CS 225

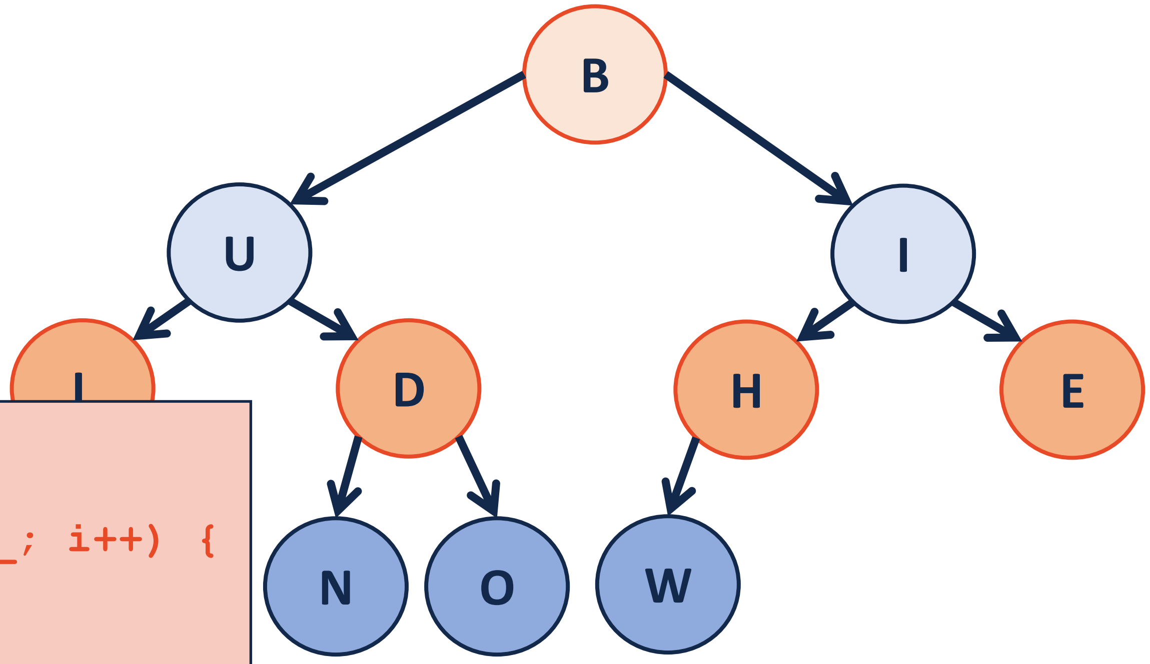
Data Structures

Nov. 8 – Disjoint Sets

Wade Fagen-Ulmschneider

buildHeap

1. Sort the array:



2.

```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = 0; i <= size_; i++) {
    heapifyUp(i);
  }
}
```

3.

```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = parent(size); i > 0; i--) {
    heapifyDown(i);
  }
}
```

	B	U	I	L	D	H	E	A	P	N	O	W			

Proving buildHeap Running Time

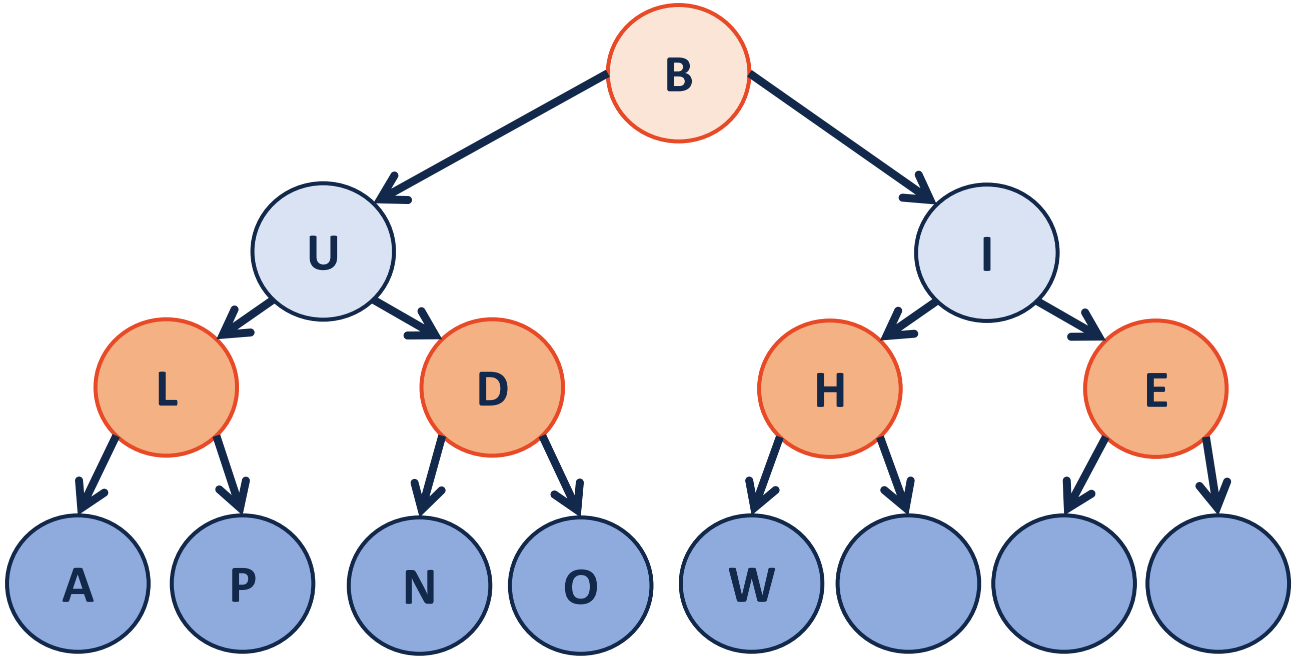
Theorem: The running time of buildHeap on array of size **n** is: _**O(n)**___.

