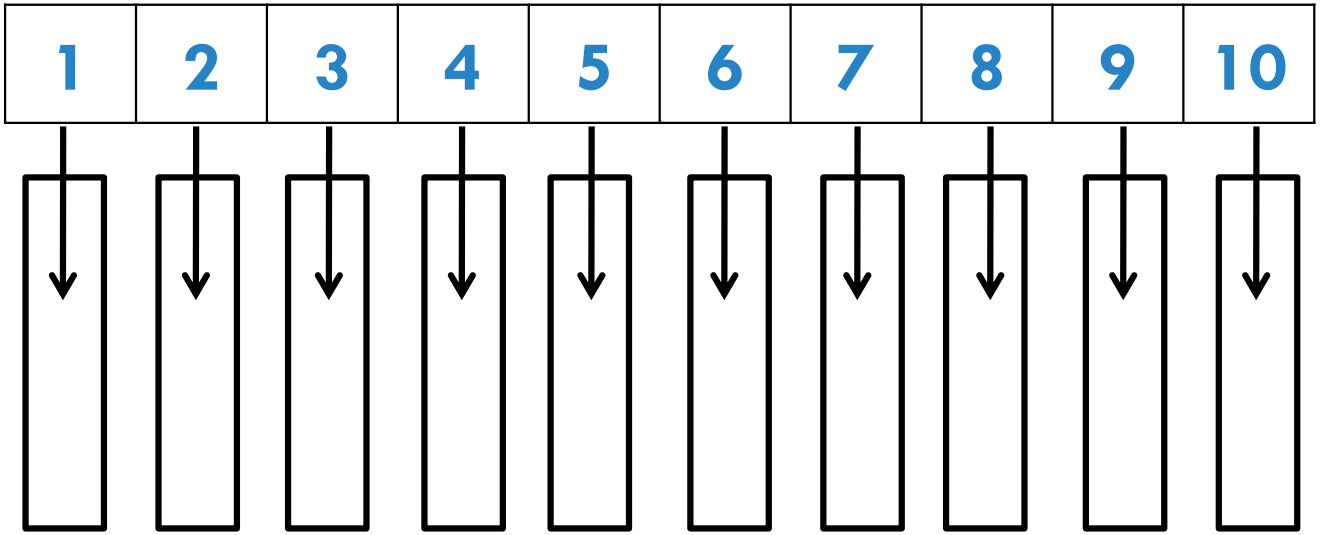
$$O(V \log V)$$

Each edge → one decreaseKey:

$$O(E \log V)$$

MST: ALL EDGES HAVE INTEGER WEIGHTS

Idea: Use an array of size 10 as a **priority queue**



slot `A[j]` holds a linked list of edges of weight `j`

PRIM'S ALGORITHM

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

Analysis:

Each vertex added/removed once from the priority queue:

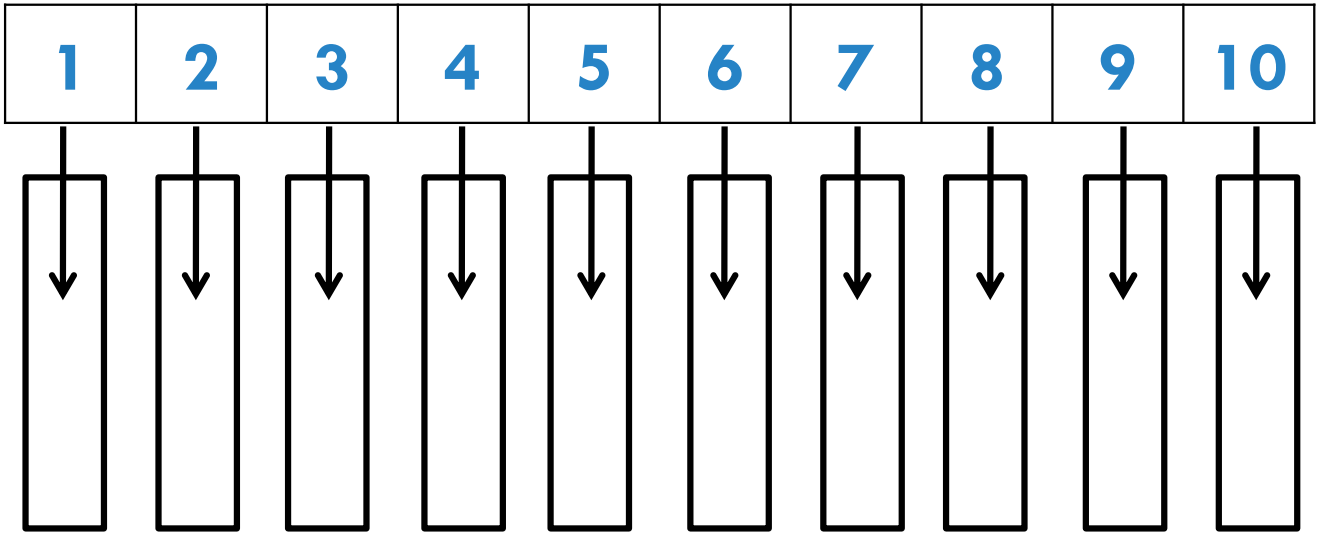
$$O(V)$$

Each edge → one decreaseKey:

MST: ALL EDGES HAVE INTEGER WEIGHTS

Idea: Use an array of size 10 as a **priority queue**

`decreaseKey`: move node to new linked list



slot `A[j]` holds a linked list of edges of weight `j`

PRIM'S ALGORITHM

Basic idea:

- S : set of nodes connected by blue edges.
- Initially: $S = \{A\}$
- Repeat:
  - Identify cut: $\{S, V - S\}$
  - Find minimum weight edge on cut.
  - Add new node to S.

