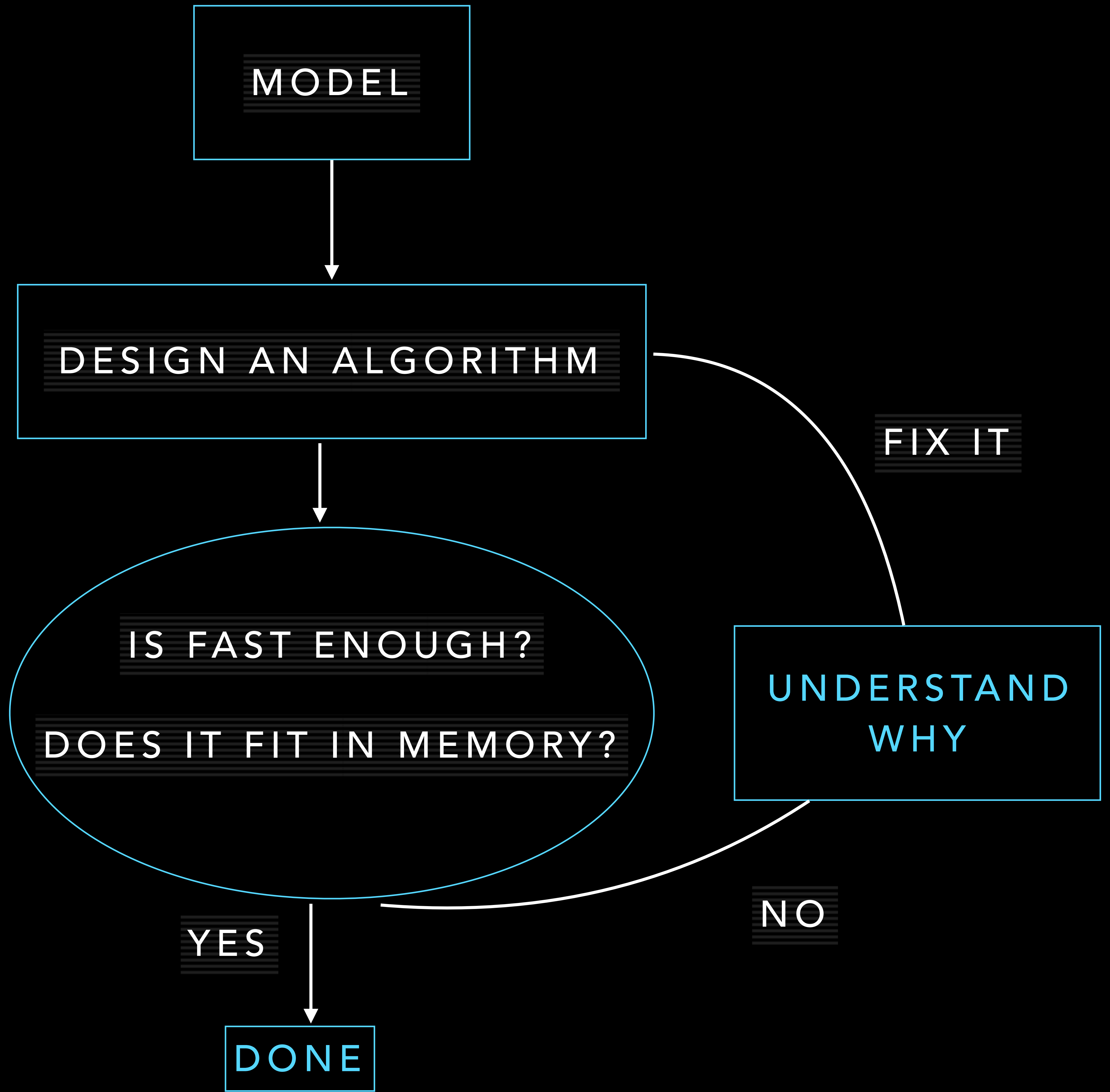
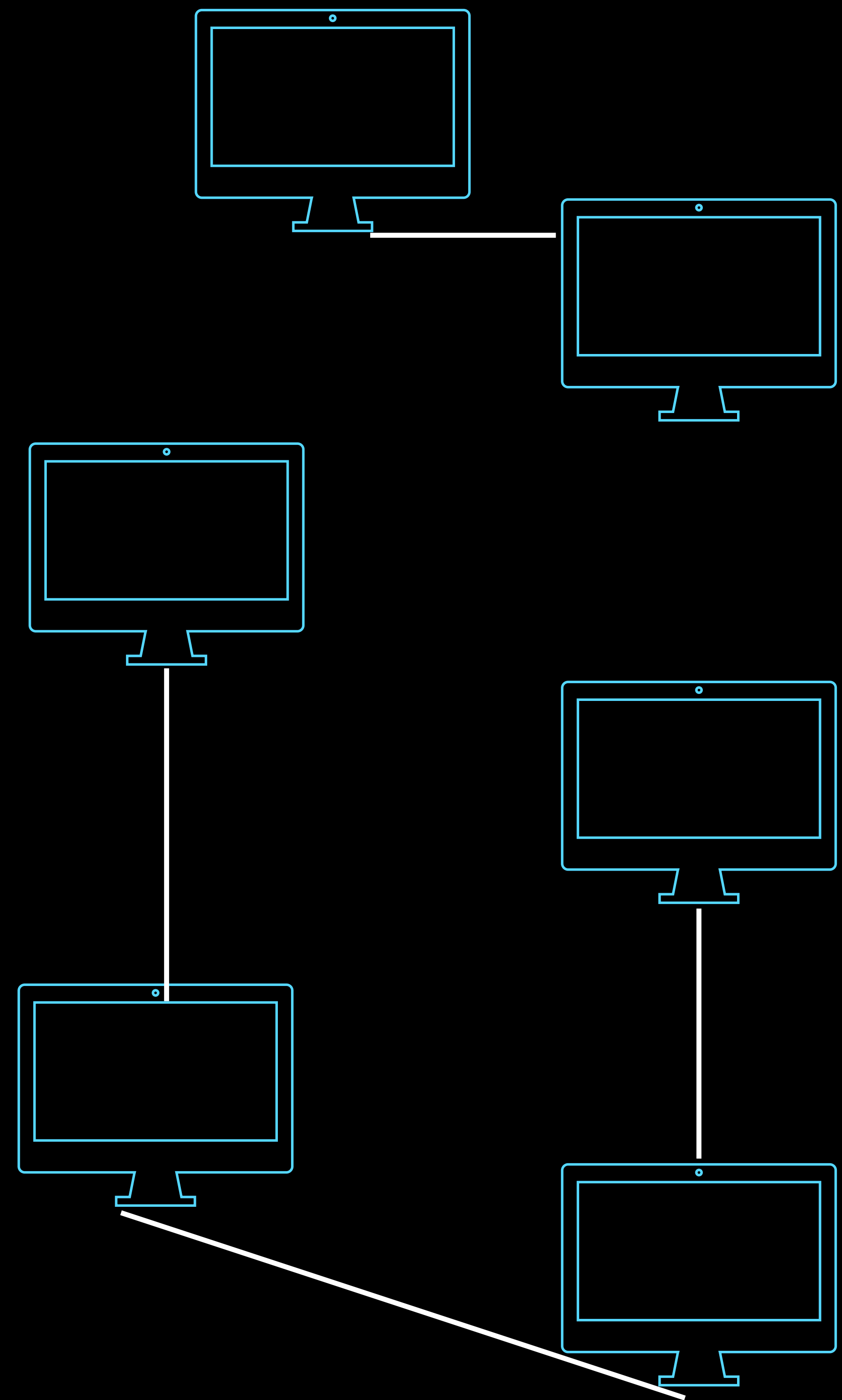
UNION FIND


MODEL
DESIGN AN ALGORITHM
FIX IT
IS FAST ENOUGH?
DOES IT FIT IN MEMORY?
UNDERSTAND
WHY
NO
YES
DONE

MODEL THE PROBLEM

HOW MUCH OF THE NETWORK CAN I INFECT?

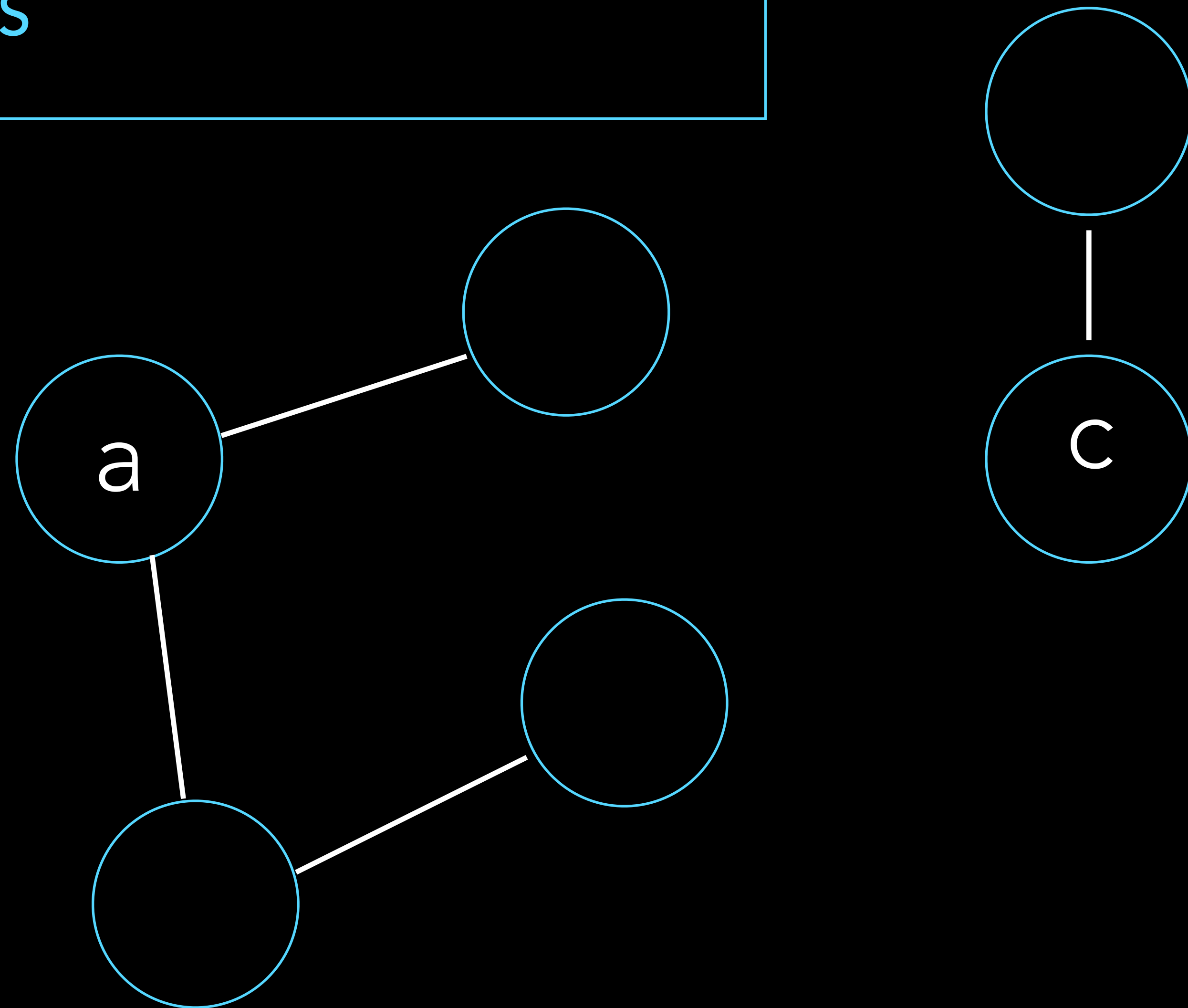


MODEL THE PROBLEM

WE CAN THINK OF IT AS A COLLECTION OF GRAPHS

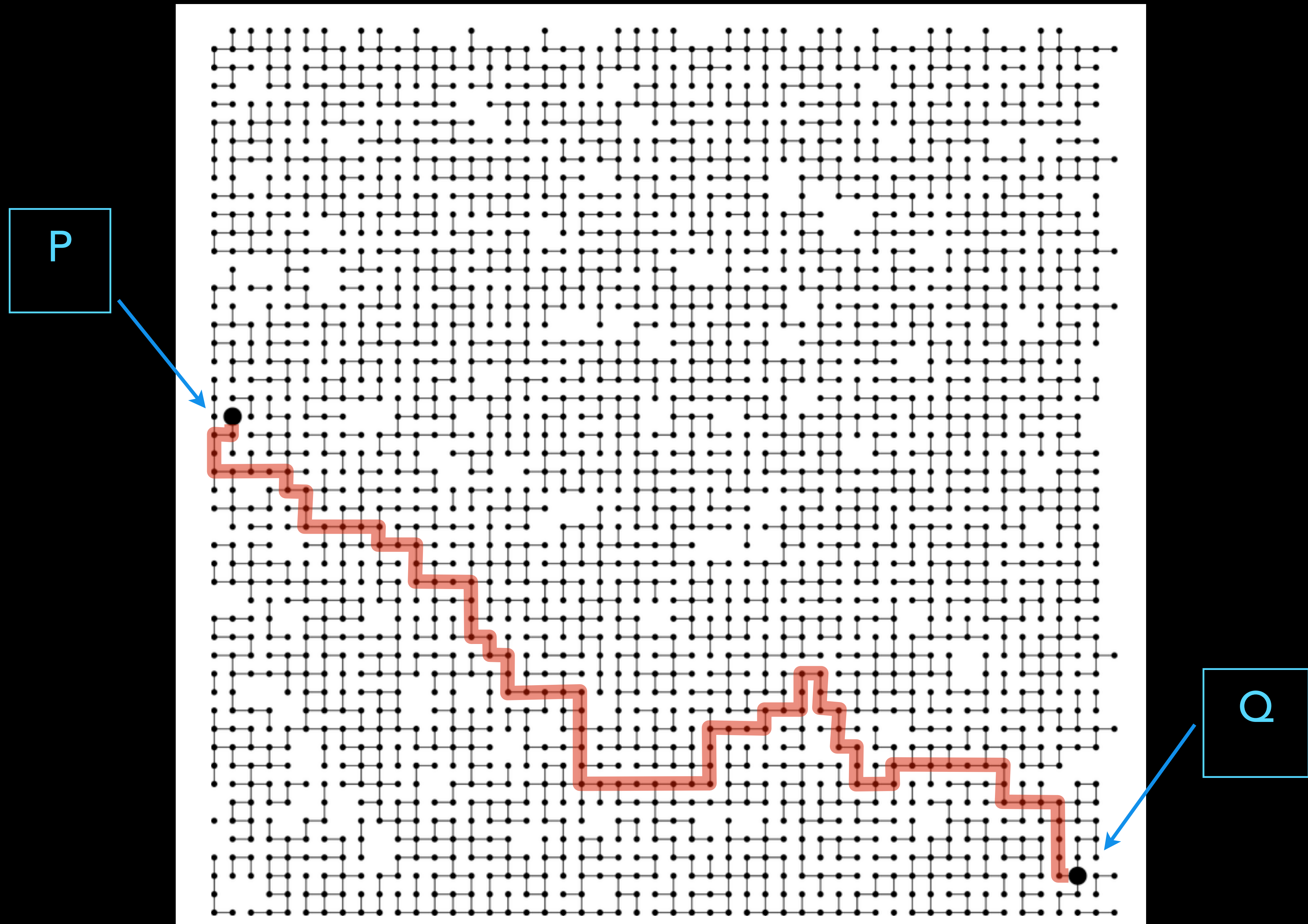
NOW THE PROBLEM BECOMES PATH PROBLEM.

Is there a path from A to C?



What about a larger graph?

Q. Is there a path connecting p and q ?



A. Yes.

DYNAMIC CONNECTIVITY PROBLEM

Given a set of N objects, support two operation:

- Connect two objects.
- Is there a path connecting the two objects?

connect 4 and 3

connect 3 and 8

connect 6 and 5

connect 9 and 4

connect 2 and 1

are 0 and 7 connected? ✗

are 8 and 9 connected? ✔

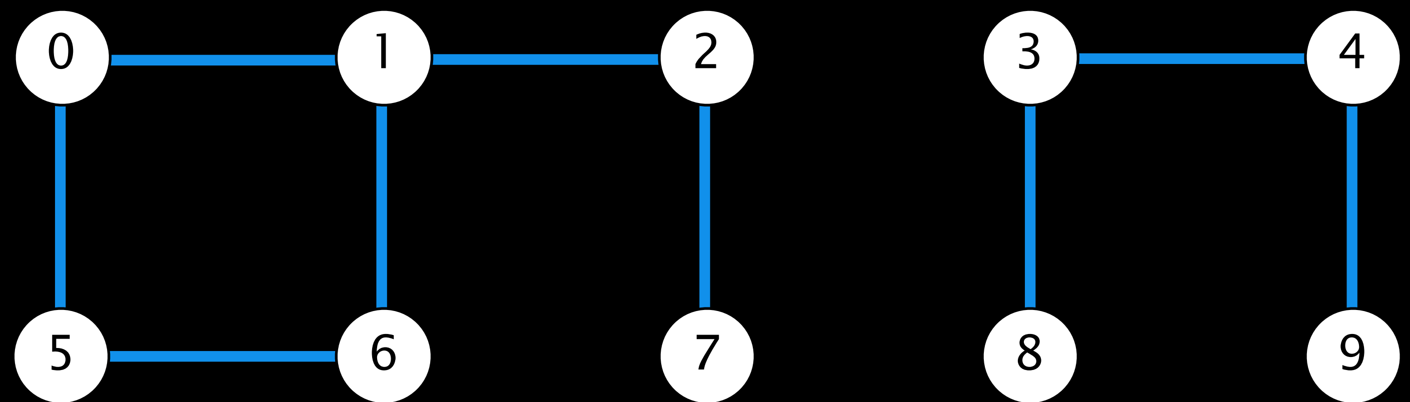
connect 5 and 0

connect 7 and 2

connect 6 and 1

connect 1 and 0

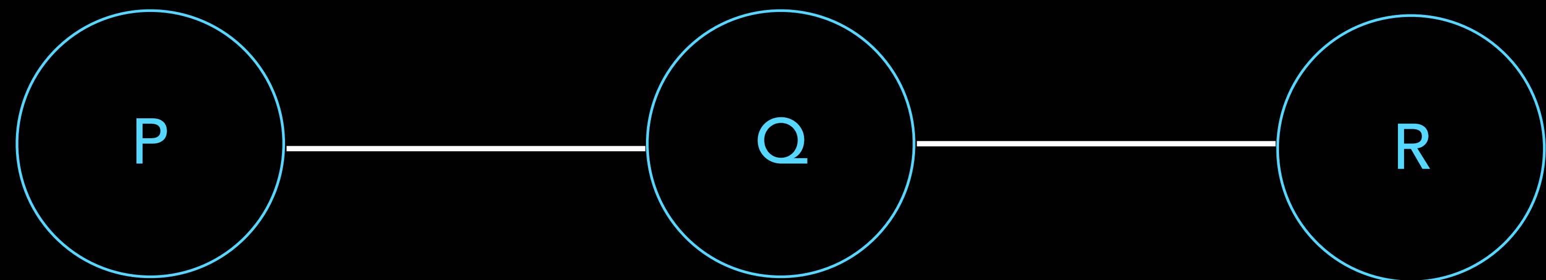
are 0 and 7 connected? ✔



DEFINE WHAT IT MEANS FOR TWO NODES TO BE CONNECTED

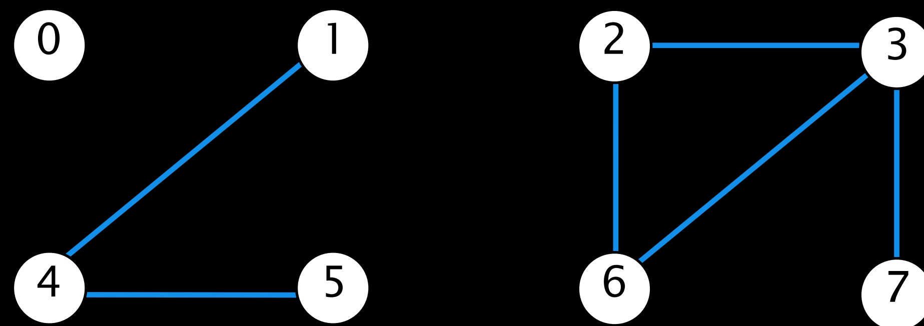
The definition of "connected" follow three properties

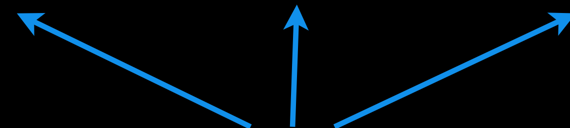
- Reflexive: (p is connect to itself) p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.



DEFINING A CONNECTED COMPONENT

Connected component. Maximal set of objects that are mutually connected



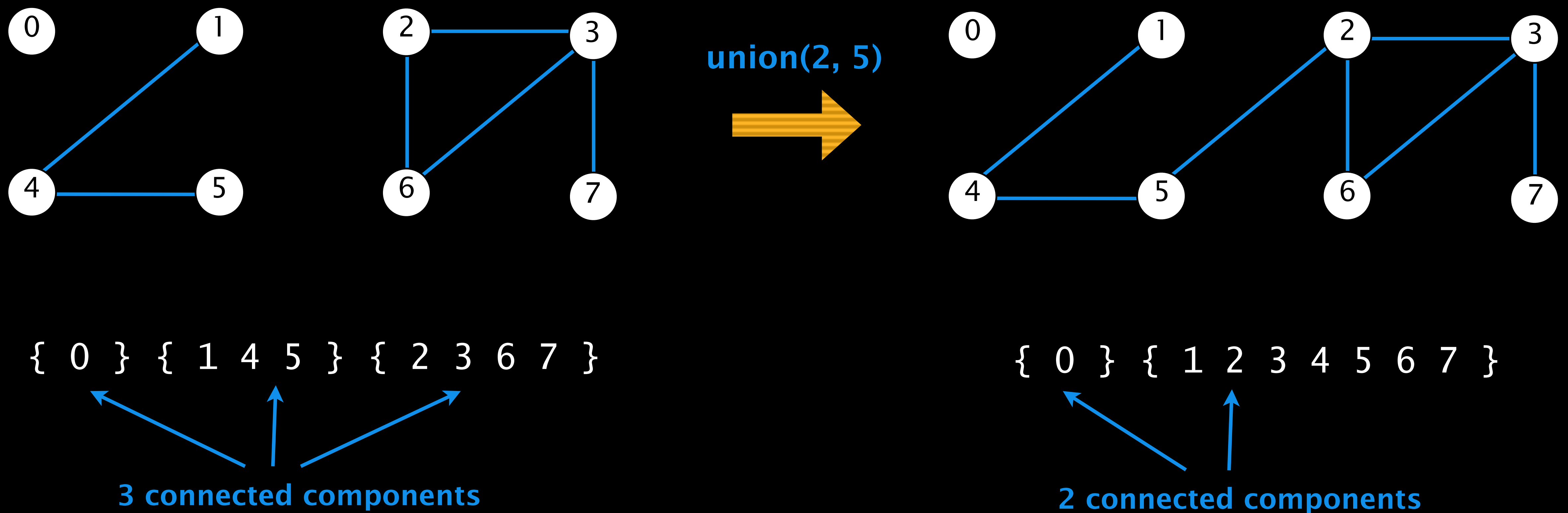


WE WANT TO IMPLEMENT TWO OPERATIONS

- Union place two nodes in the same connected component.
- Find ``query” to a node to find out its connected component.

SO IF NODES SHARE THE SAME COMPONENT THEY ARE CONNECTED

VISUAL EXAMPLE OF UNION OPERATION



UNION FIND DATA STRUCTURE

Goal. Design efficient data structure for union-find.

- Number of objects N can be huge.
- Number of (union and find) operations M can be huge.
- Union and find operations may be intermixed.