Strategy:

- We know that constant work is done based on the distance a node is away from the root (eg: it's height).
- Therefore, the running time is proportional to the sum of the heights of the heights of all the nodes.
- We will work towards creating a proof around the sum of the heights of all the nodes.

Proving buildHeap Running Time

S(h): Sum of the heights of all nodes in a ~~complete~~ perfect tree of height **h**.



S(0) = $\mathbf{0}$

S(1) = $\mathbf{1}$

S(2) = $\mathbf{4}$

S(h) = $\mathbf{2S(h-1) + h}$

$\mathbf{= 2^{(h+1)} - 2 - h}$

Proving buildHeap Running Time

We proved the recurrence:

$S(h) = 2S(h-1) + h = 2^{(h+1)} - 2 - h$

Proving buildHeap Running Time

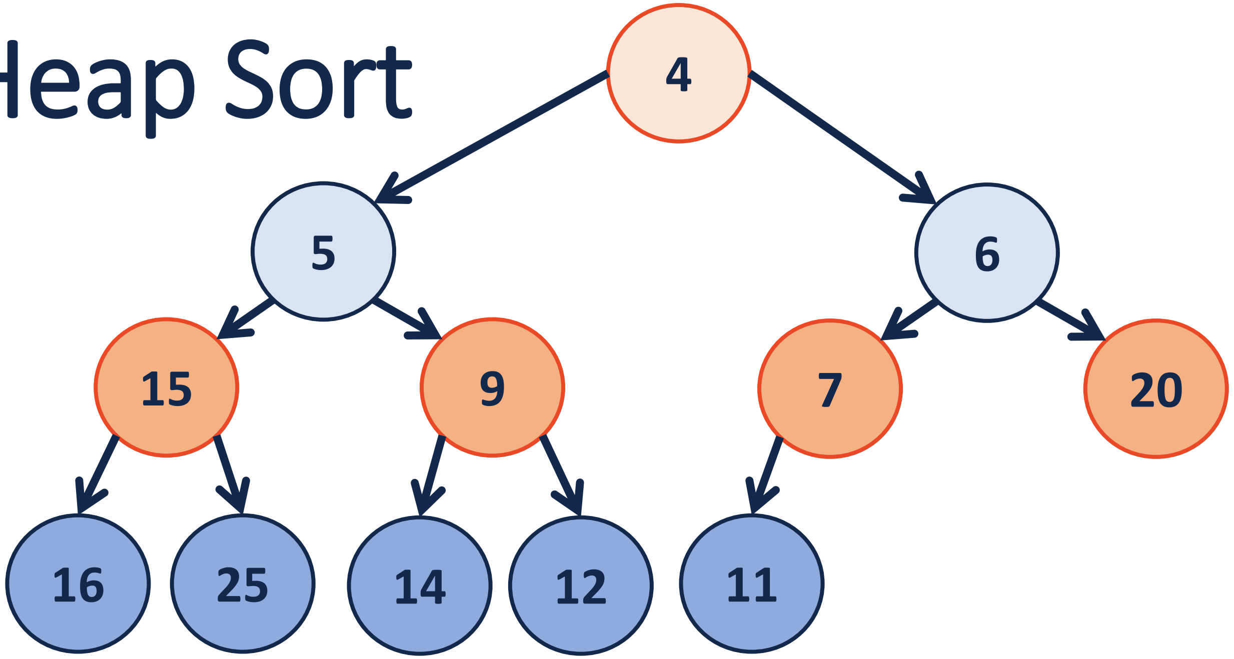
No one cares about things in terms of height:

$S(h) = \mathbf{2^{(h+1)} - 2 - h}$

We know that the nodes in a perfect tree of height **h** is:

$n =$

Heap Sort



1.