Total: $O(V + E) = O(E)$

Analysis:

Each vertex added/removed once from the priority queue:

$$O(V)$$

Each edge → one decreaseKey:

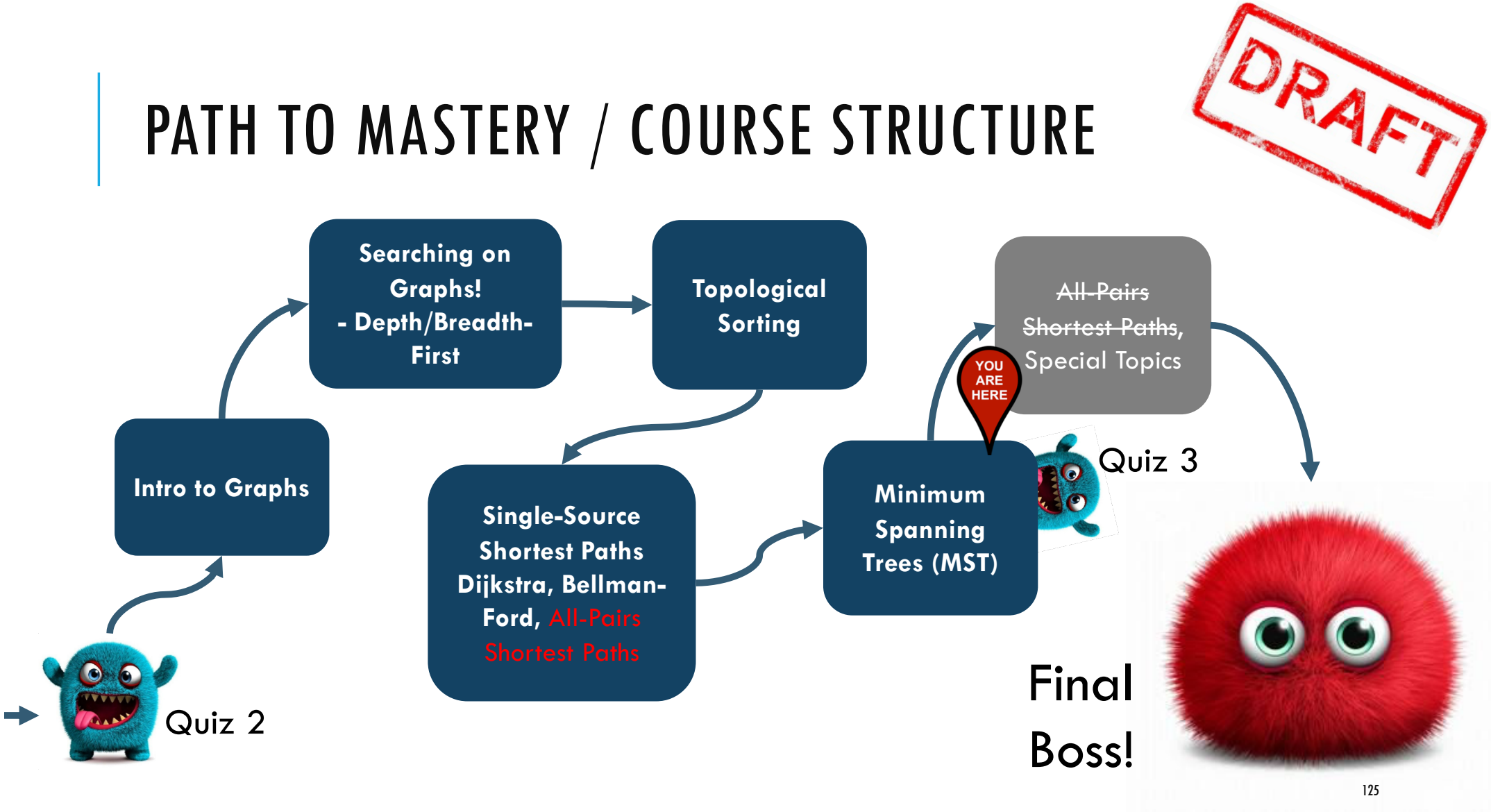
$$O(E)$$

QUESTIONS?

Poll Everywhere

https://bit.ly/2LvG9bq

PATH TO MASTERY / COURSE STRUCTURE



ADMINISTRATIVE ISSUES: QUIZ 3

In lecture tomorrow (Wed)

Please come on time!

- Start at 12pm.
- End at 1:30pm.

Open Book but No electronic equipment.