UNION FIND DATA STRUCTURE

public class UF	
UF(int N)	*initialize union-find data structure with N singleton objects (0 to $N-1$)*
void union(int p, int q)	*add connection between p and q*
int find(int p)	*component identifier for p (0 to $N-1$)*
boolean connected(int p, int q)	*are p and q in the same component?*

```
public boolean connected(int p, int q)
{  return find(p) == find(q);  }
```

1-line implementation of connected()

EXAMPLE CLIENT

- Read in number of objects N from standard input.
- Repeat:
  - read in pair of integers from standard input
  - if they are not yet connected, connect them and print out pair

```
public static void main(String[] args)
{
   int N = StdIn.readInt();
   UF uf = new UF(N);
   while (!StdIn.isEmpty())
   {
      int p = StdIn.readInt();
      int q = StdIn.readInt();
      if (!uf.connected(p, q))
      {
         uf.union(p, q);
         StdOut.println(p + " " + q);
      }
   }
}
```

```
% more tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
```

LET'S IMPLEMENT THE FIND METHOD

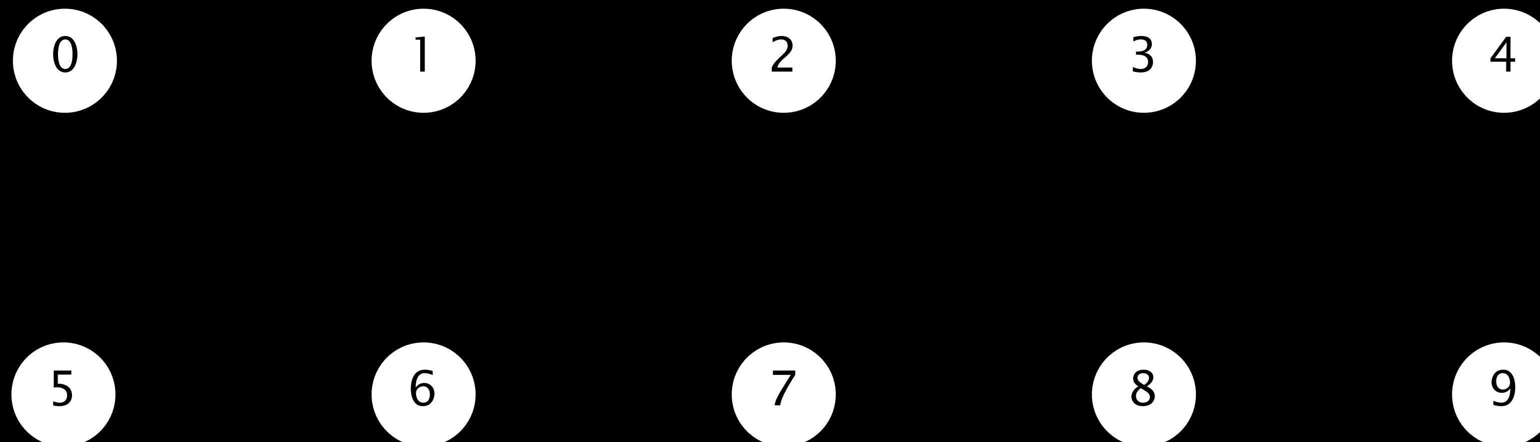
BUT BEFORE WE DO, LET'S THINK ABOUT HOW WE WILL REPRESENT THE GRAPH?

COULD WE USE AN ARRAY

Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` is the id of the component containing `p`.

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



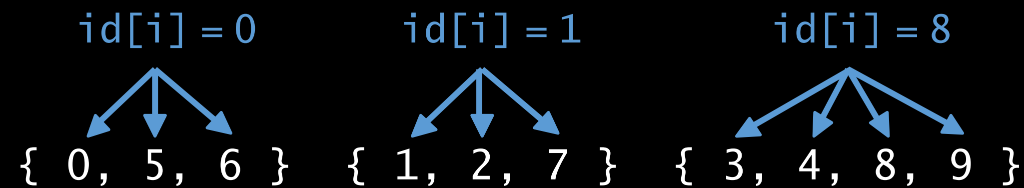
LETS CONSIDER AN EXAMPLE WITH CONNECTIONS

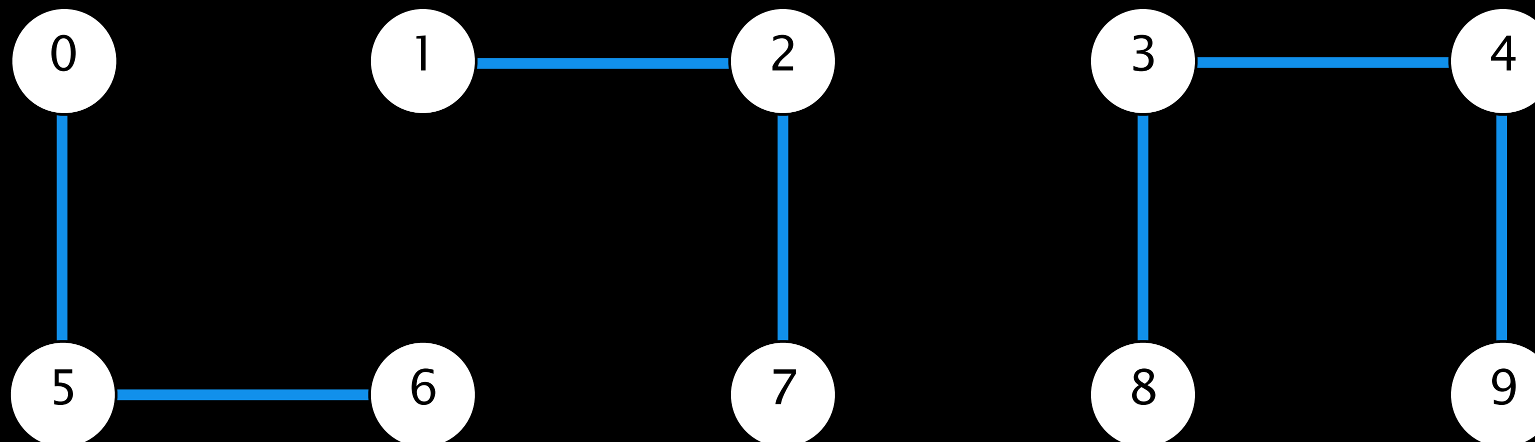
Data structure.

- Integer array `id[]` of length `N`.
- Interpretation: `id[p]` is the id of the component containing `p`.

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

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected





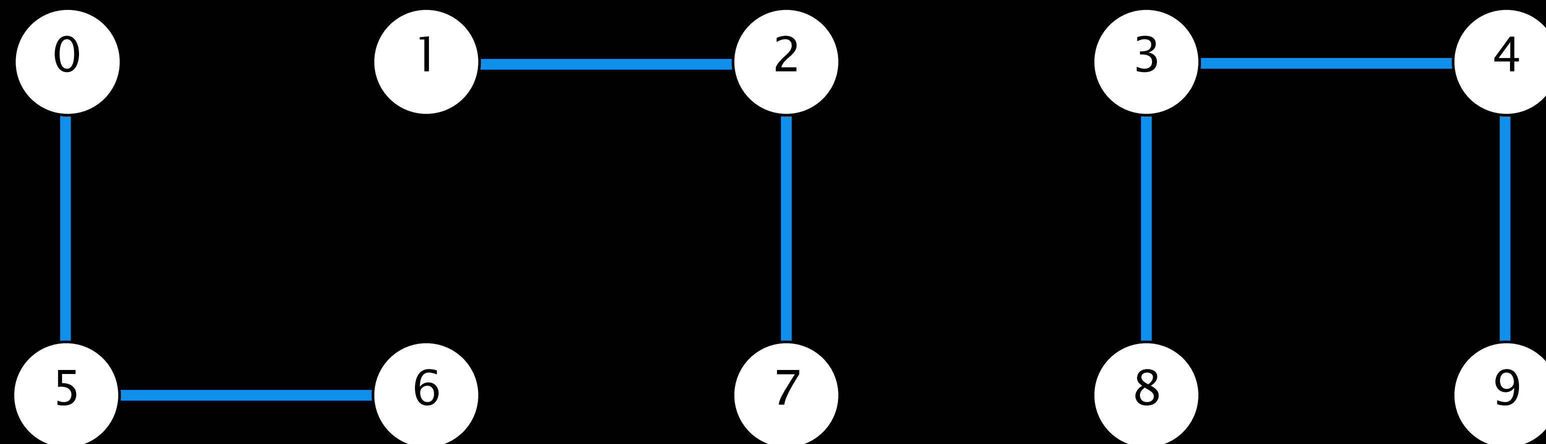
IMPLEMENTING FIND IS STRAIGHT FORWARD

Implement find by looking at the id[i]

IMPLEMENTING UNION IS NOT AS STRAIGHT FORWARD

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

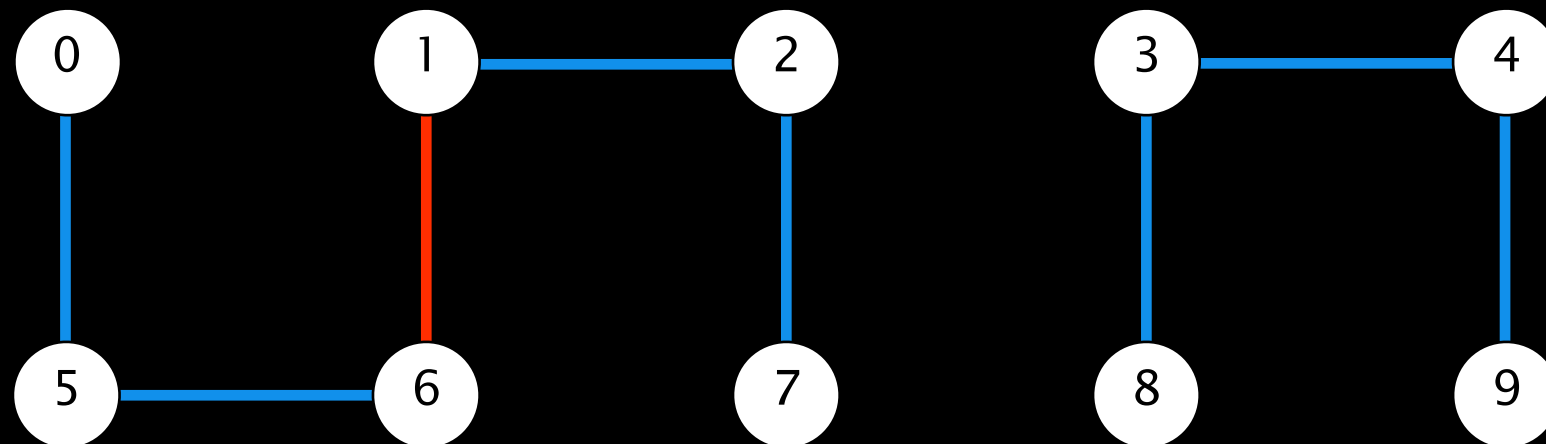
What happens after union of 6 and 1



IMPLEMENTING UNION IS NOT AS STRAIGHT FORWARD

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

after union of 6 and 1

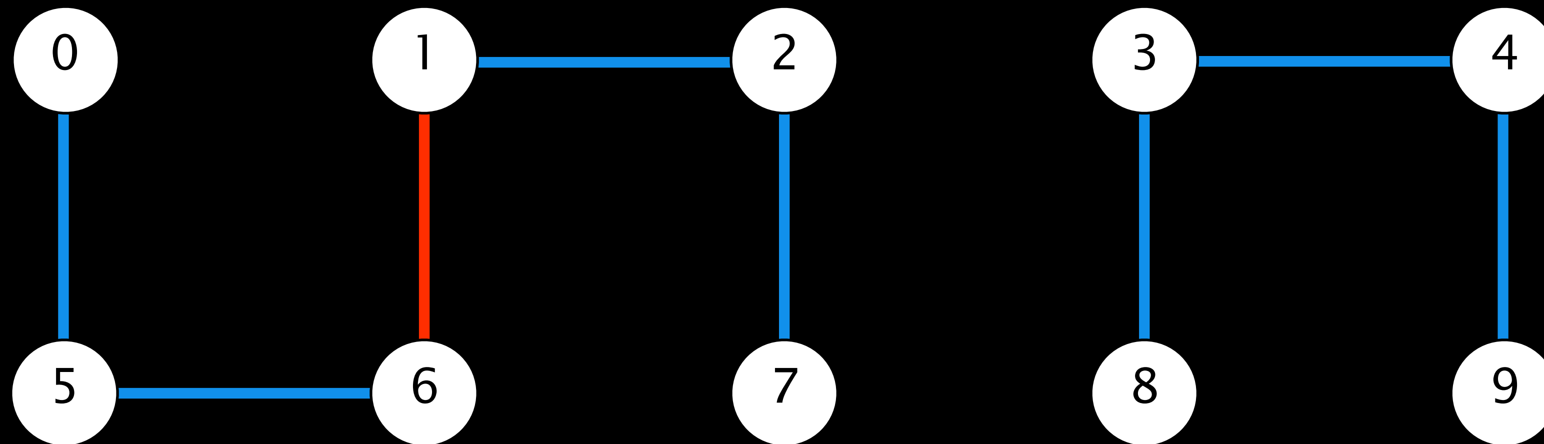


IMPLEMENTING UNION IS NOT AS STRAIGHT FORWARD

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

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

after union of 6 and 1



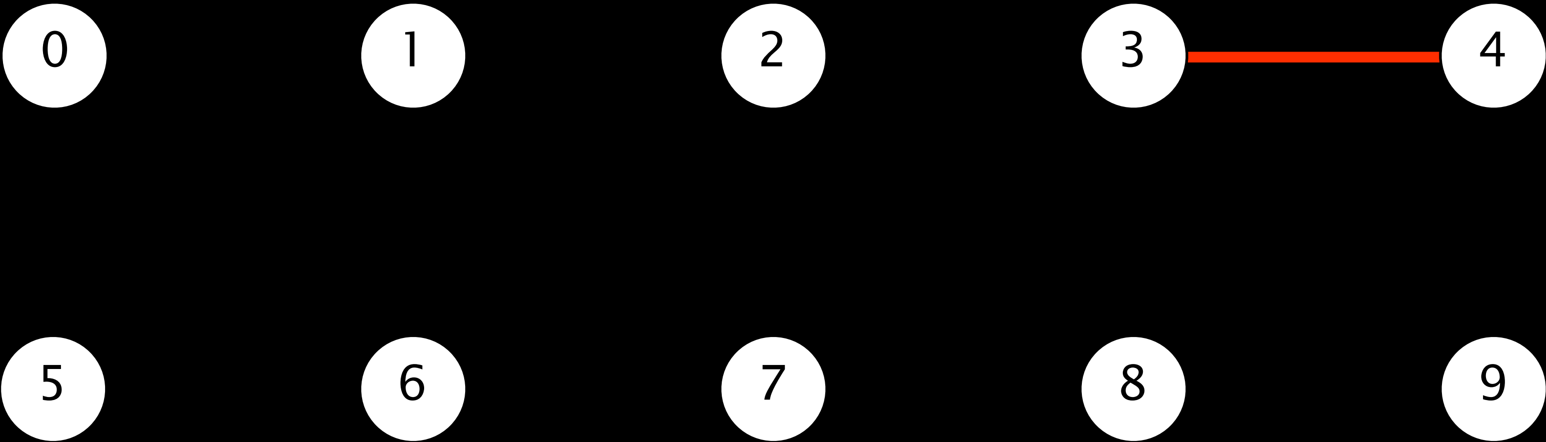
NOT CONSTANT TIME

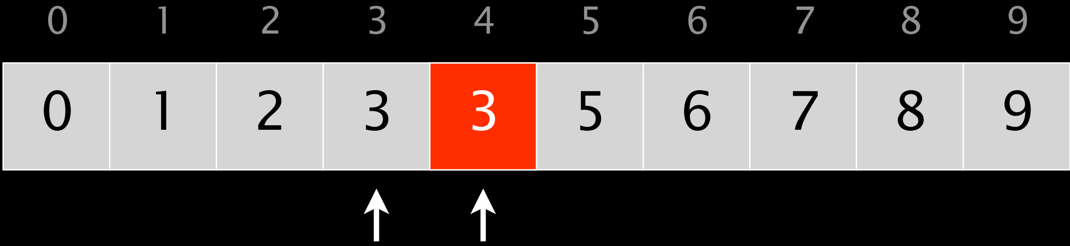
problem: many values can change

LET'S STEP THROUGH AN EXAMPLE

QUICK-FIND DEMO

UNION(4, 3)

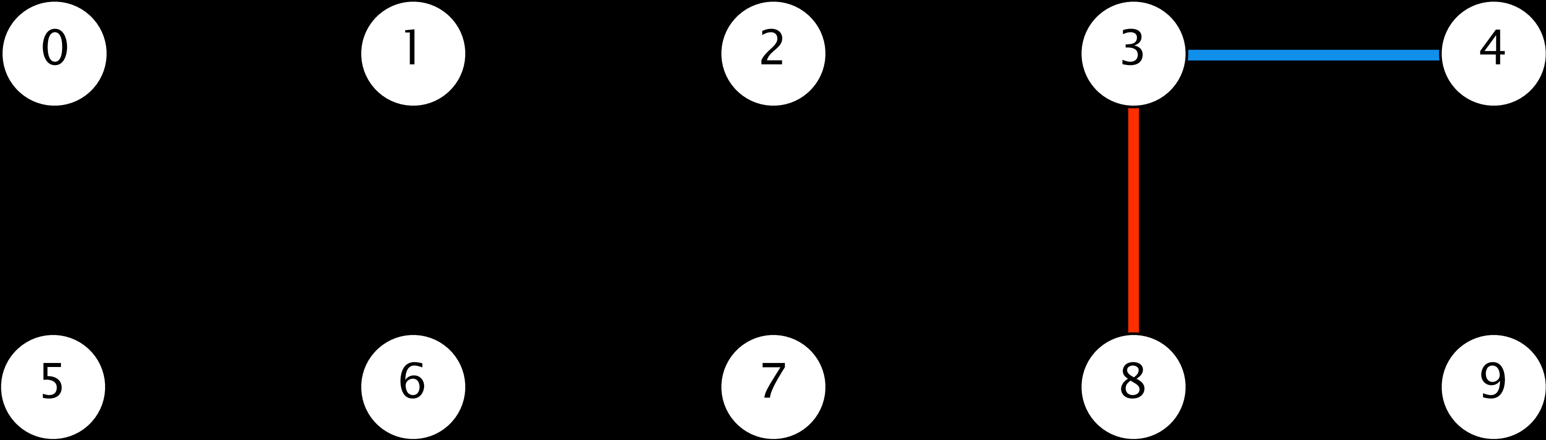


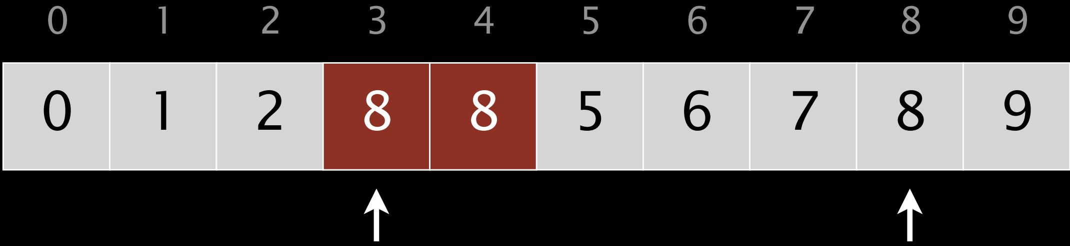


DESIGN CHOICE: CHOOSE TO MAKE THE 2ND PARAMETER THE PARENT

QUICK-FIND DEMO

UNION(3, 8)

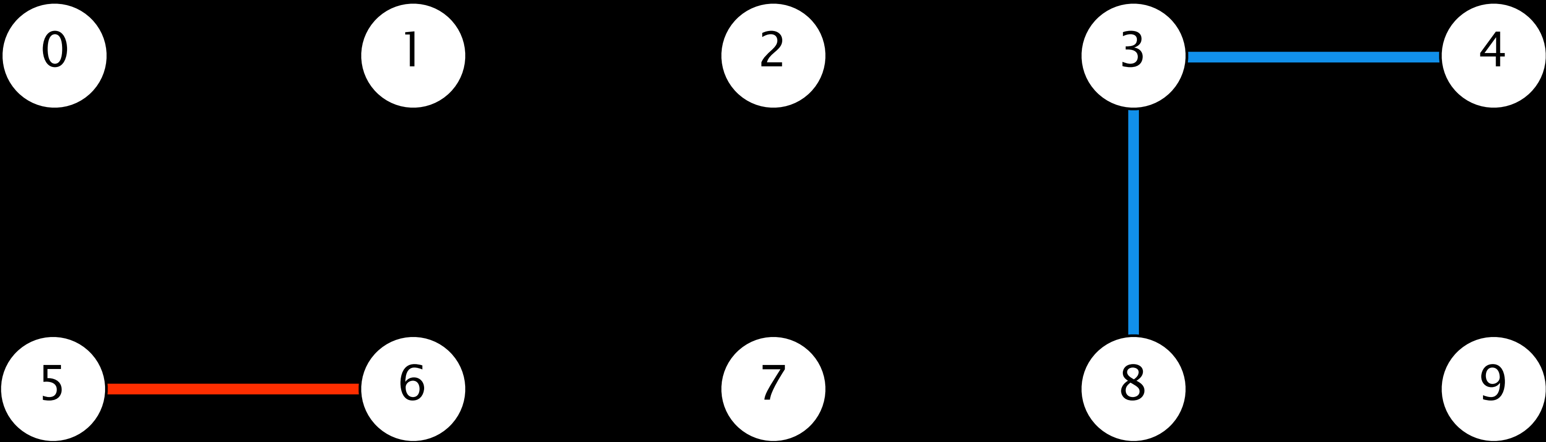




0	1	2	3	4	5	6	7	8	9
0	1	2	8	8	5	6	7	8	9

QUICK-FIND DEMO

UNION(6, 5)



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

↑ ↑

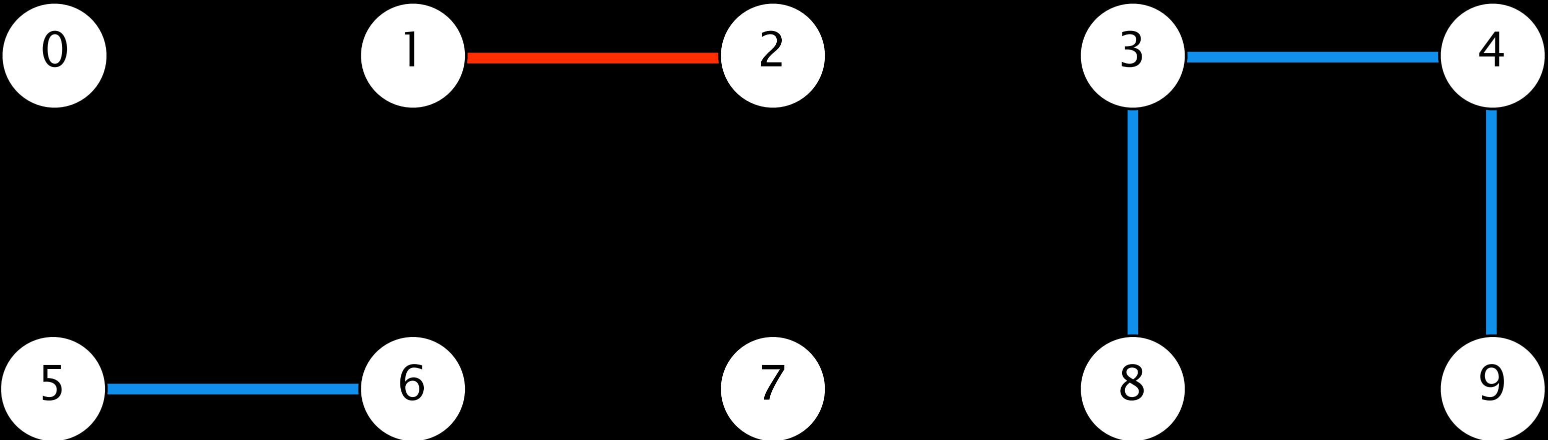
QUICK-FIND DEMO

UNION(9, 4)