2.

3.

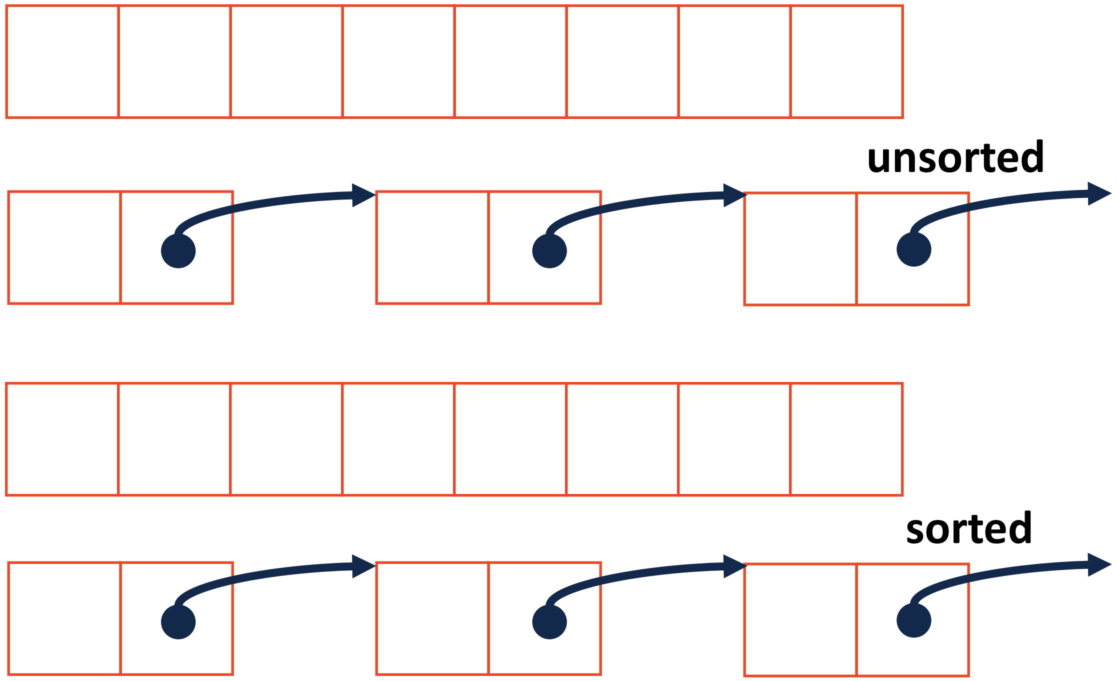
	4	5	6	15	9	7	20	16	25	14	12	11			

Running Time?

Why do we care about another sort?

Priority Queue Implementation

insert	removeMin	buildHeap
$O(1)^A$	O(n)	
O(1)	O(n)	
O(n)	O(1)	
O(n)	O(1)	