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

QUICK-FIND DEMO

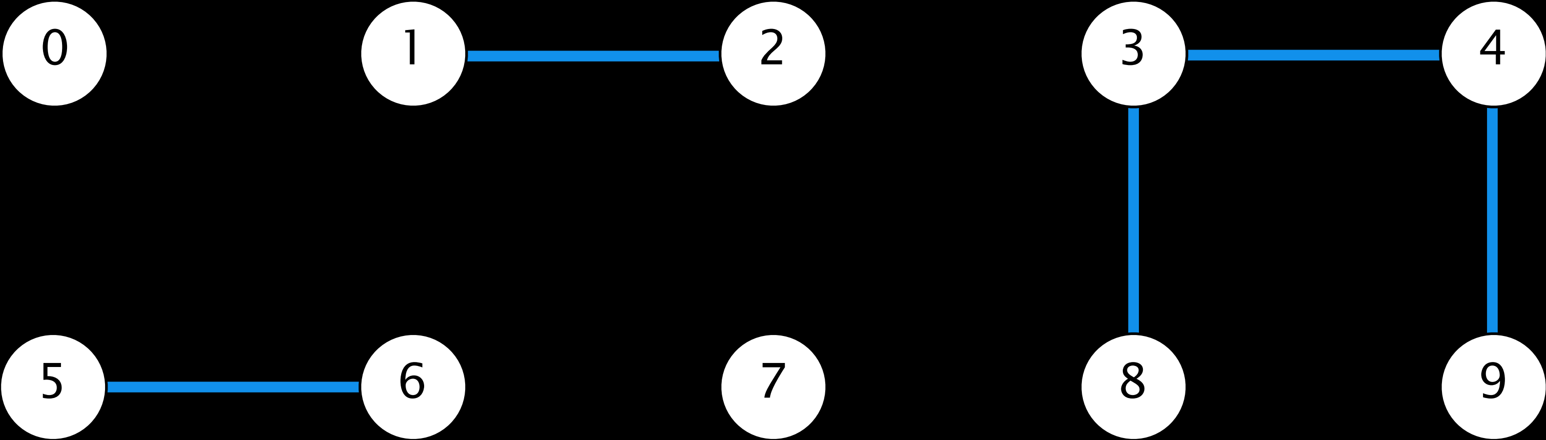
UNION(2, 1)



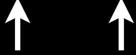
0	1	2	3	4	5	6	7	8	9
0	1	1	8	8	5	5	7	8	8

QUICK-FIND DEMO

CONNECTED(8, 9)



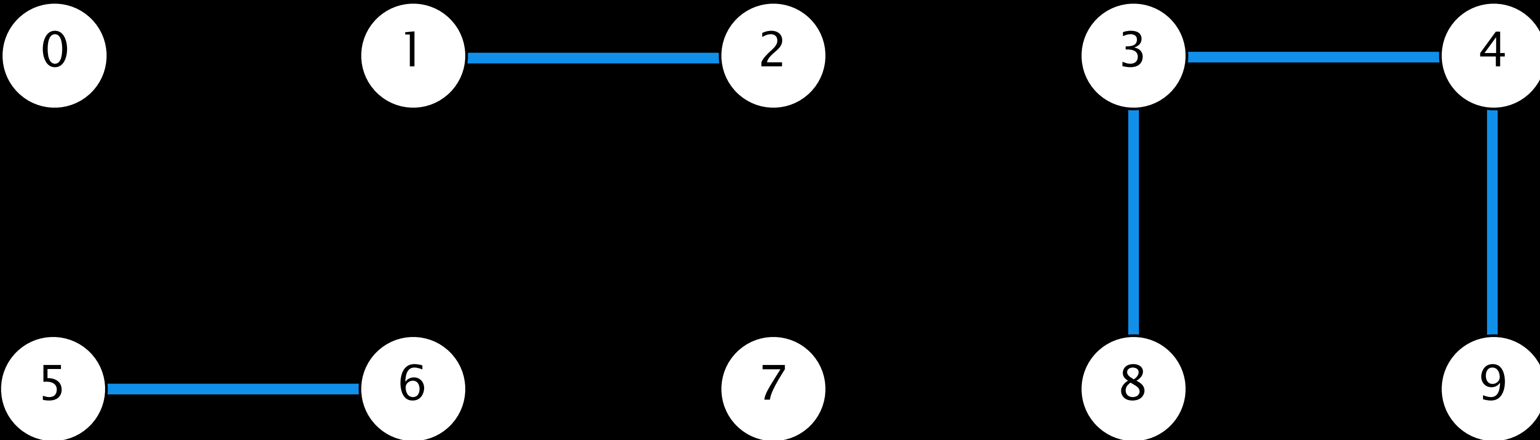
0	1	2	3	4	5	6	7	8	9
0	1	1	8	8	5	5	7	8	8



ALREADY CONNECTED

QUICK-FIND DEMO

CONNECTED(5, 0)



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

NOT CONNECTED

QUICK-FIND DEMO

UNION(5, 0)

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

QUICK-FIND DEMO

UNION(7, 2)

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

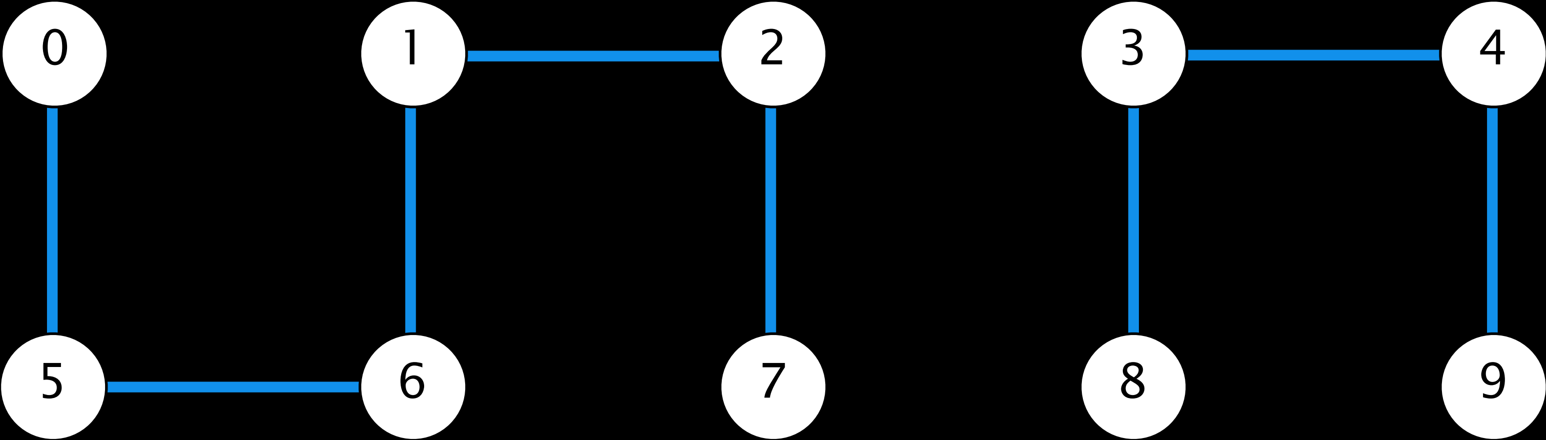
QUICK-FIND DEMO

UNION(6, 1)

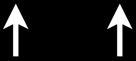
0	1	2	3	4	5	6	7	8	9
1	1	1	8	8	1	1	1	8	8

QUICK-FIND DEMO

CONNECTED(1, 0)

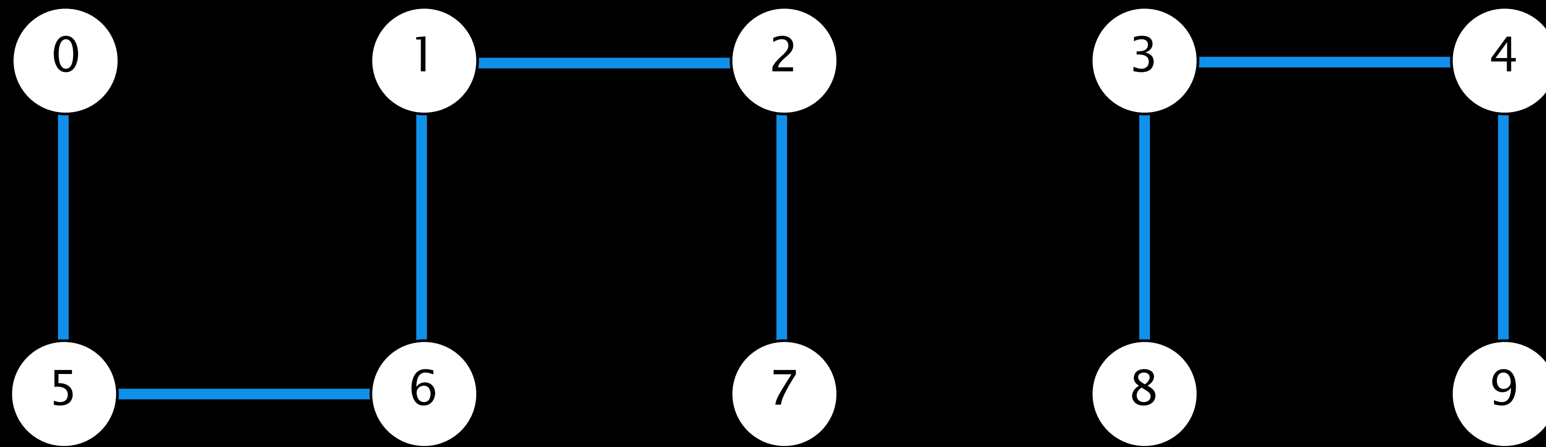


0	1	2	3	4	5	6	7	8	9
1	1	1	8	8	1	1	1	8	8



ALREADY CONNECTED

QUICK-FIND DEMO



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

QUICK-FIND: JAVA IMPLEMENTATION

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
      id[i] = i;
   }

   public boolean find(int p)
   {  return id[p];  }

   public void union(int p, int q)
   {
      int pid = id[p];
      int qid = id[q];
      for (int i = 0; i < id.length; i++)
         if (id[i] == pid) id[i] = qid;
   }
}
```

set id of each object to itself
(N array accesses)

return the id of p
(1 array access)

change all entries with id[p] to id[q]
(at most 2N + 2 array accesses)

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find	connected
quick-find	N	N	1	1

order of growth of number of array accesses

Union is too expensive. It takes N^2 array accesses to process a sequence of N union operations on N objects.

QUADRATIC ALGORITHMS DON'T SCALE

AS N^2 UNION OPERATION IS NO GOOD

Assume that we have a processor that can do 10^9 operations per second

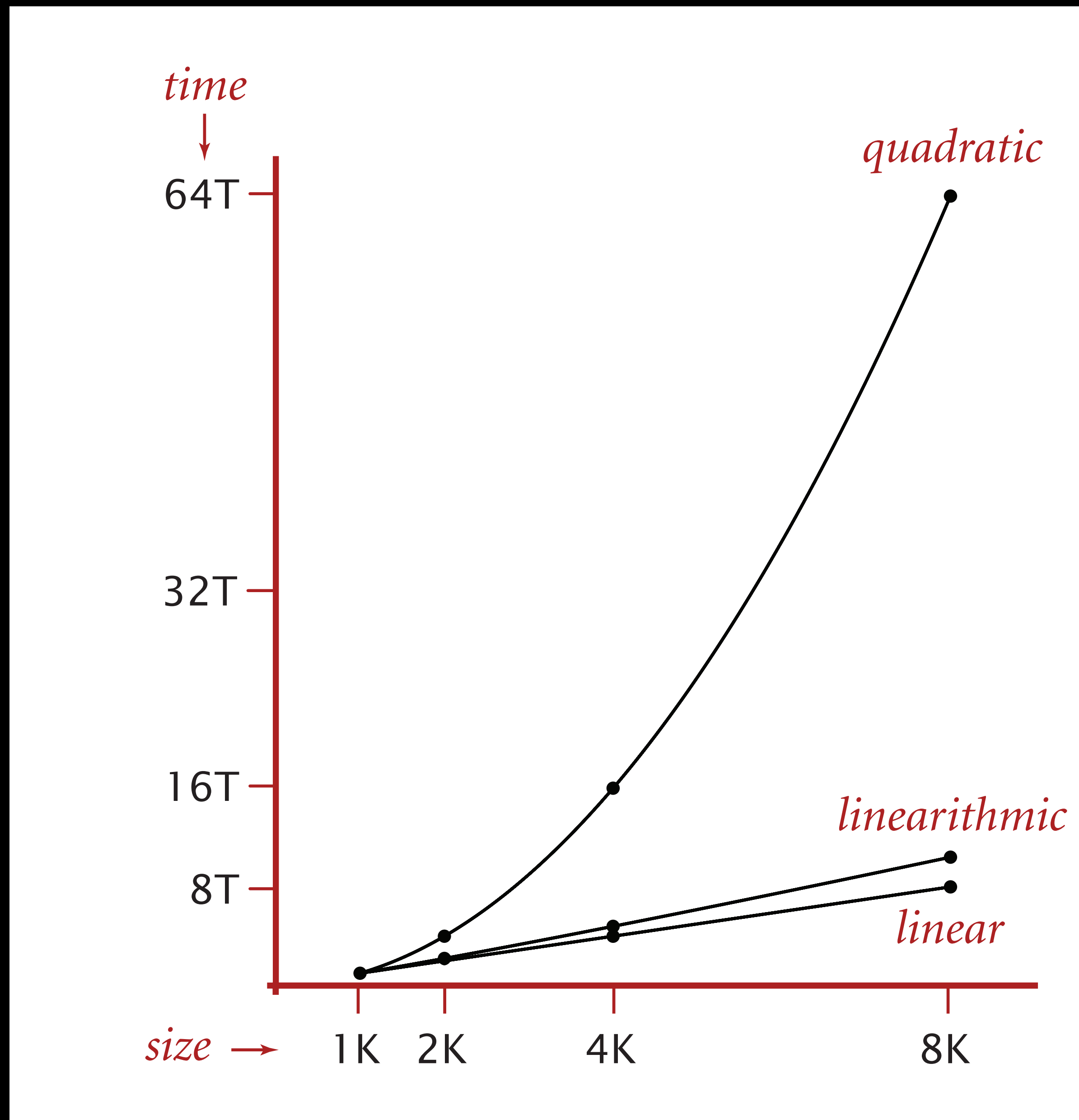
Assume a Dataset with 10,000,000 10^7 records

Assume you could access all records in 10 milli seconds 10^{-2}

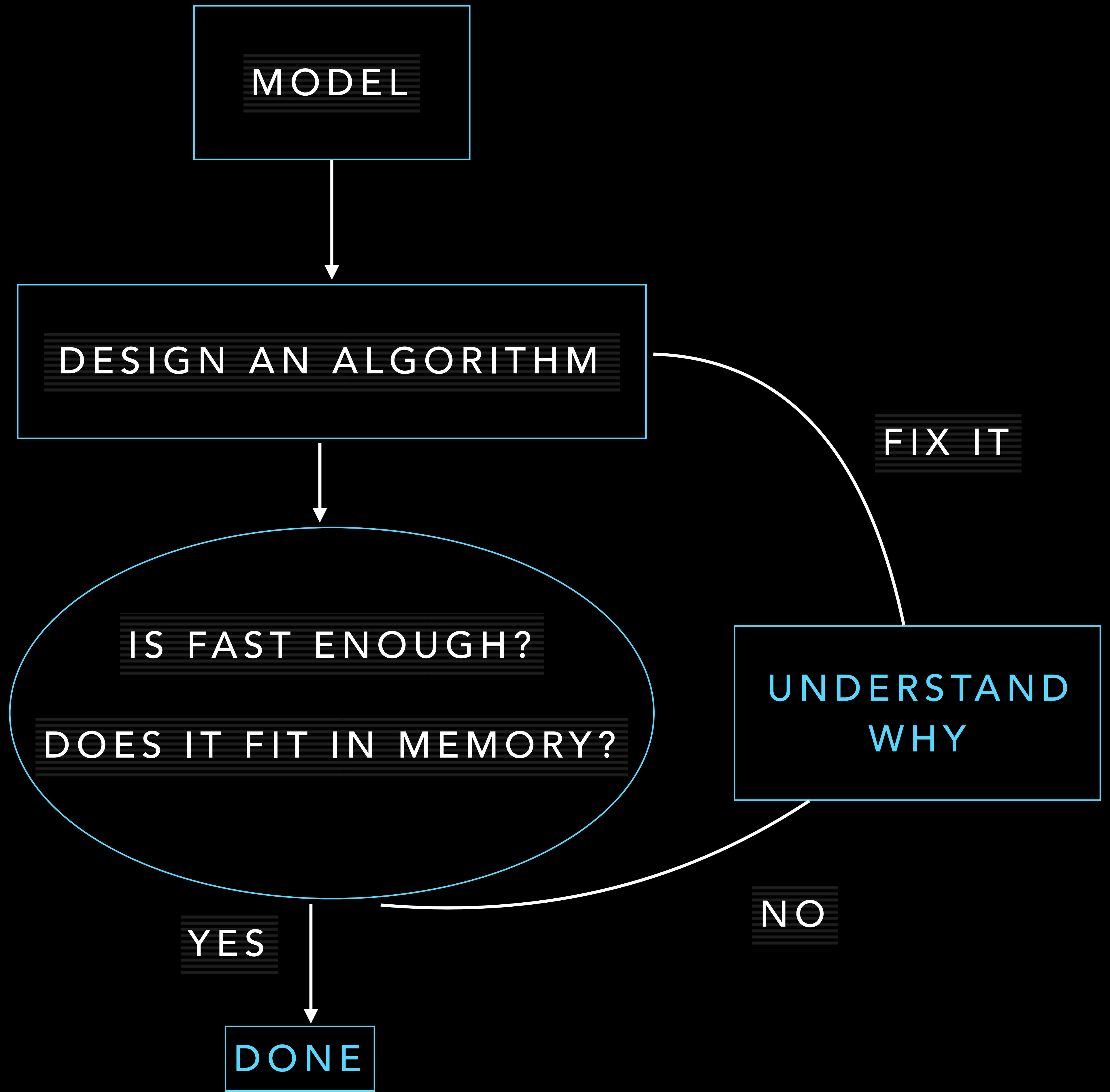
It would take $10^7 \times 10^{-2}$ = (27.7 hours)

Double the program size

It takes 4 times as long?



WELL. HOW DO WE FIX THIS?


MODEL
DESIGN AN ALGORITHM
FIX IT
IS FAST ENOUGH?
DOES IT FIT IN MEMORY?
UNDERSTAND
WHY
NO
YES
DONE

UNION OPERATION IS SLOW

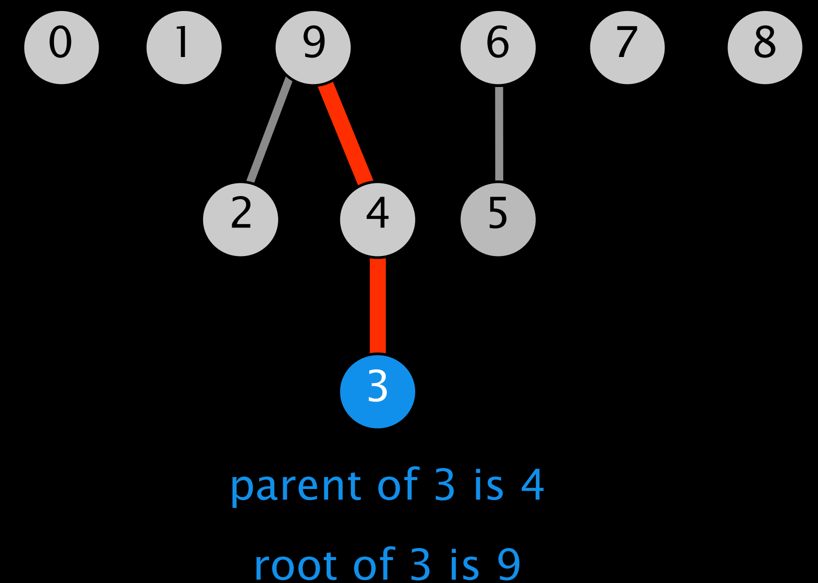
HOW DO WE SPEED UP THE UNION OPERATION?

WHAT IF WE JUST UPDATE THE ONLY ONE ENTRY

Data structure.

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

  Root of `i` is `id[id[id[...id[i]...]]]`.



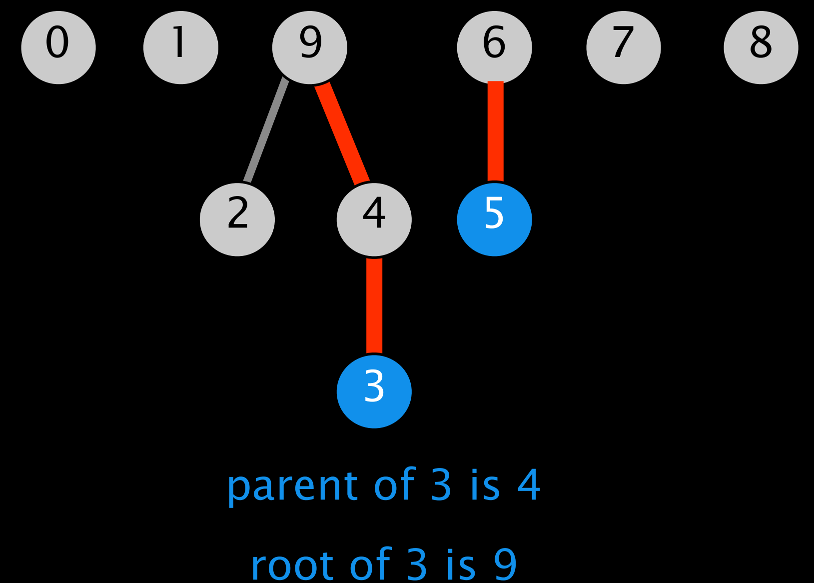
0	1	2	3	4	5	6	7	8	9
0	1	9	4	9	6	6	7	8	9

WHAT IF WE JUST UPDATE THE ONLY ONE ENTRY

Data structure.

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

  Root of `i` is `id[id[id[...id[i]...]]]`.



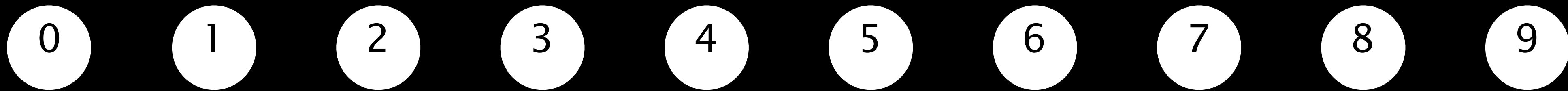
0	1	2	3	4	5	6	7	8	9
0	1	9	4	9	6	6	7	8	9

ARE 3 AND 5 CONNECTED

Find. What is the root of `p`

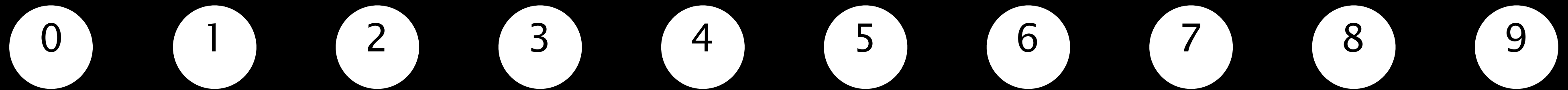
Connected. Do `p` and `q` have the same root?

LET’S STEP THROUGH AN EXAMPLE



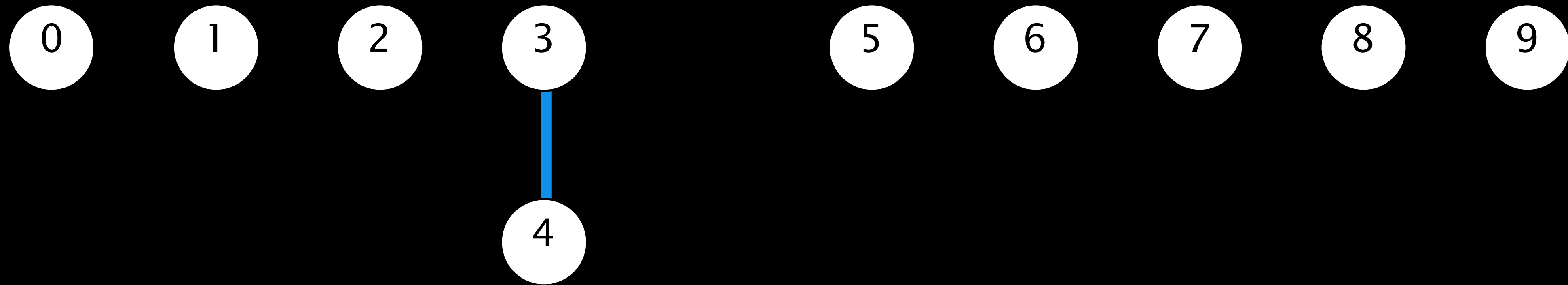
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

UNION(4, 3)

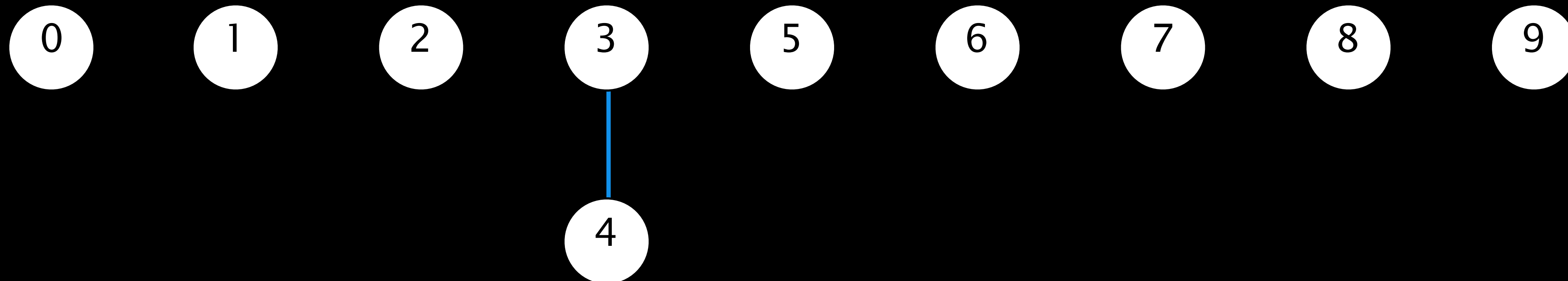


0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

UNION(4, 3)

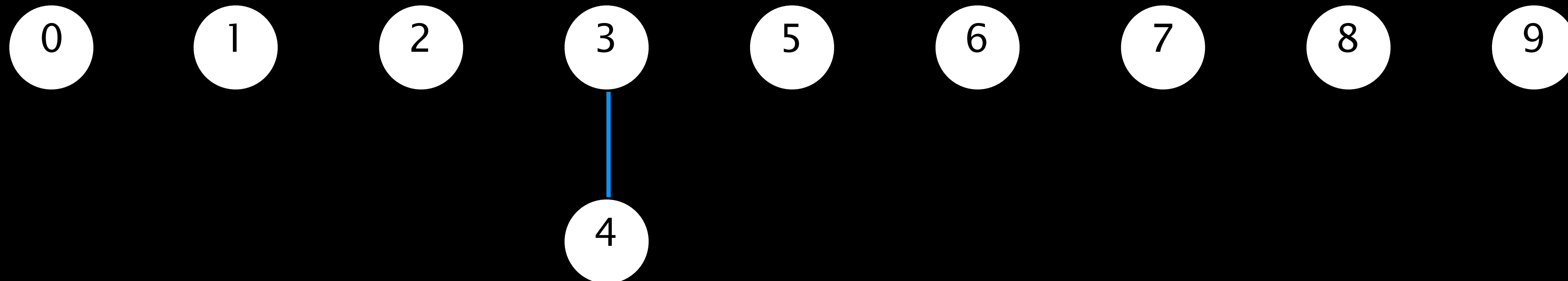


0	1	2	3	4	5	6	7	8	9
0	1	2	3	3	5	6	7	8	9



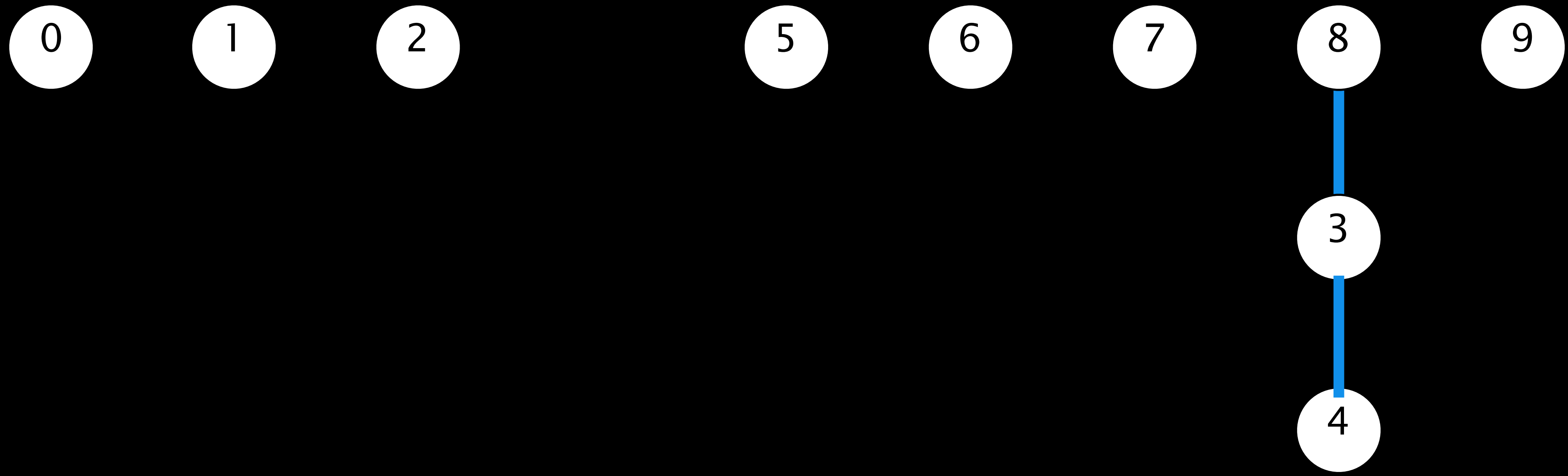
0	1	2	3	4	5	6	7	8	9
0	1	2	3	3	5	6	7	8	9

UNION(3, 8)

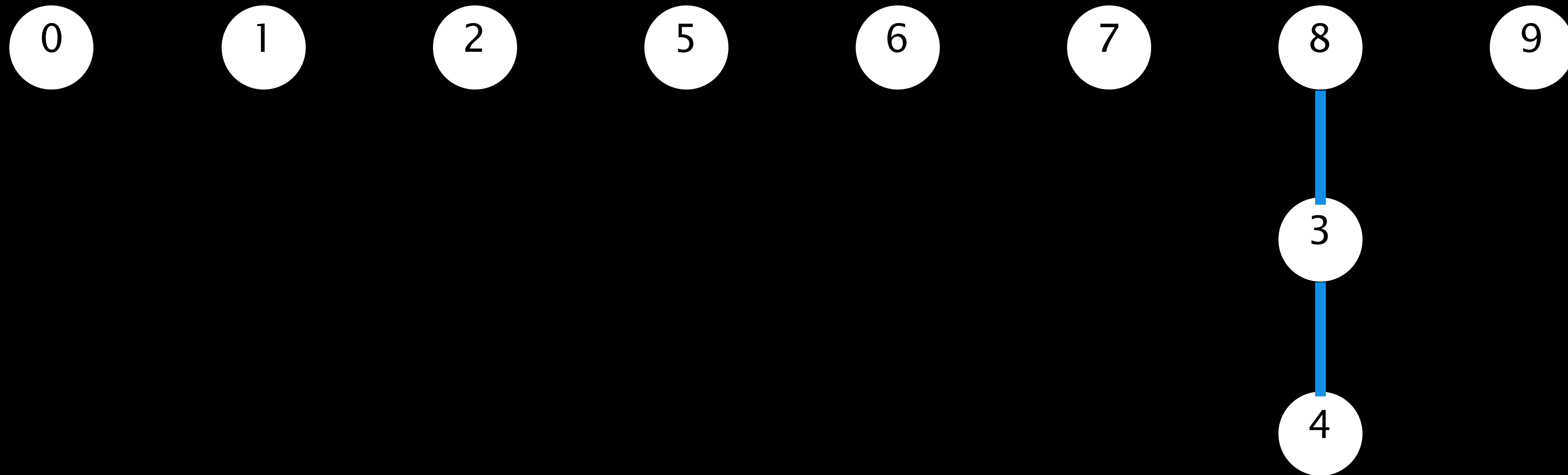


0	1	2	3	4	5	6	7	8	9
0	1	2	3	3	5	6	7	8	9

UNION(3, 8)

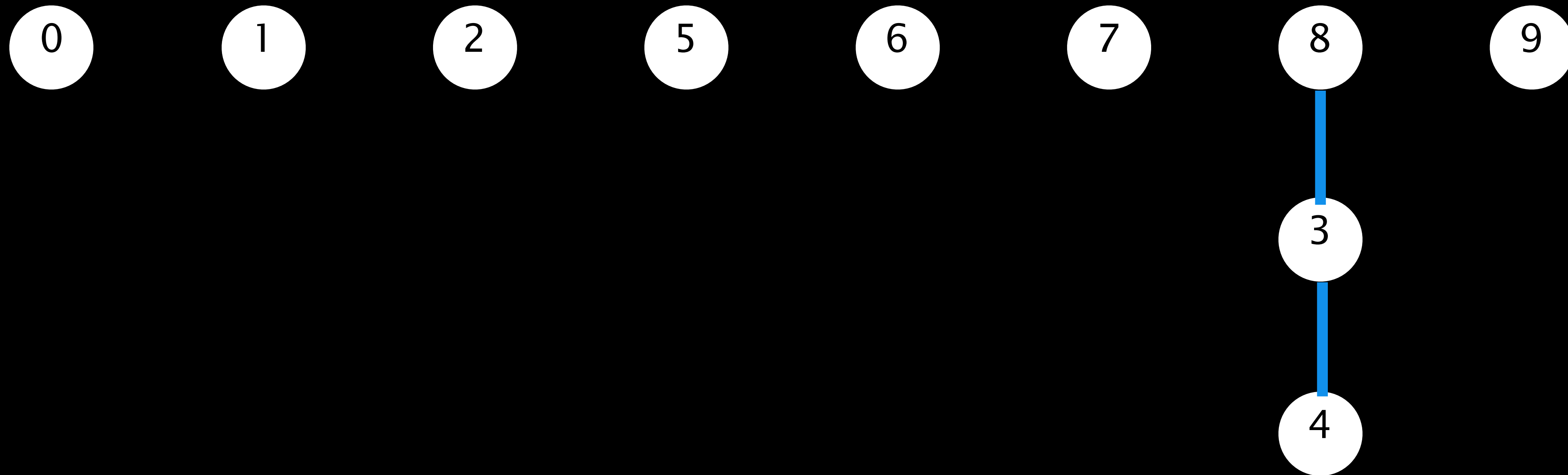


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9



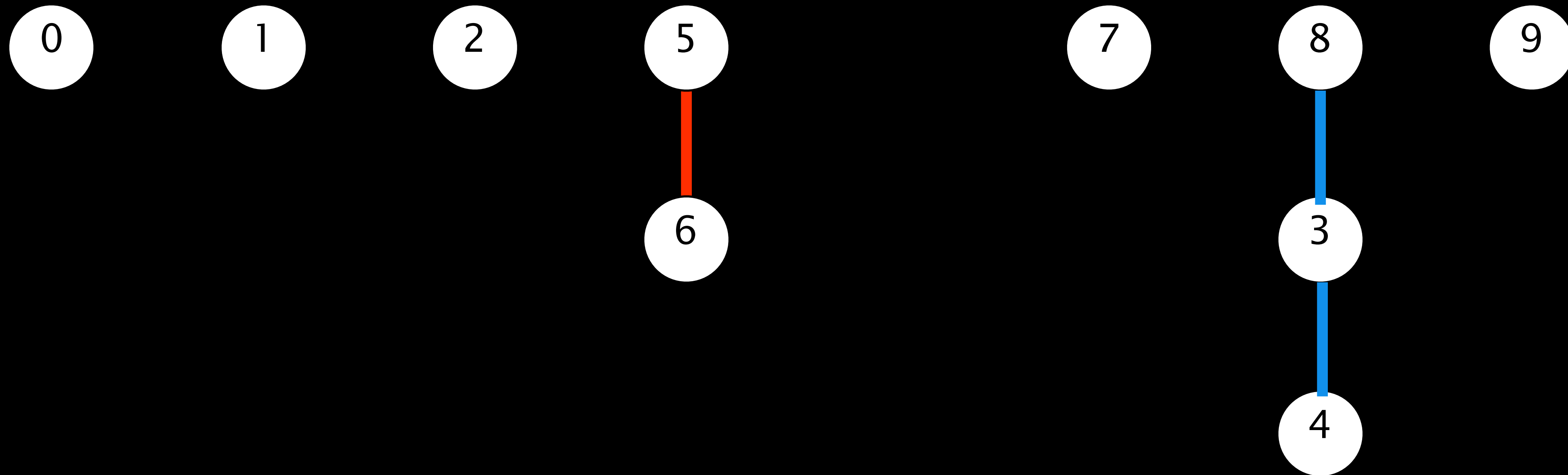
0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9

UNION(6, 5)

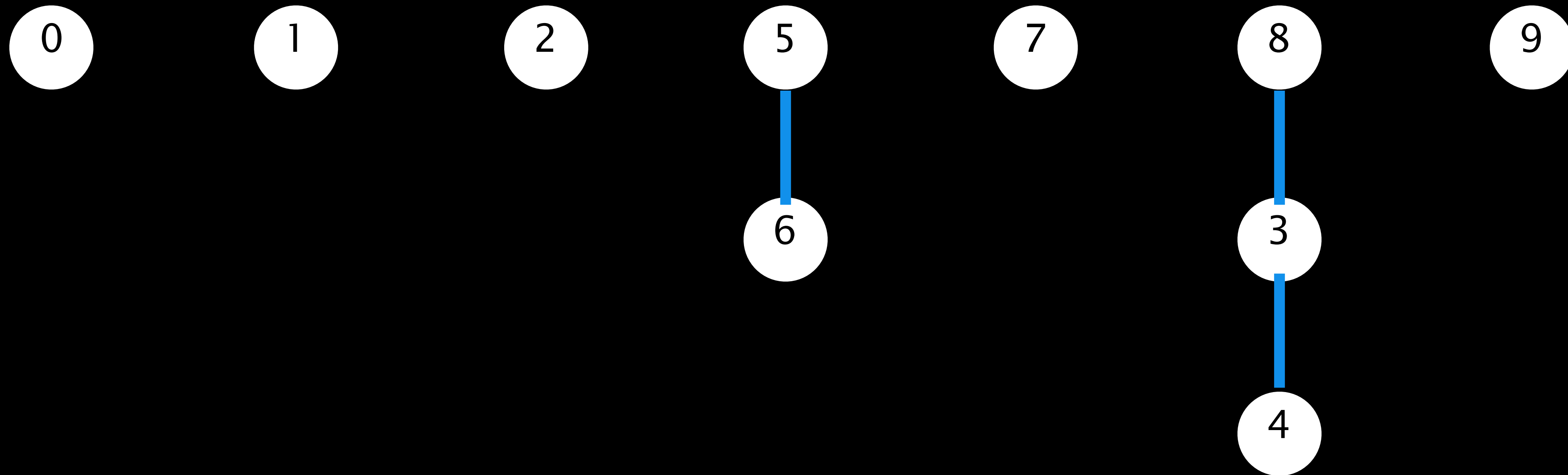


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	6	7	8	9

UNION(6, 5)

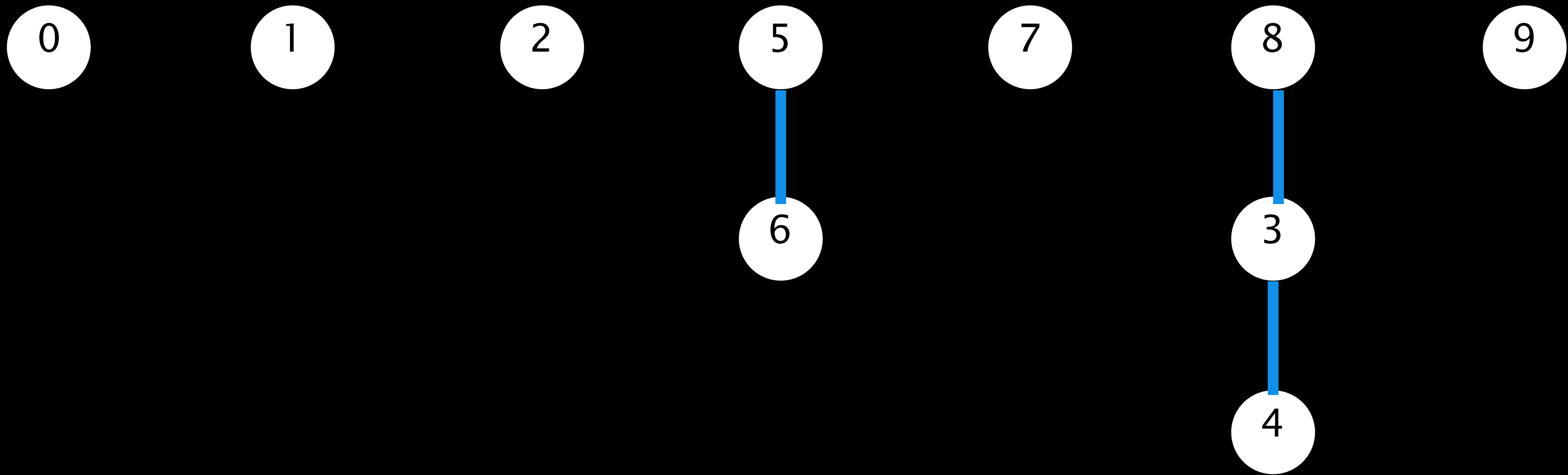


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	9



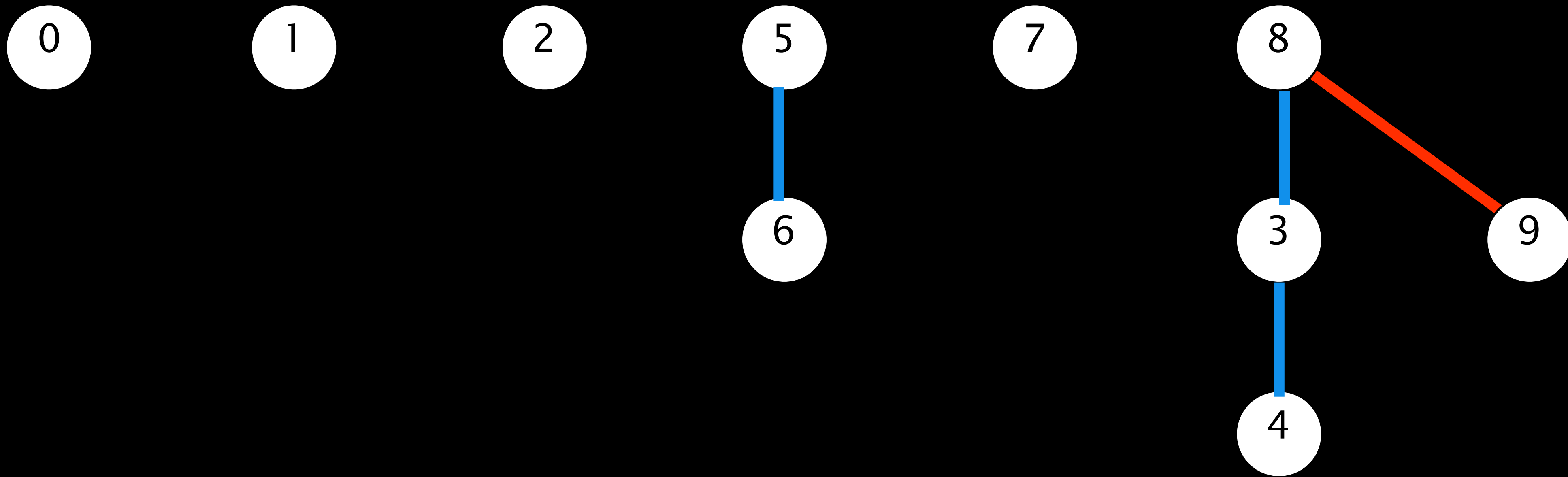
0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	9

UNION(9, 4)

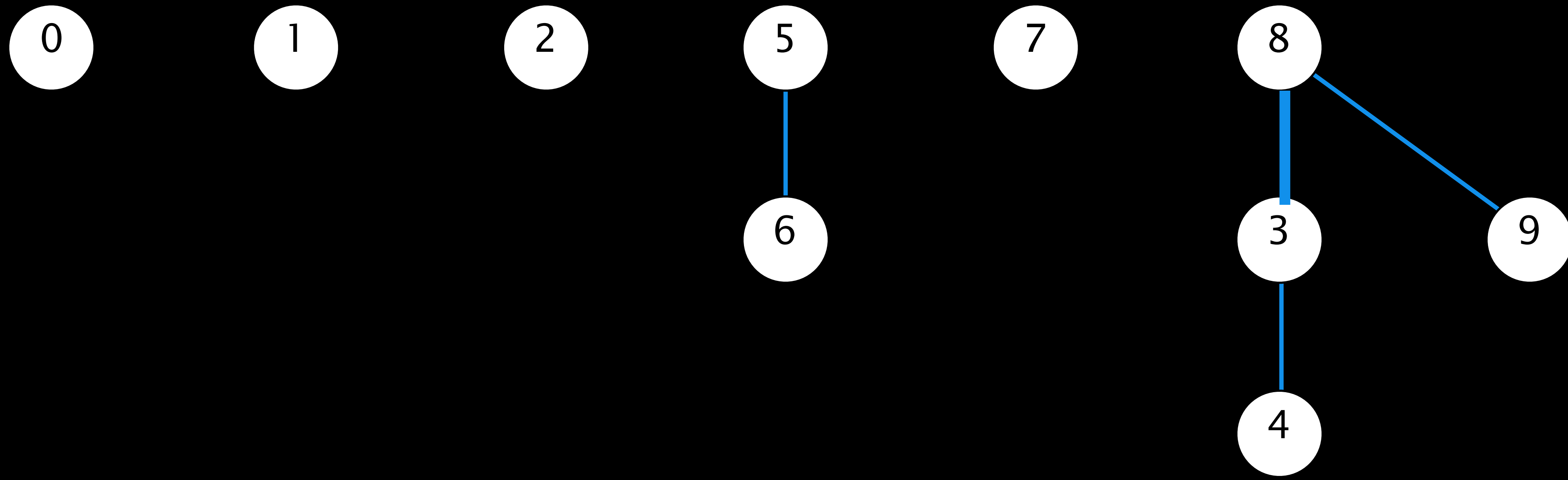


0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	9

UNION(9, 4)



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

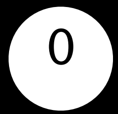


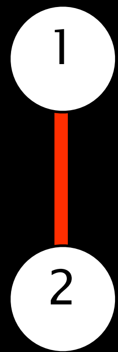
0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	8

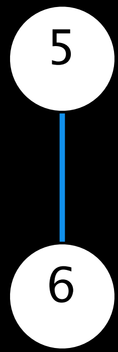
UNION(2, 1)

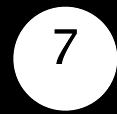
0	1	2	3	4	5	6	7	8	9
0	1	2	8	3	5	5	7	8	8

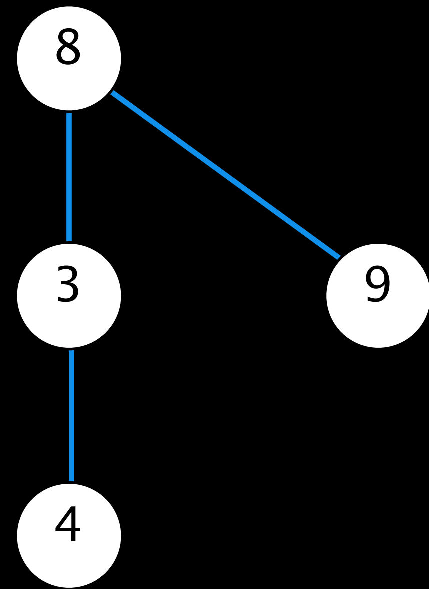
UNION(2, 1)



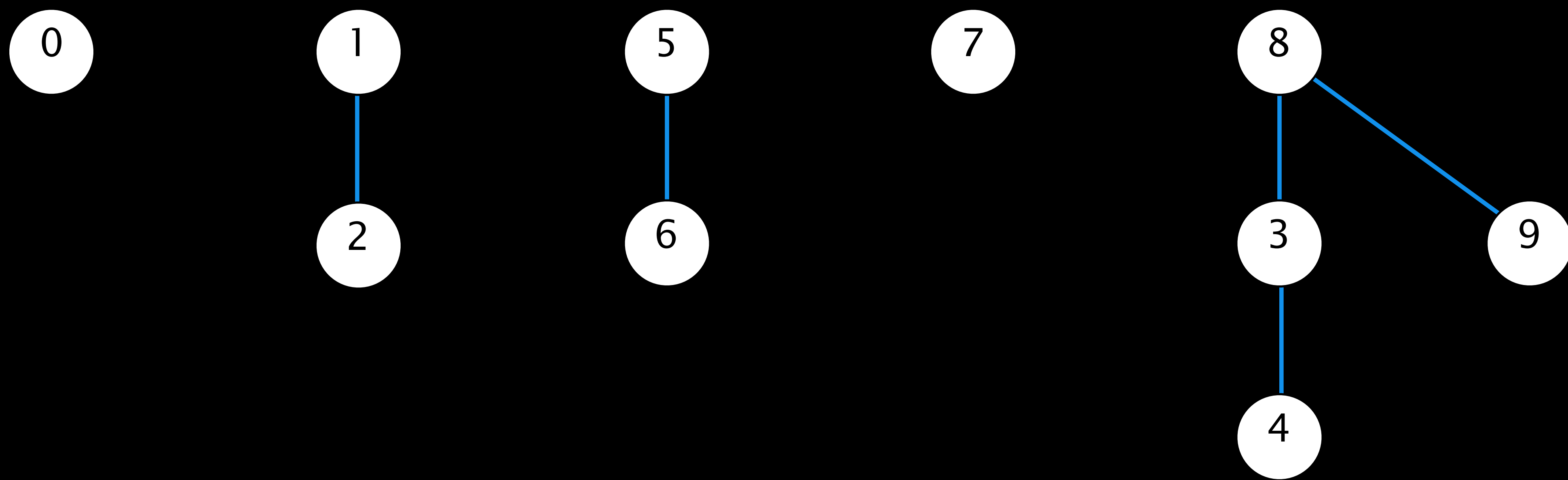






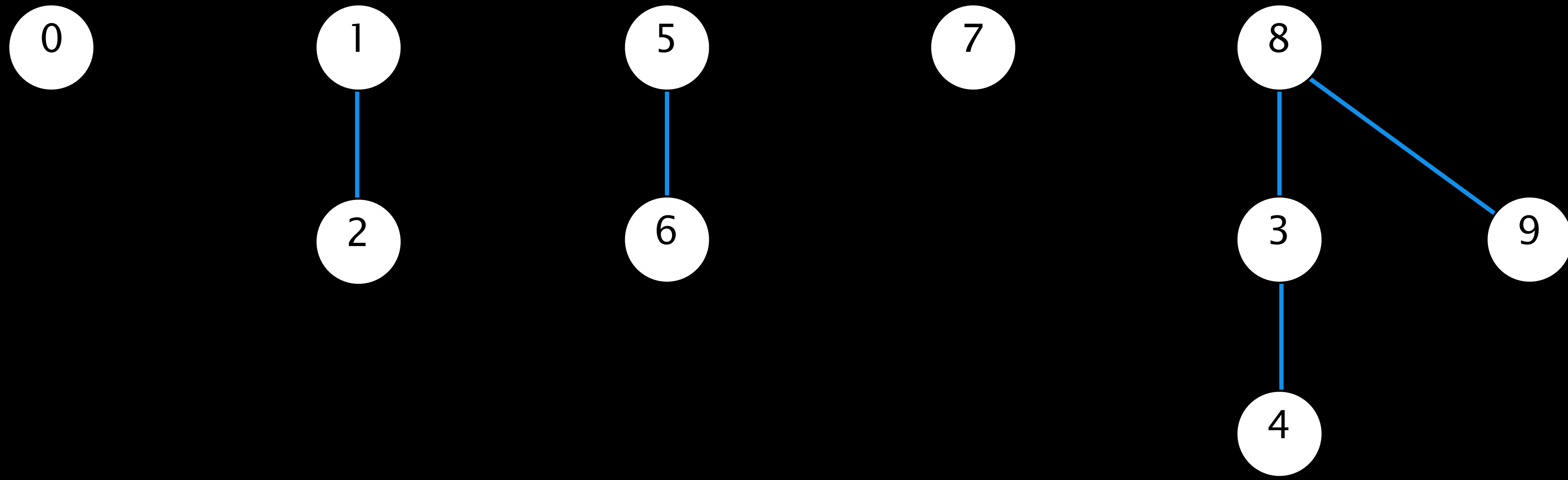


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8



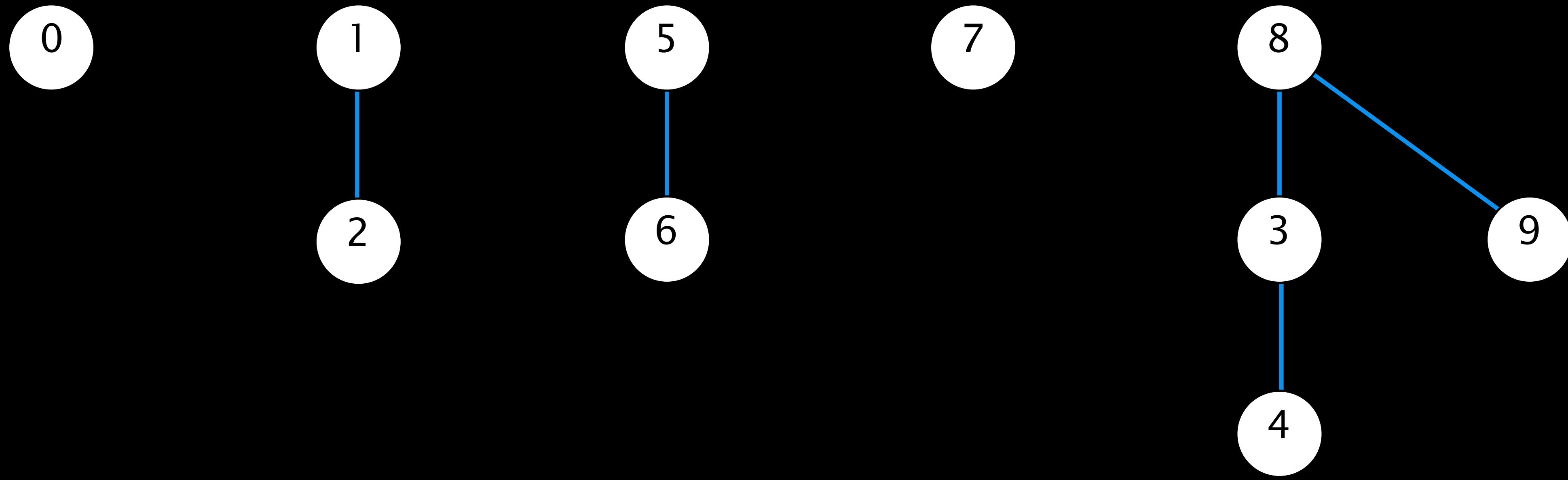
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

CONNECTED(8, 9) ✔



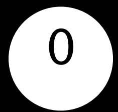
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

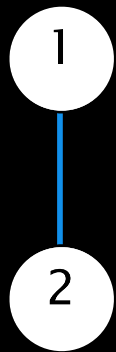
CONNECTED(5, 4) ✗

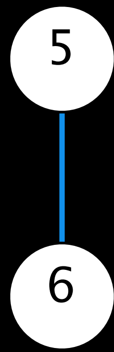


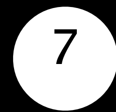
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

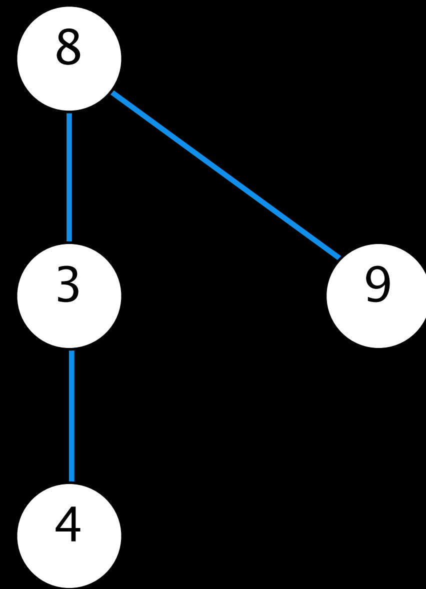
UNION(5, 0)





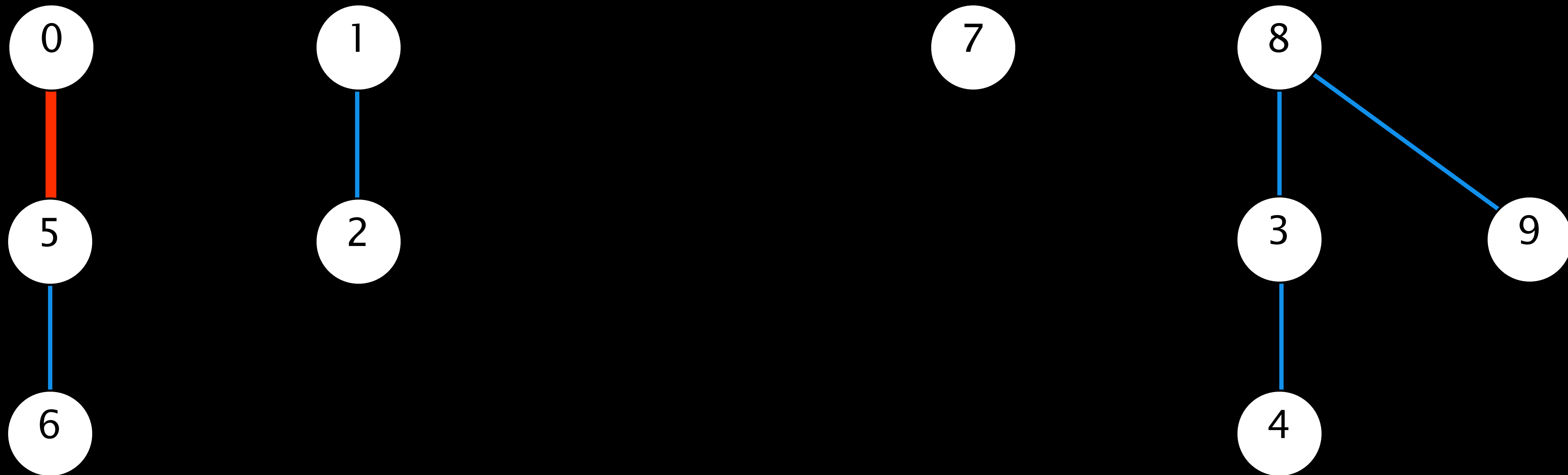




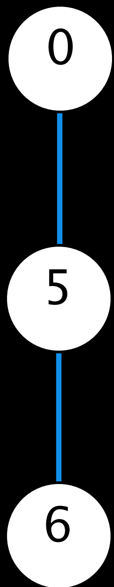


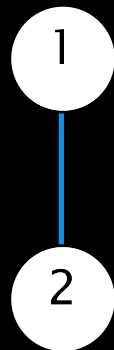
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	5	5	7	8	8

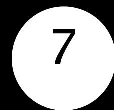
UNION(5, 0)

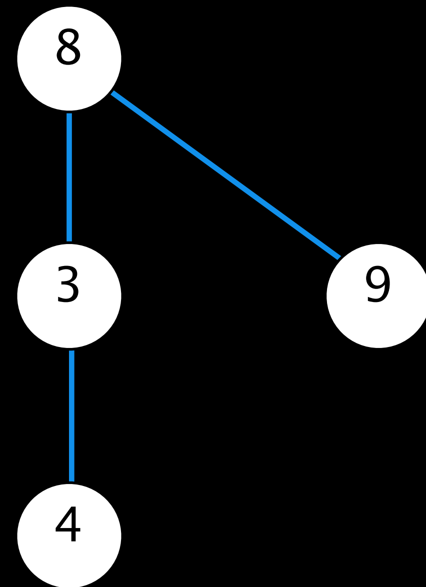


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8



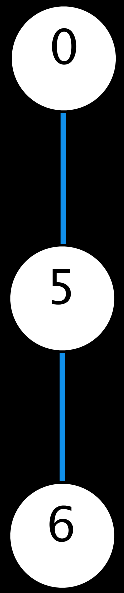


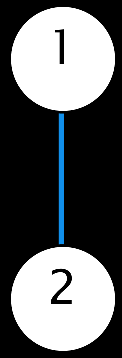


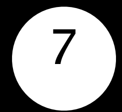


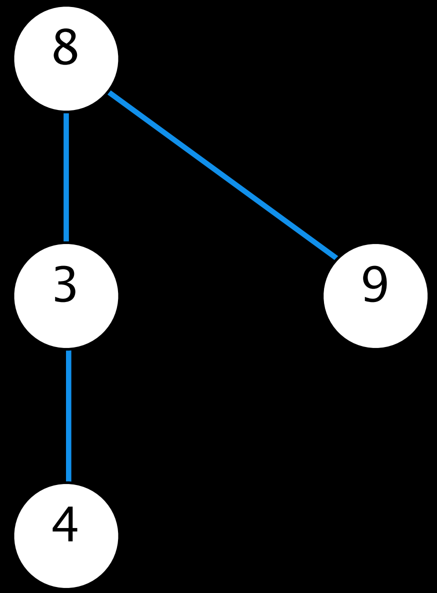
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

UNION(7, 2)



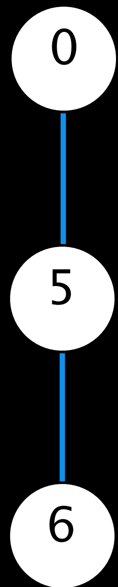


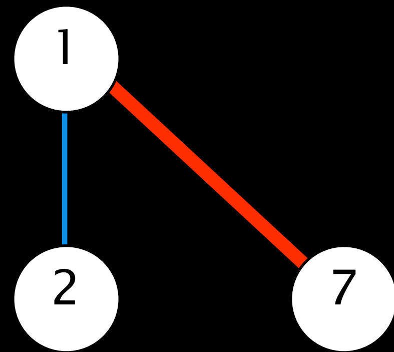


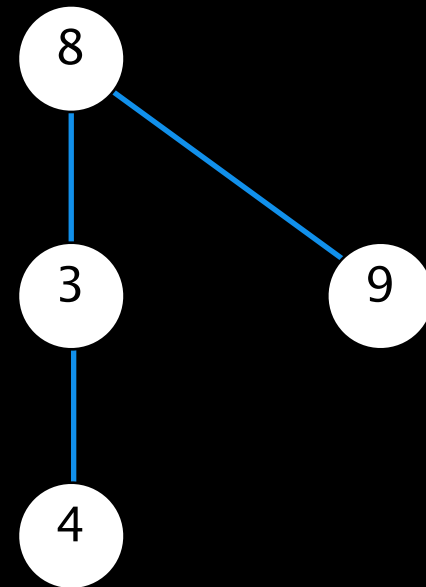


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	7	8	8

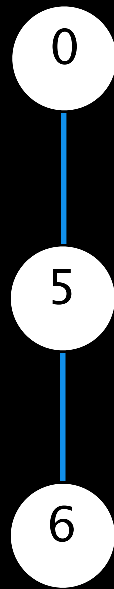
UNION(7, 2)

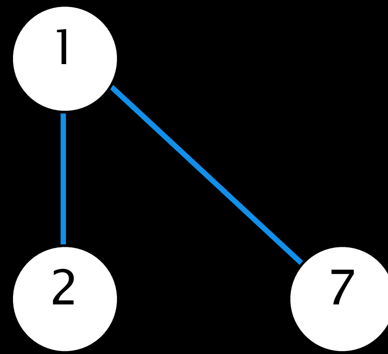


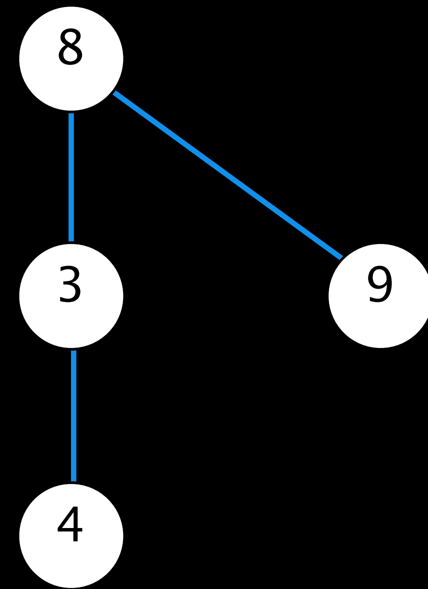




0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	1	8	8

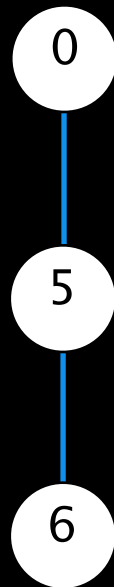


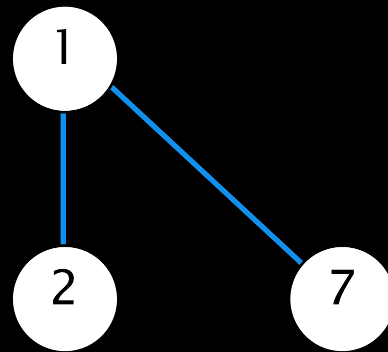


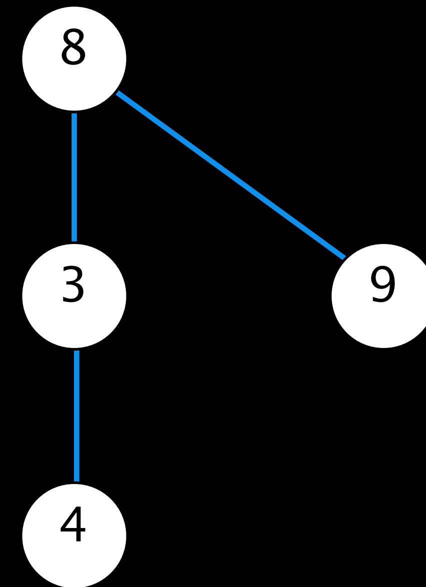


0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	1	8	8

UNION(6, 1)

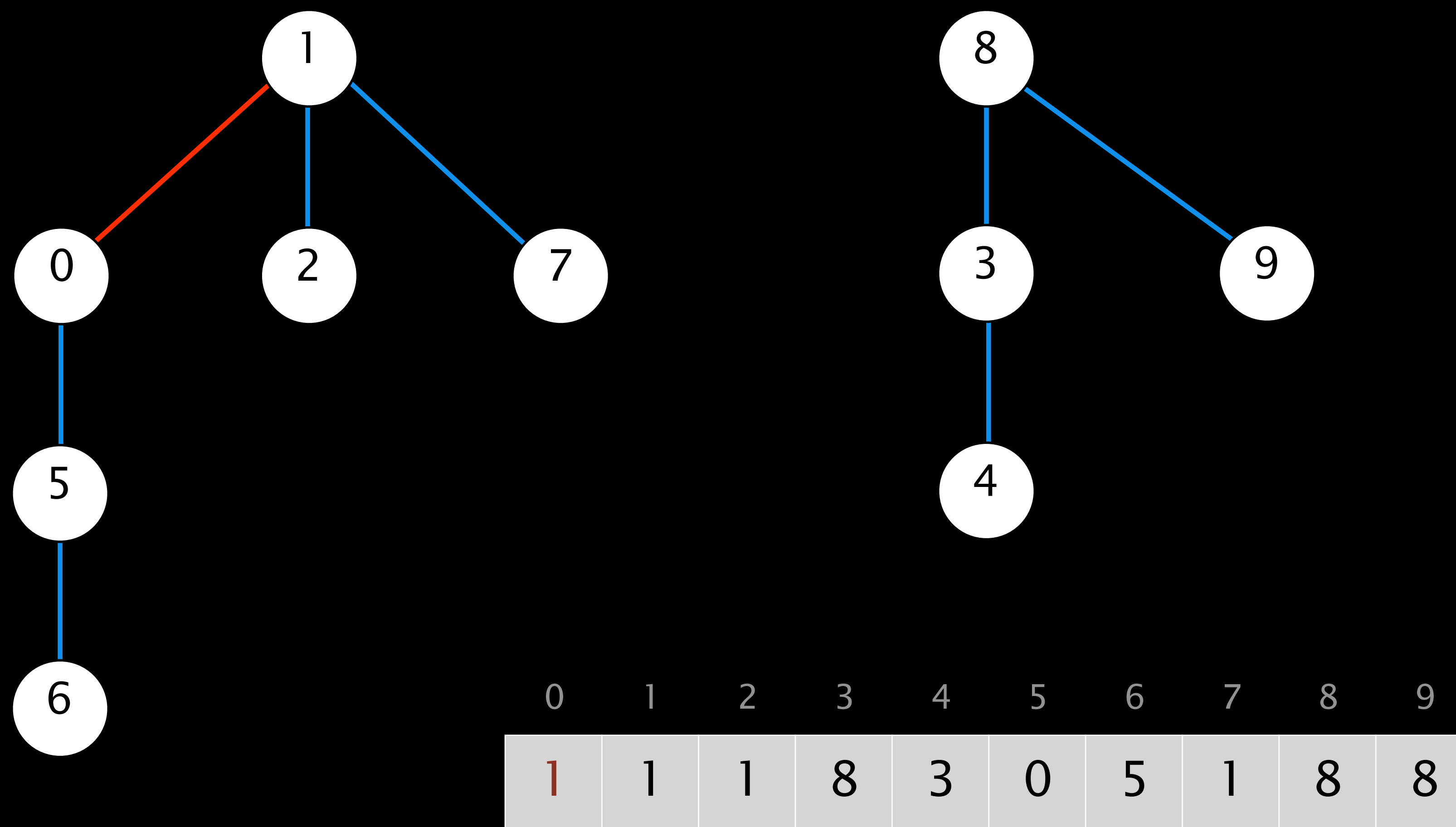


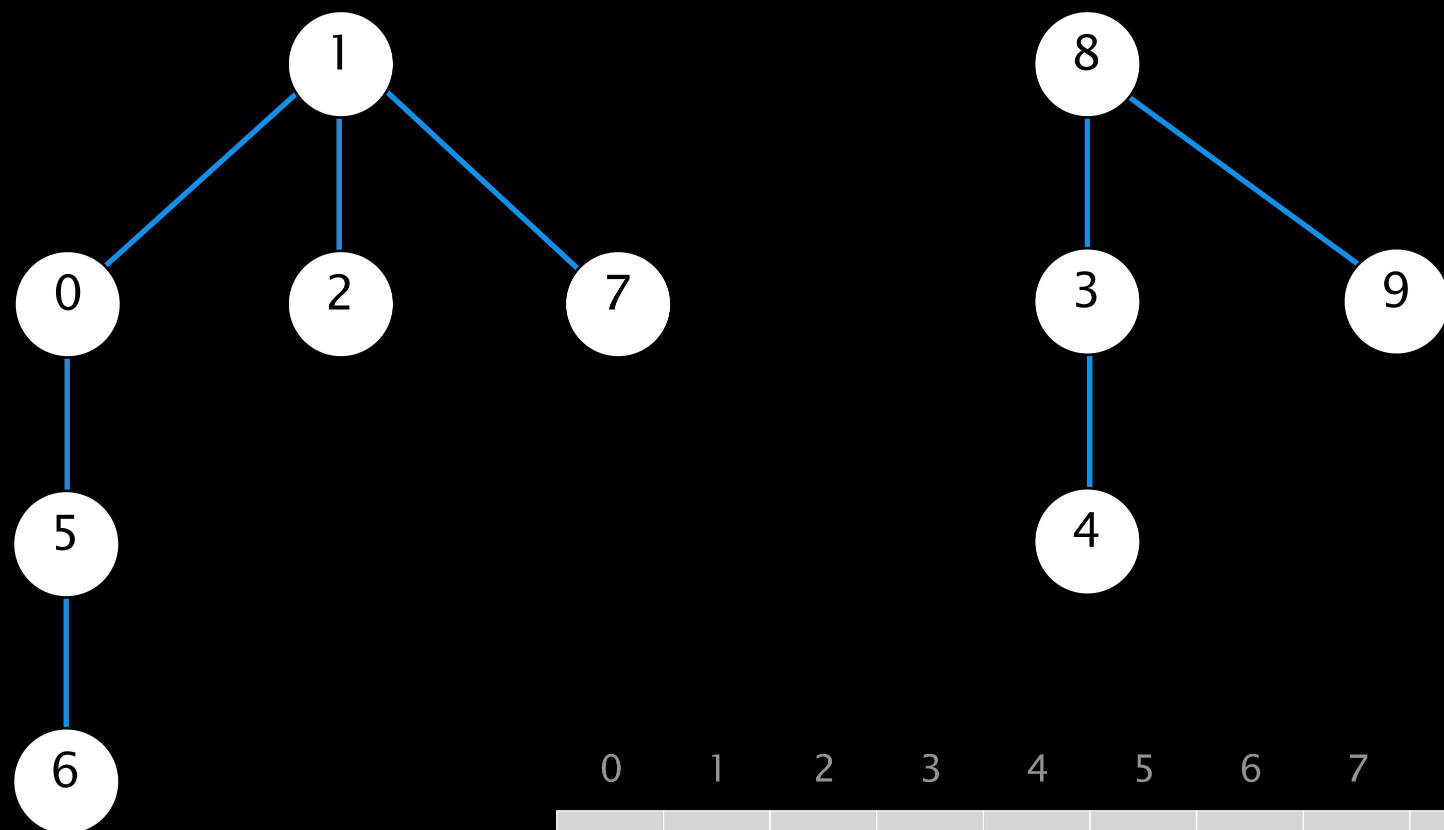




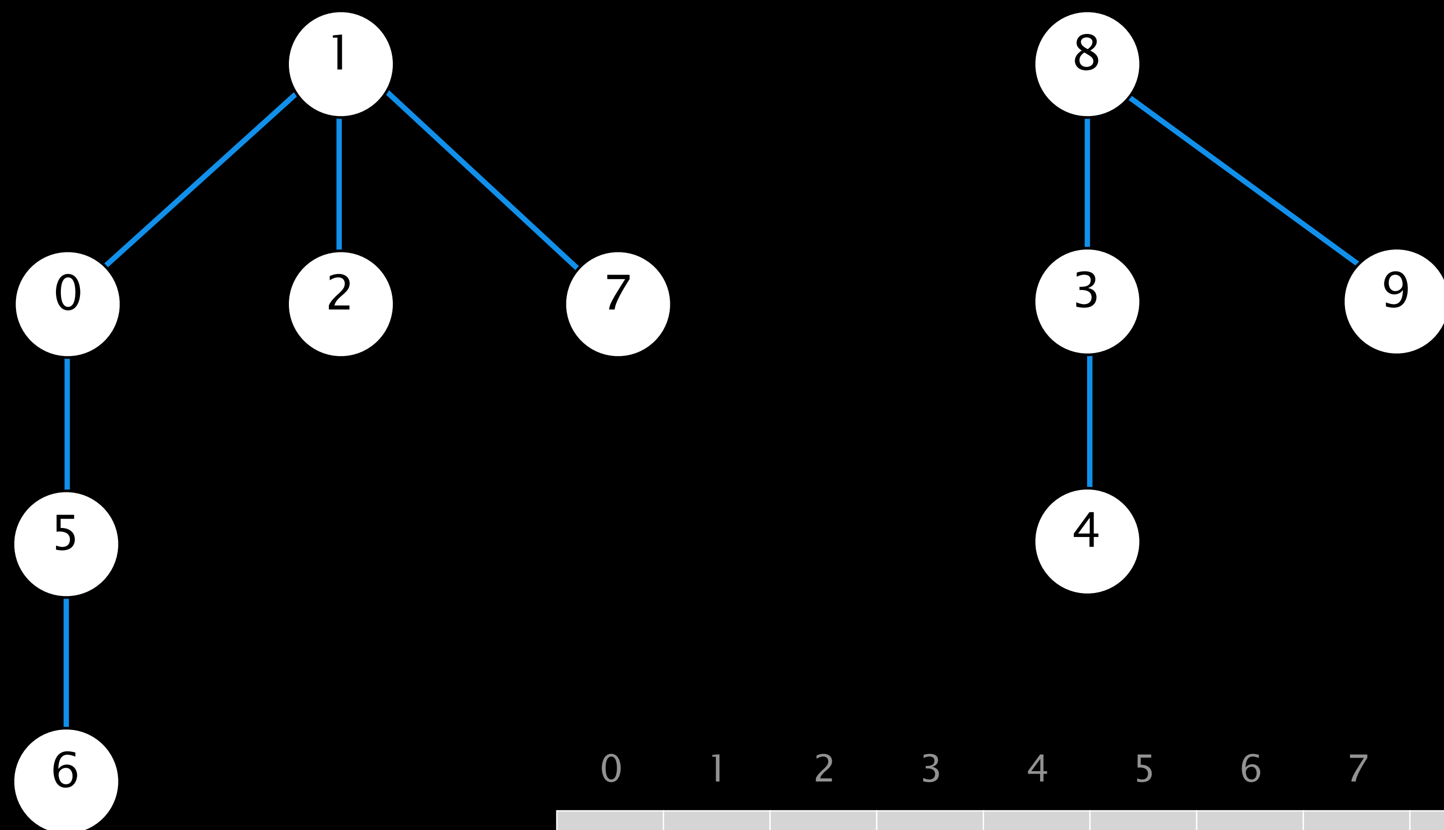
0	1	2	3	4	5	6	7	8	9
0	1	1	8	3	0	5	1	8	8

UNION(6, 1)

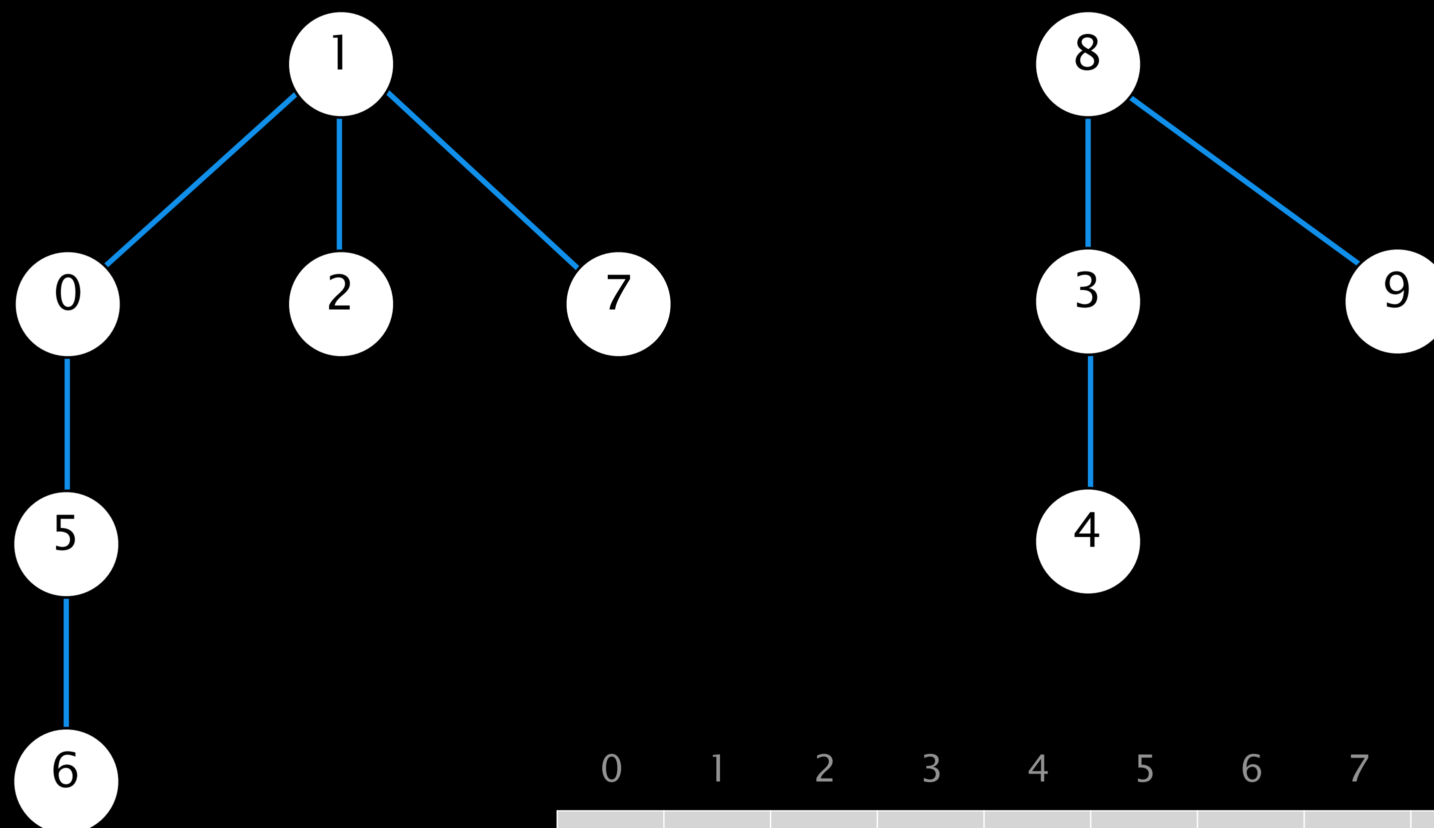



1
0
2
7
5
6
8
3
9
4
0 1 2 3 4 5 6 7 8 9
1 1 1 8 3 0 5 1 8 8

CONNECTED(1, 0)

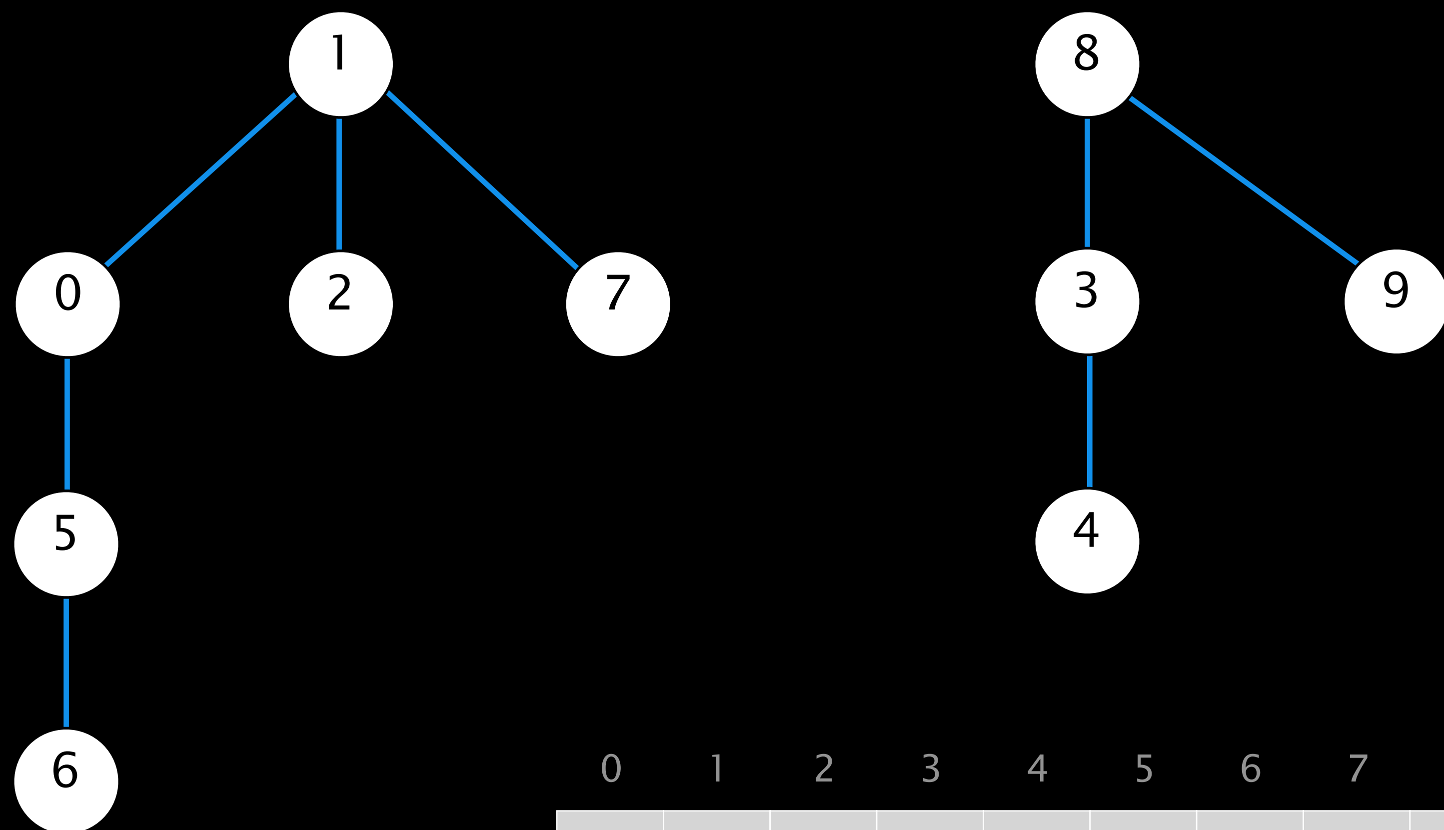


CONNECTED(6, 7)

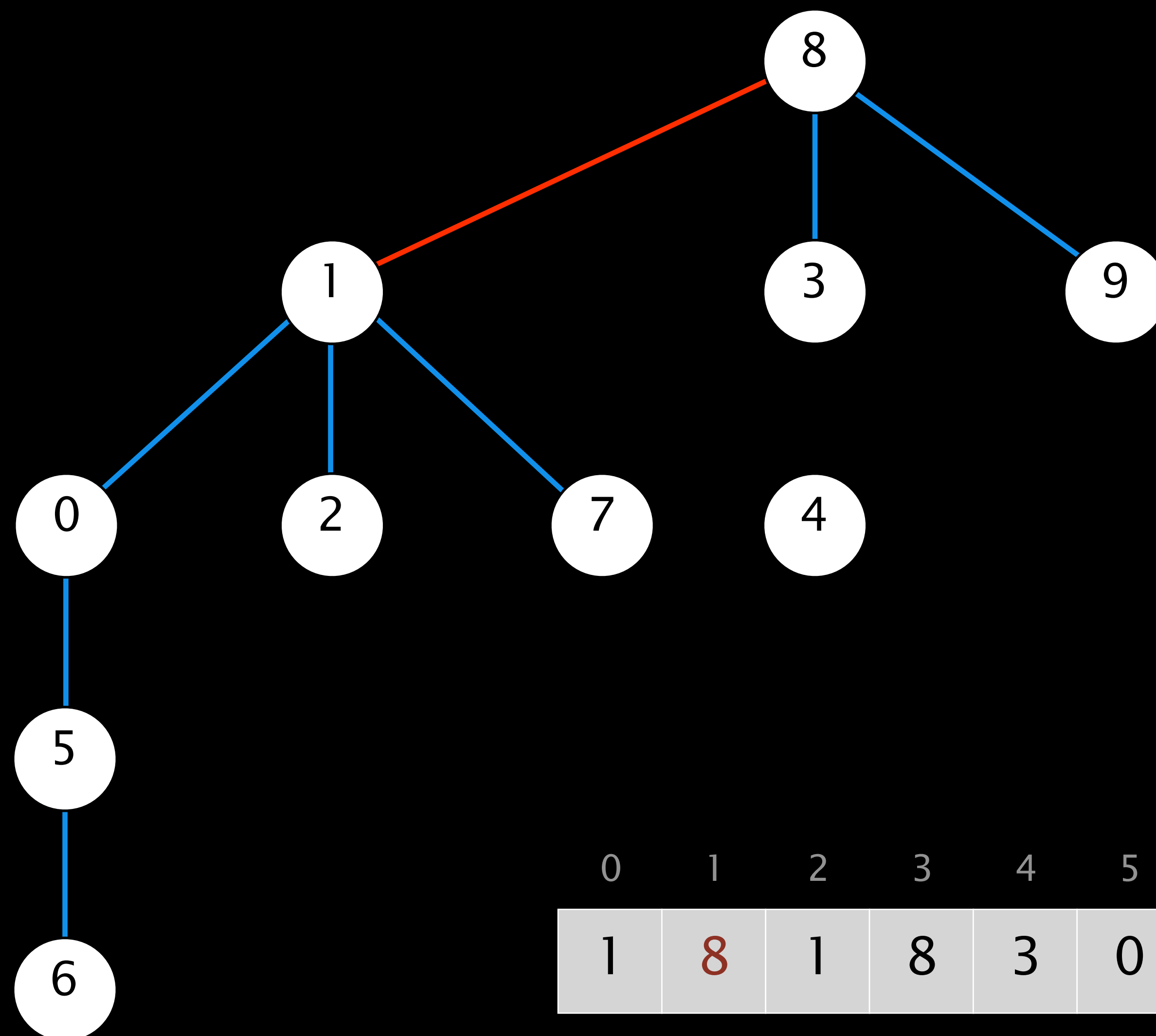


0	1	2	3	4	5	6	7	8	9
1	1	1	8	3	0	5	1	8	8

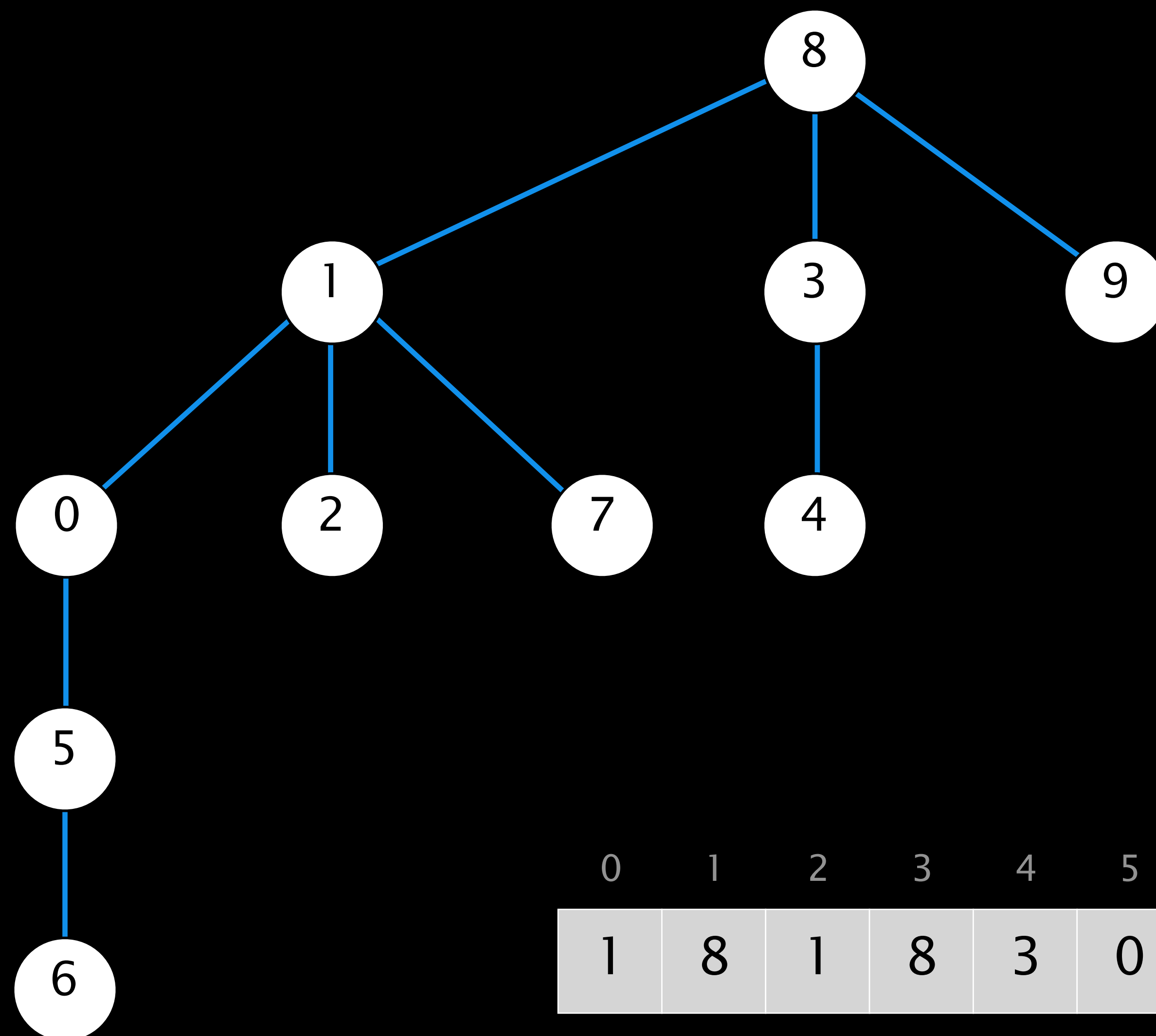
UNION(7, 3)



UNION(7, 3)



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


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

QUICK-UNION: JAVA IMPLEMENTATION

```
public class QuickUnionUF
{
   private int[] id;

   public QuickUnionUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++) id[i] = i;
   }

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

   public boolean connected(int p, int q)
   {
      return find(p) == find(q);
   }

   public void union(int p, int q)
   {
      int i = find(p);
      int j = find(q);
      id[i] = j;
   }
}
```

set id of each object to itself
(N array accesses)

chase parent pointers until reach root
(depth of i array accesses)

do p and q have the same root?
(depth of p and q array accesses)

change root of p to point to root of q
(depth of p and q array accesses)

COULD WE THINK OF CASE WHERE THIS WOULD GO REALLY BADLY

QUICK-UNION IS ALSO TOO

Cost model. Number of array accesses (for read or write).

algorithm	initialize	union	find	connected
quick-find	N	N	1	1
quick-union	N	N †	N	N

← worst case

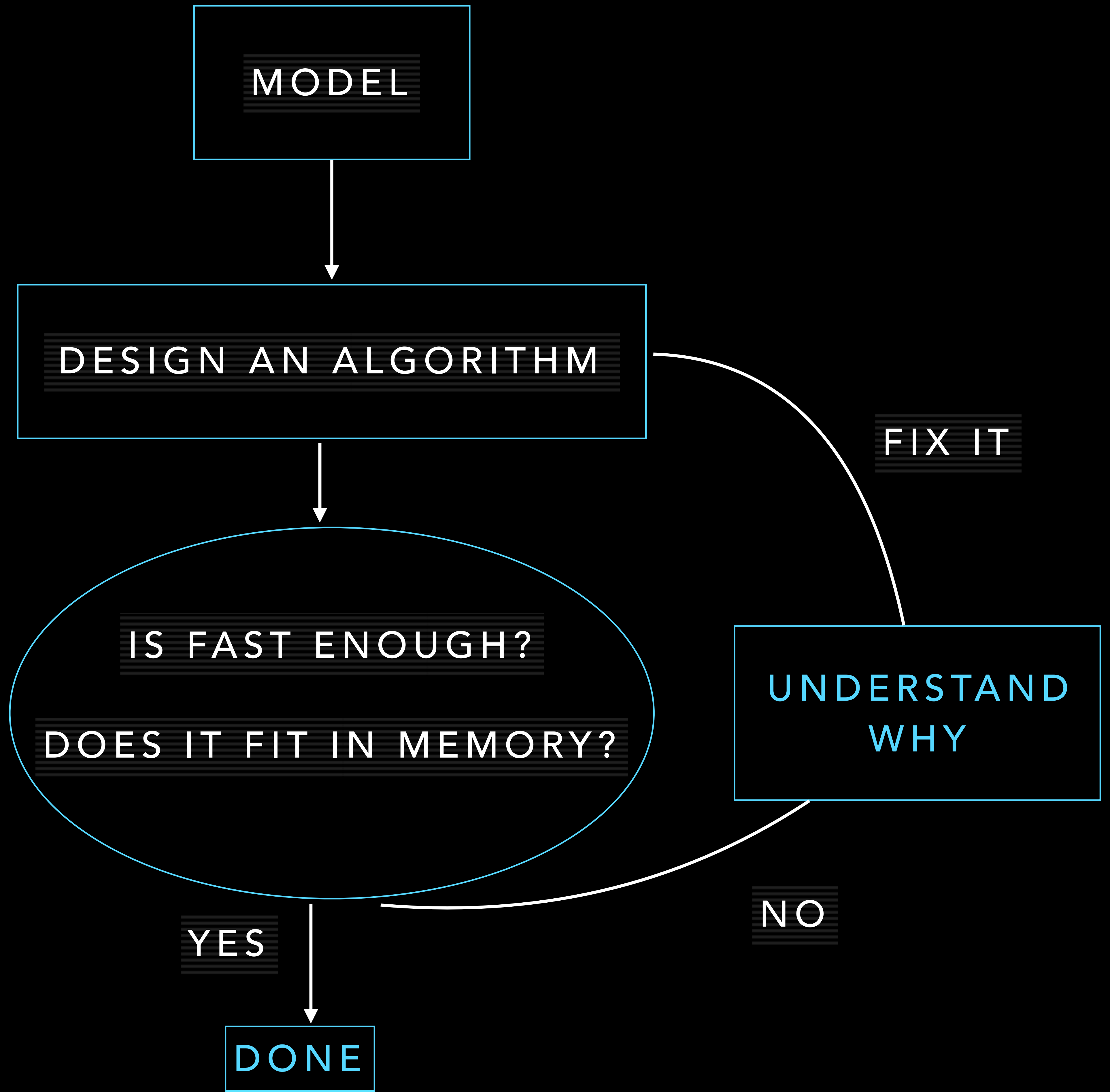
† INCLUDES COST OF FINDING ROOTS

Quick-find defect.

- Union too expensive (N array accesses).

Quick-union defect.

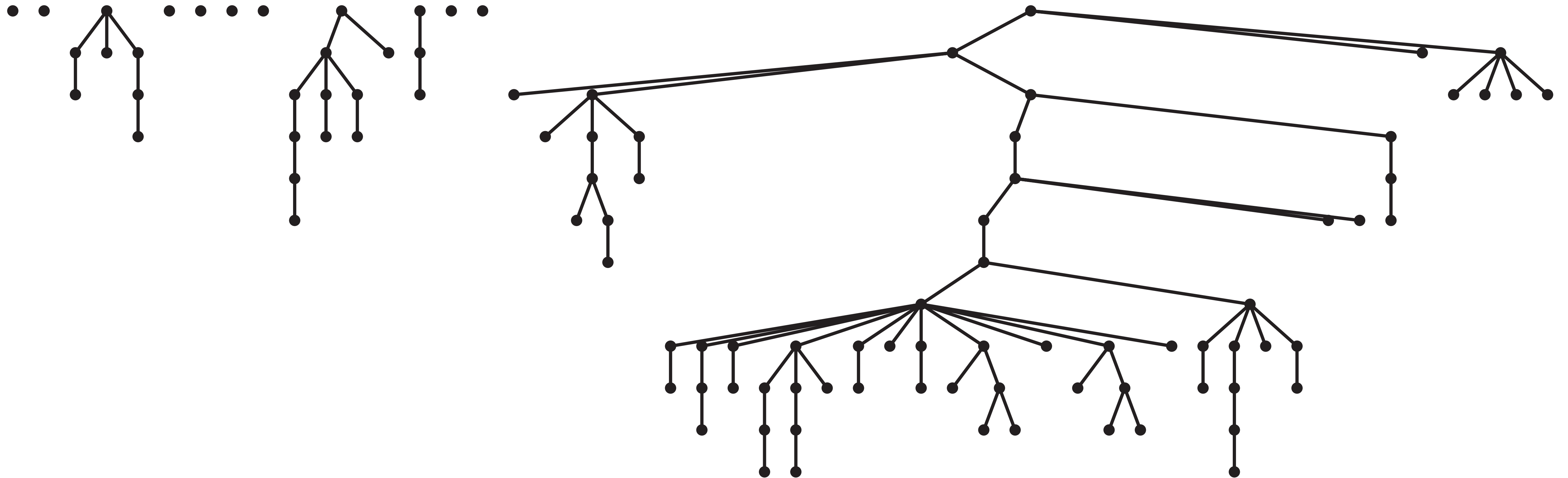
- Trees can get tall.
- Find/connected too expensive (could be N array accesses).

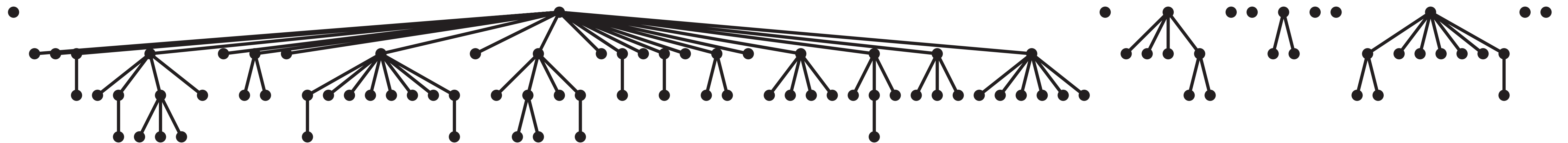

MODEL
DESIGN AN ALGORITHM
FIX IT
IS FAST ENOUGH?
DOES IT FIT IN MEMORY?
UNDERSTAND
WHY
NO
YES
DONE

NEED SOME WAY TO BALANCE THE TREES

A WEIGHTED APPROACH

QUICK-UNION AND WEIGHTED



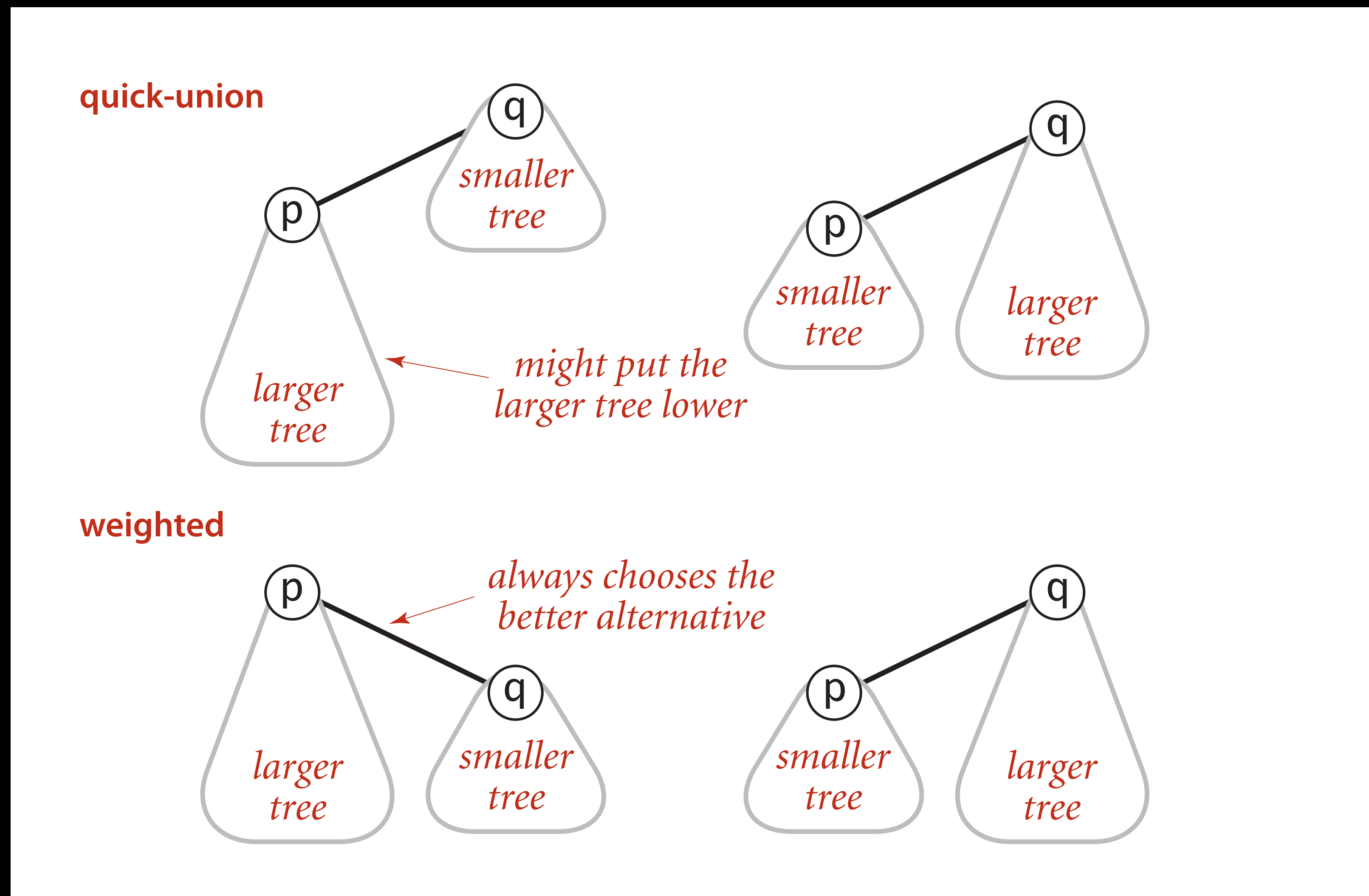


Quick-union and weighted quick-union (100 sites, 88 `union()` operations)

IMPROVEMENT 1: WEIGHTING

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each tree (number of objects).
- Balance by linking root of smaller tree to root of larger tree.



WHERE SIZE IS DETERMINE BY TOTAL NUMBER OF CHILDREN

WEIGHT UNION QUIZ

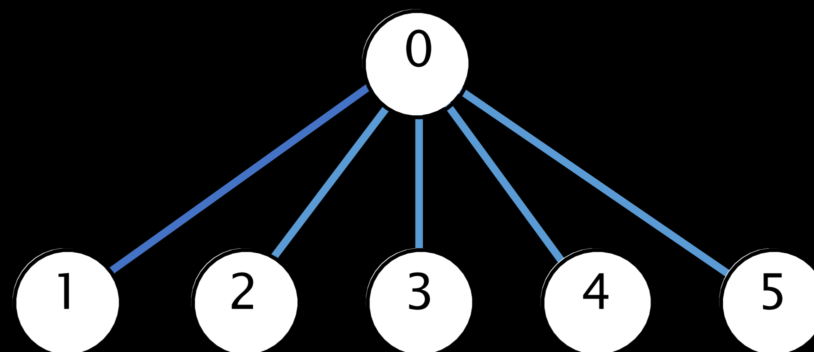
Which parent entry changes during union (2, 6)

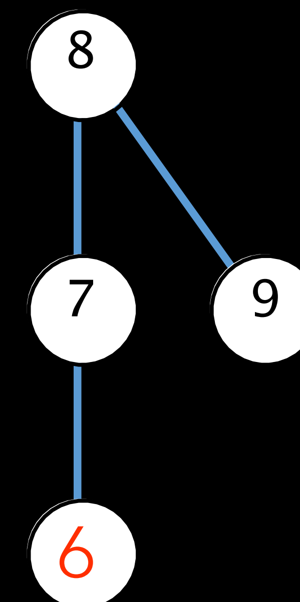
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	7	8	8	8

WEIGHT UNION QUIZ

Which parent entry changes during union (2, 6)

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





PARENT 8

HOW WOULD WE IMPLEMENT WEIGHTED UNION?

IMPLEMENTING WEIGHTED UNION

How would we keep track of number of children?

Maintain an array `sz[i]`to count number of objects in the tree rooted at `i`.

What about the find and connected operations

The don't change

What about the union operation

```
int i = find(p);
int j = find(q);
if (i == j) return;
if  (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else                { id[j] = i; sz[i] += sz[j]; }
```

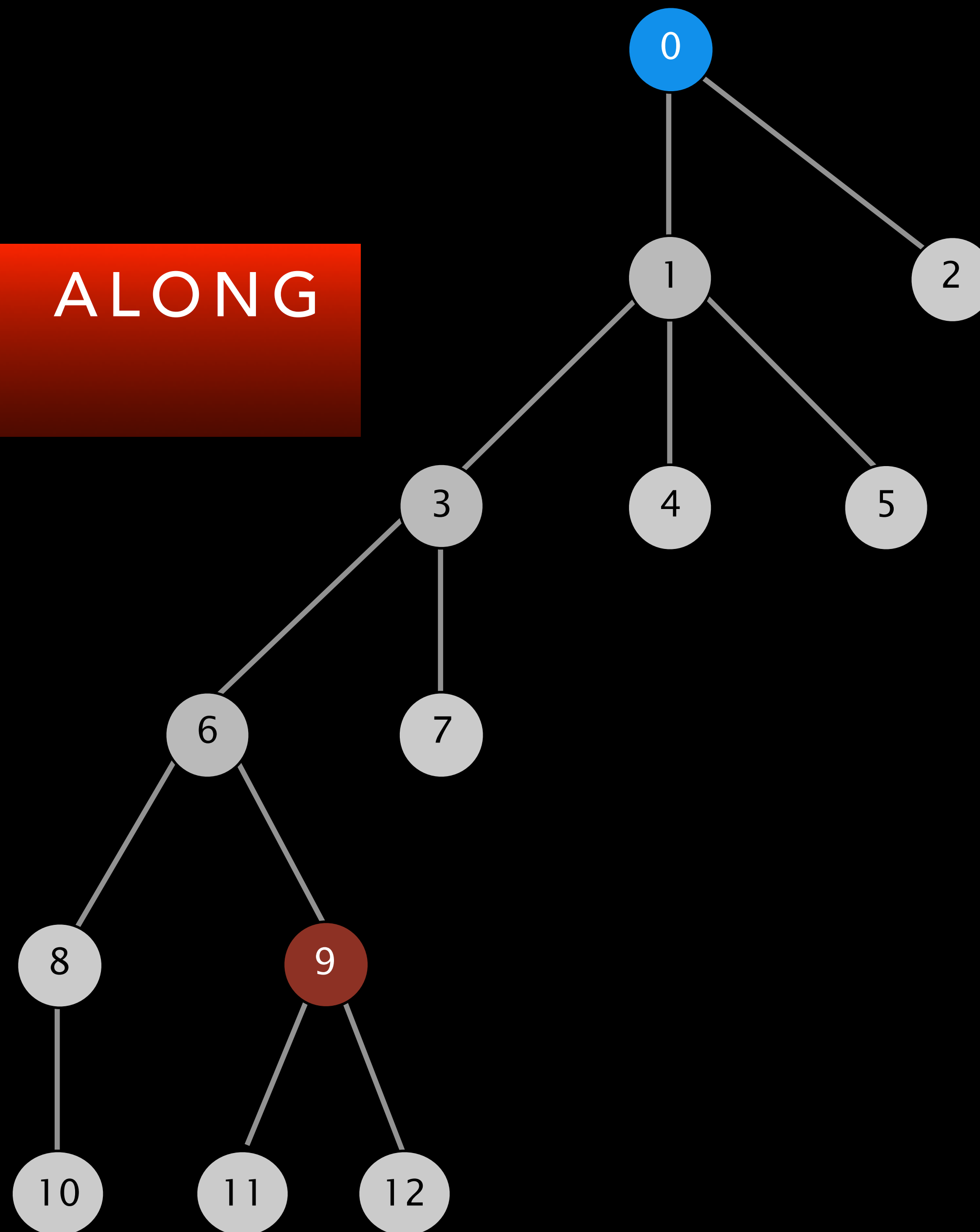
CAN WE THINK OF ANY MORE IMPROVEMENTS

CAN WE MAKE ANY OPTIMIZATIONS AS WE CALCULATE THE ROOT?

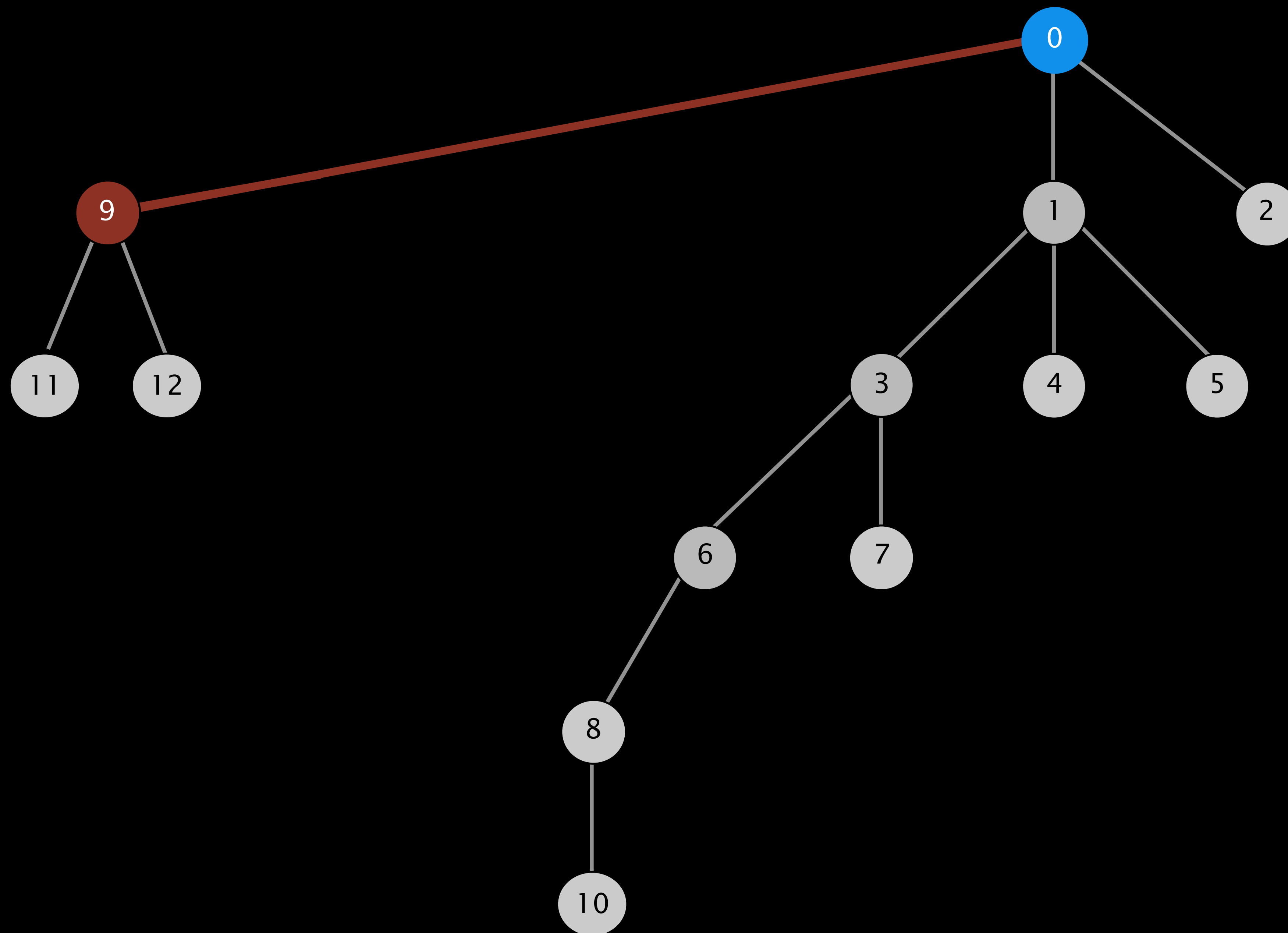
IMPROVEMENT 2: PATH COMPRESSION

FIND 9

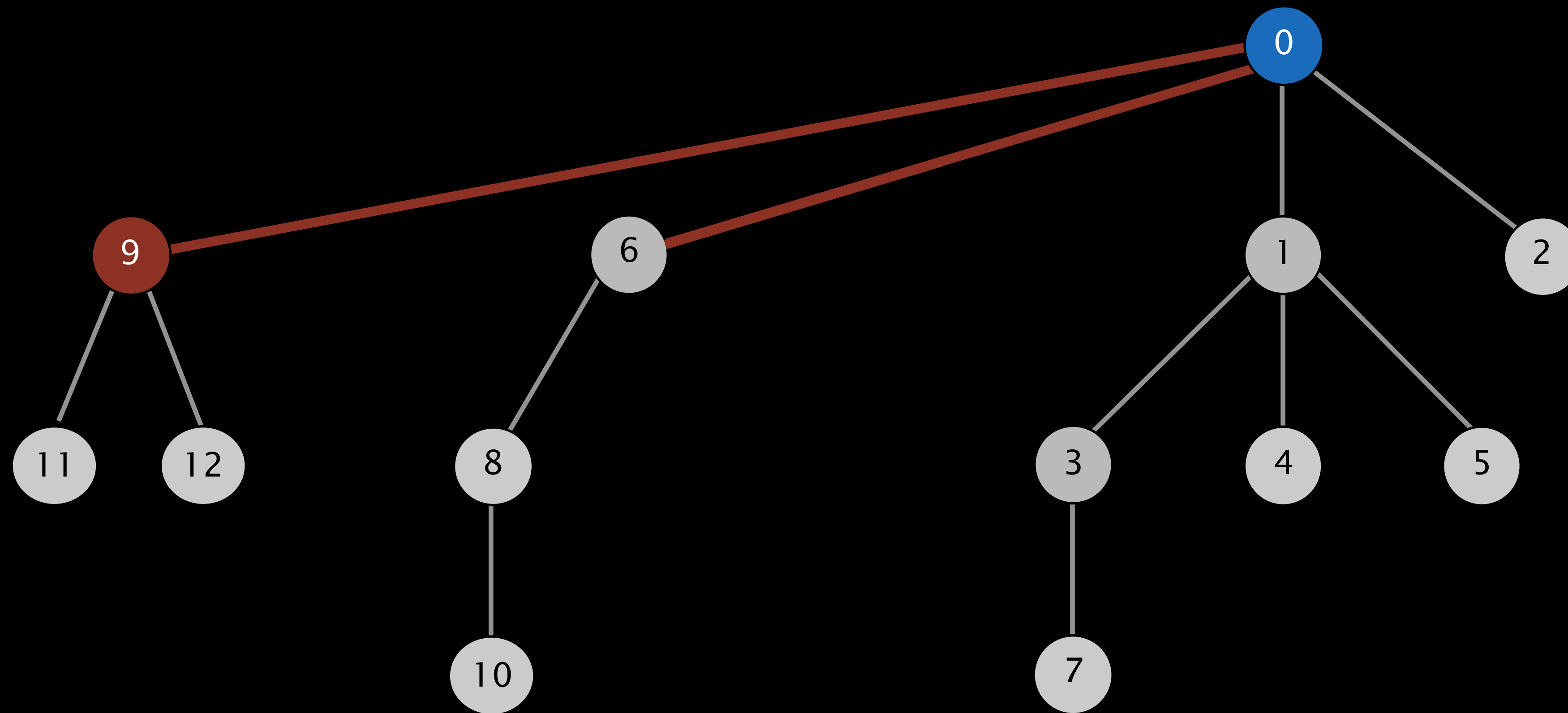
INSIGHT SET ALL NODES ALONG PATH TO ROOT



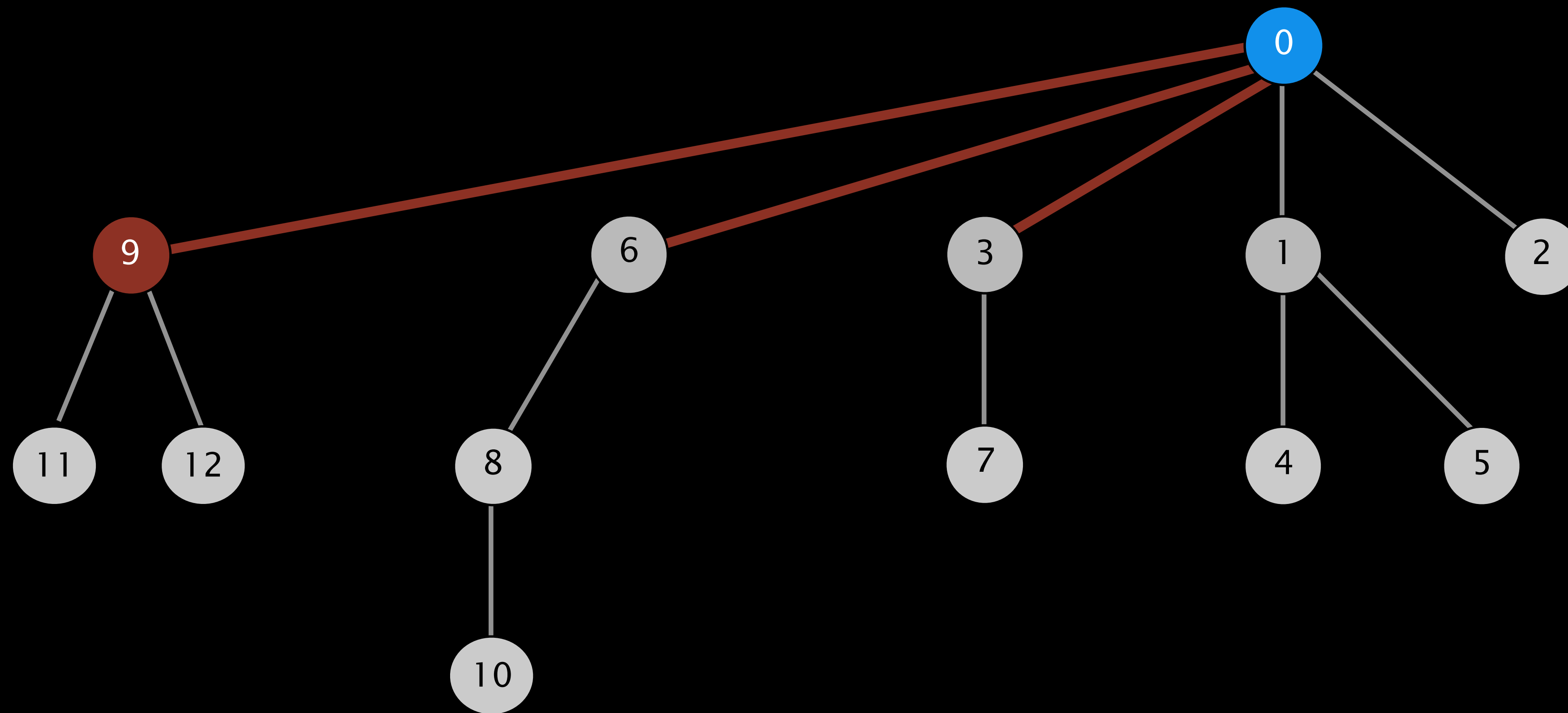
IMPROVEMENT 2: PATH COMPRESSION



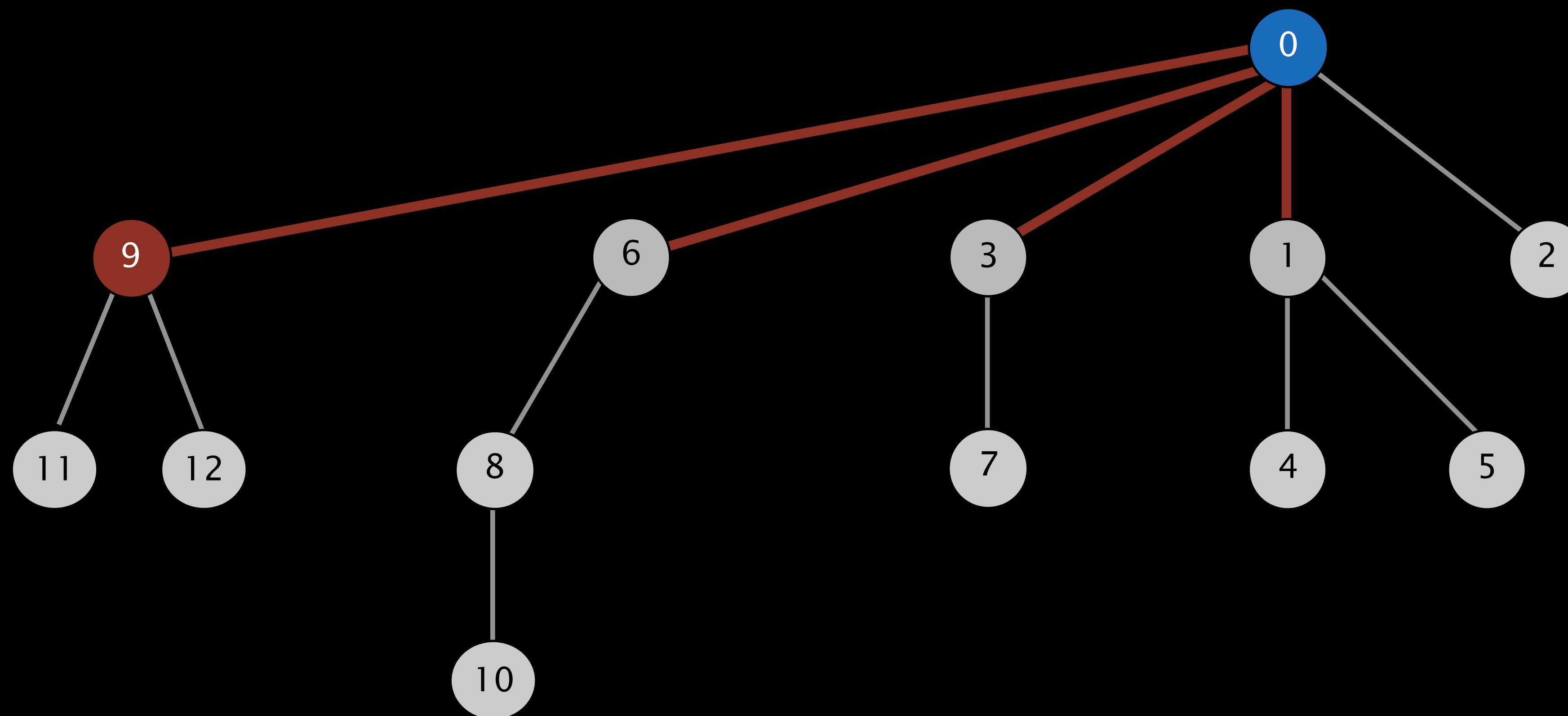
IMPROVEMENT 2: PATH COMPRESSION



IMPROVEMENT 2: PATH COMPRESSION



IMPROVEMENT 2: PATH COMPRESSION



BOTTOM LINE. NOW, `FIND()` HAS THE SIDE EFFECT OF COMPRESSING THE TREE.

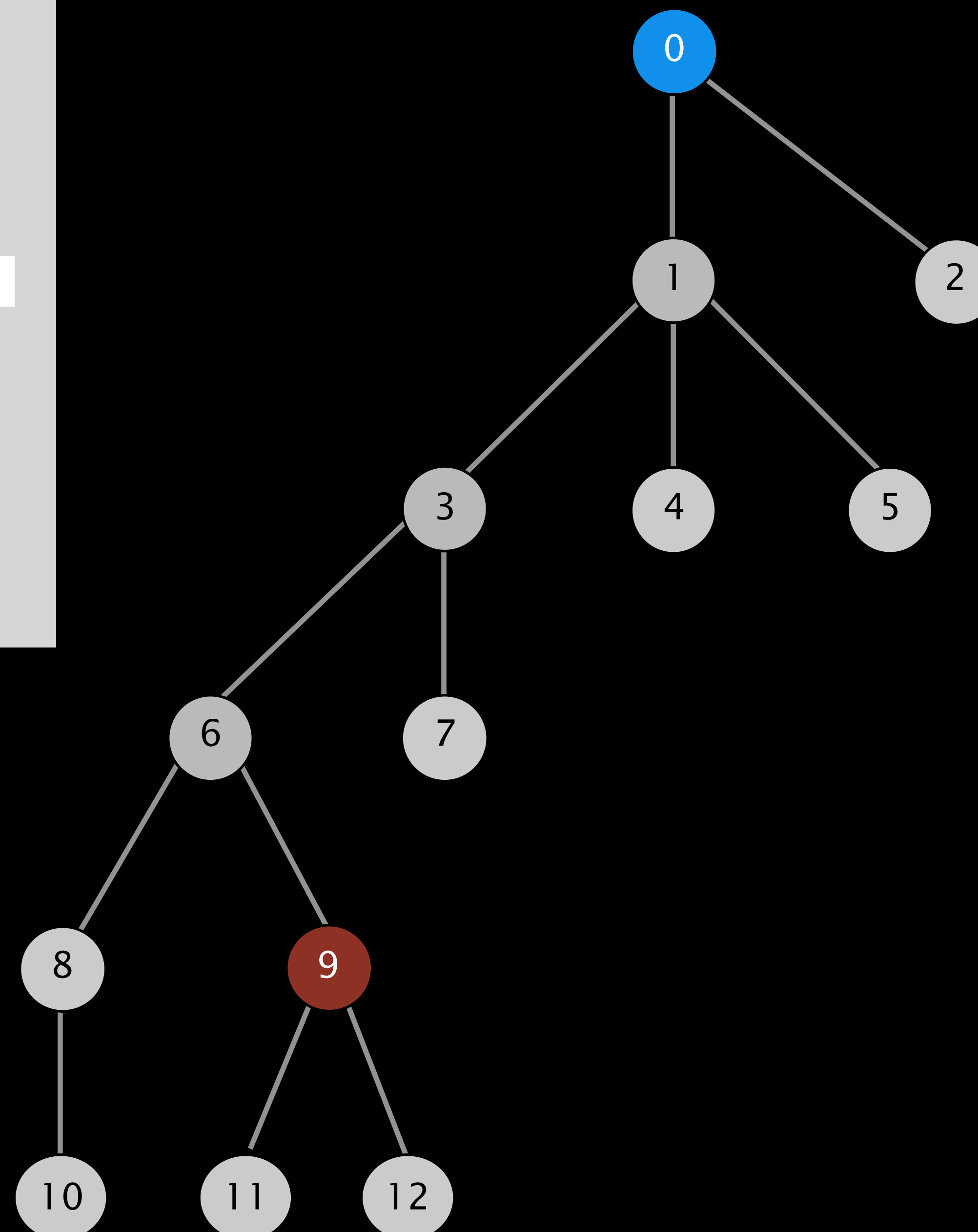
PATH COMPRESSION: JAVA IMPLEMENTATION

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

only one extra line of code !

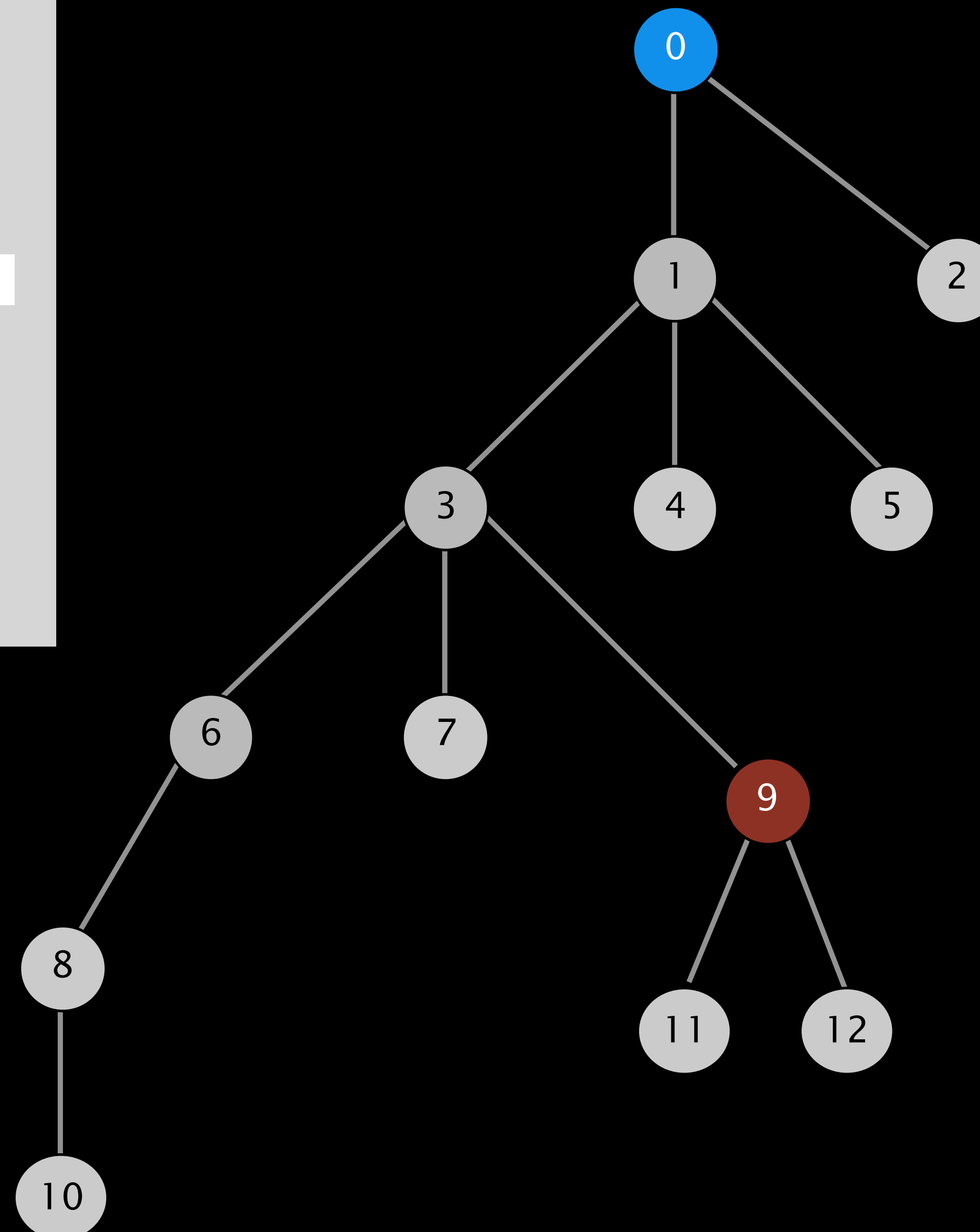
PATH COMPRESSION: JAVA IMPLEMENTATION

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



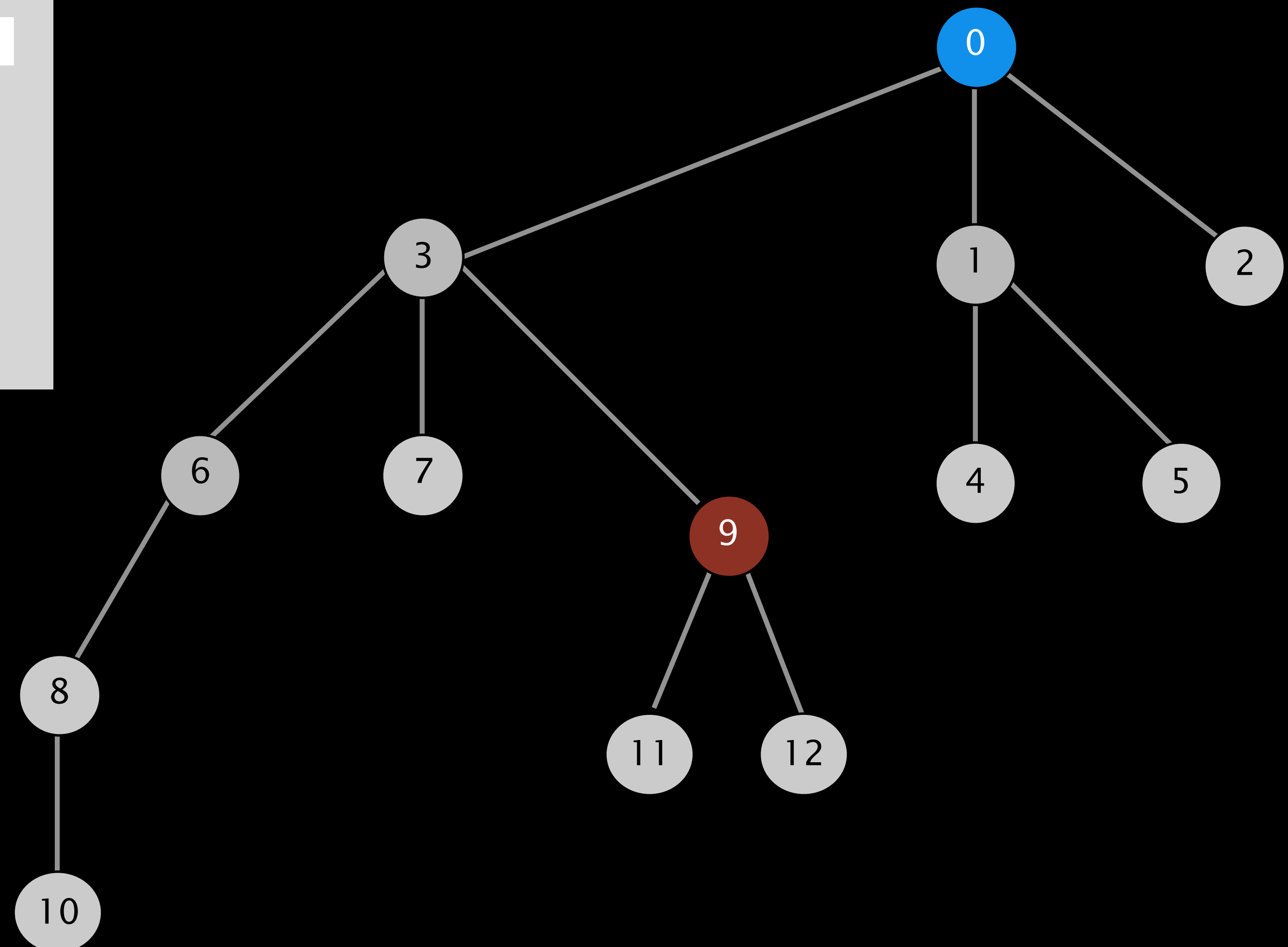
PATH COMPRESSION: JAVA IMPLEMENTATION

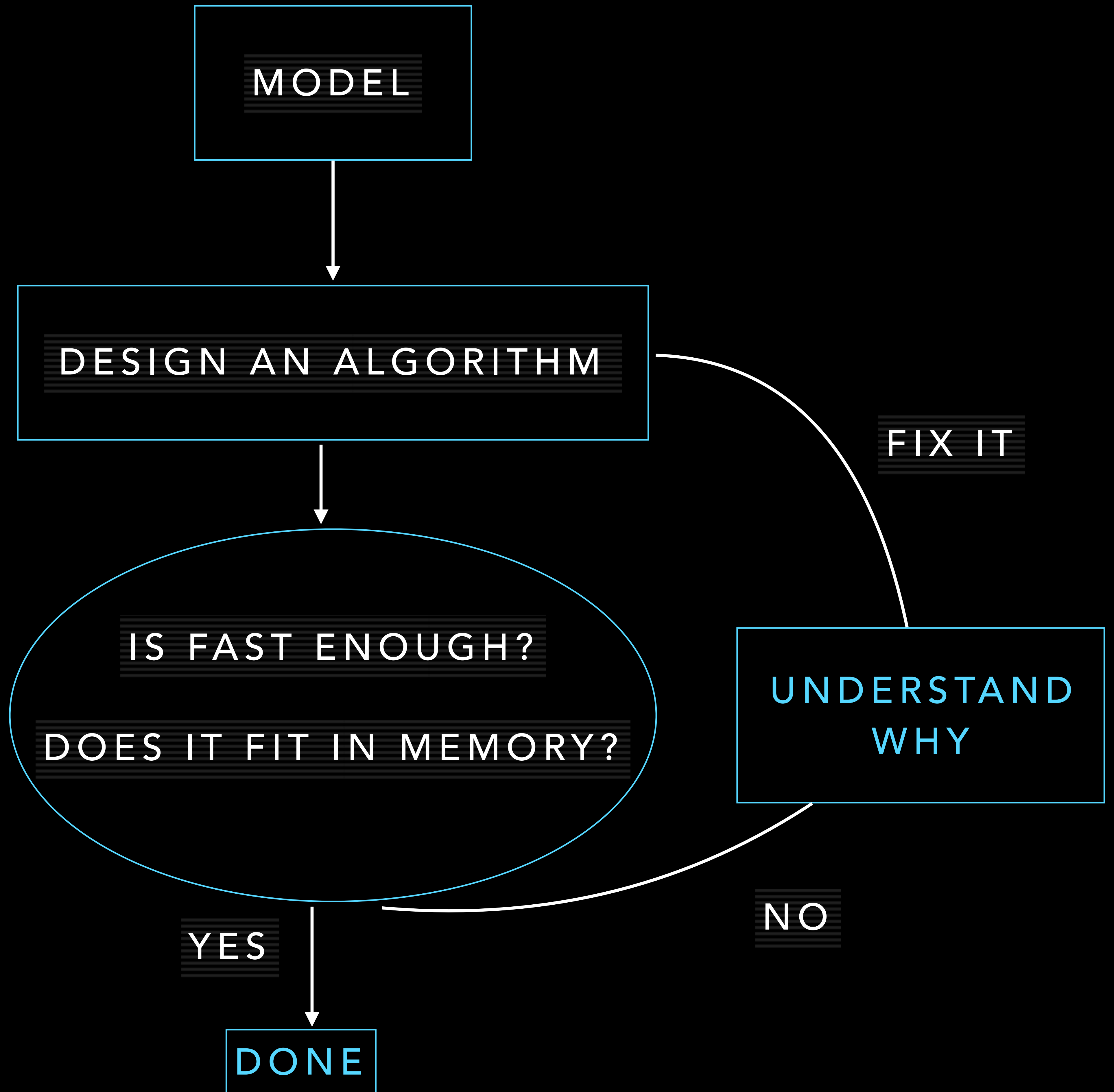
```
public int find(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```



PATH COMPRESSION: JAVA IMPLEMENTATION

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




MODEL
DESIGN AN ALGORITHM
FIX IT
IS FAST ENOUGH?
DOES IT FIT IN MEMORY?
UNDERSTAND
WHY
NO
YES
DONE

REFERENCES

- Robert Sedgewich & Kevin Wayne

  Section 1.5