AVL Tree

Heap

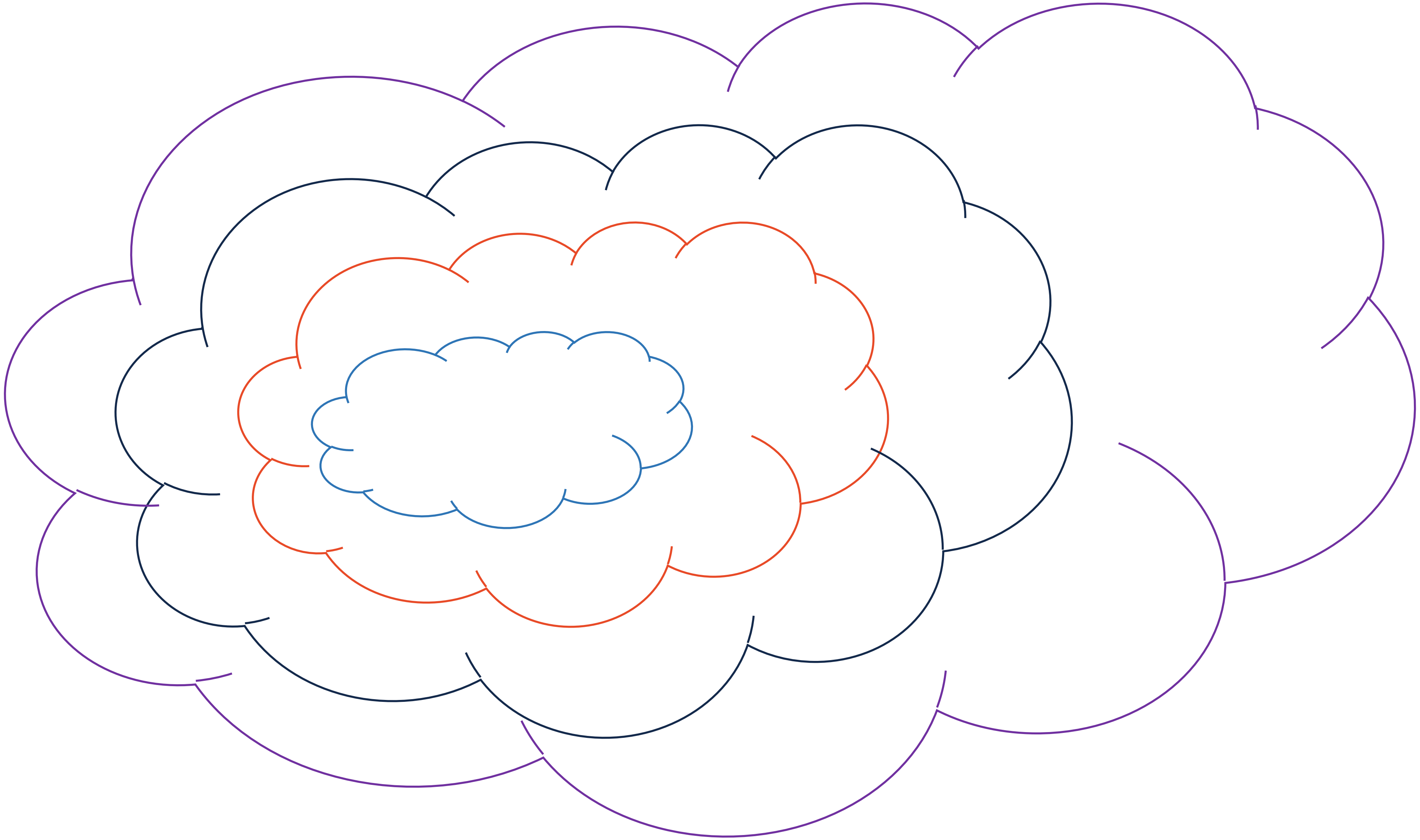
MPs to finish the semester

Fall break is ***almost here – 1.5 more weeks*!**

MP6: A quick case study on MP5.
- Released today around 4:00pm
- Due next Friday (the Friday before break), no EC deadline

MP7: The big finale for CS 225!
- Released next Tuesday
- Due Dec. 11 (4 weeks), has 3 parts, +14 points of EC!

Array Abstractions

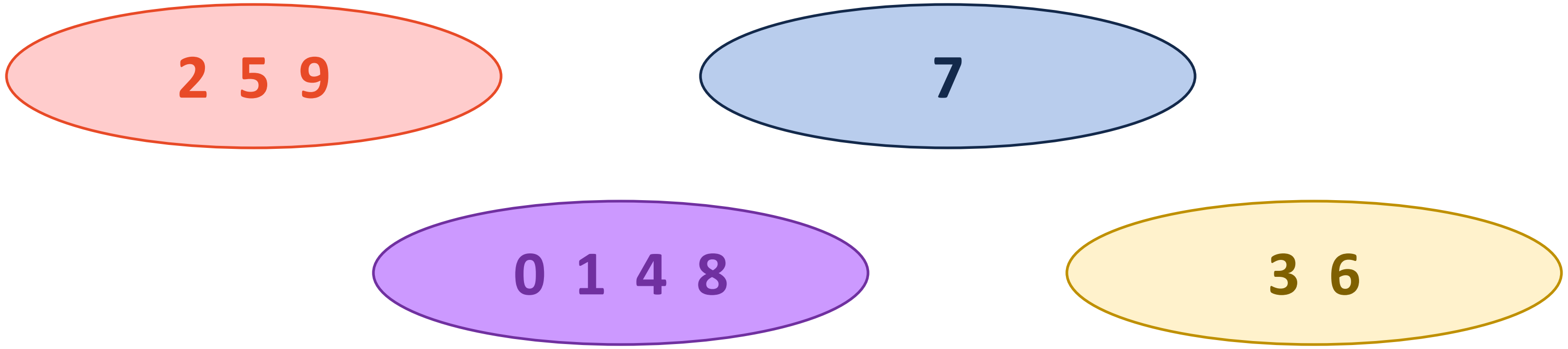


A(nother) throwback to CS 173…

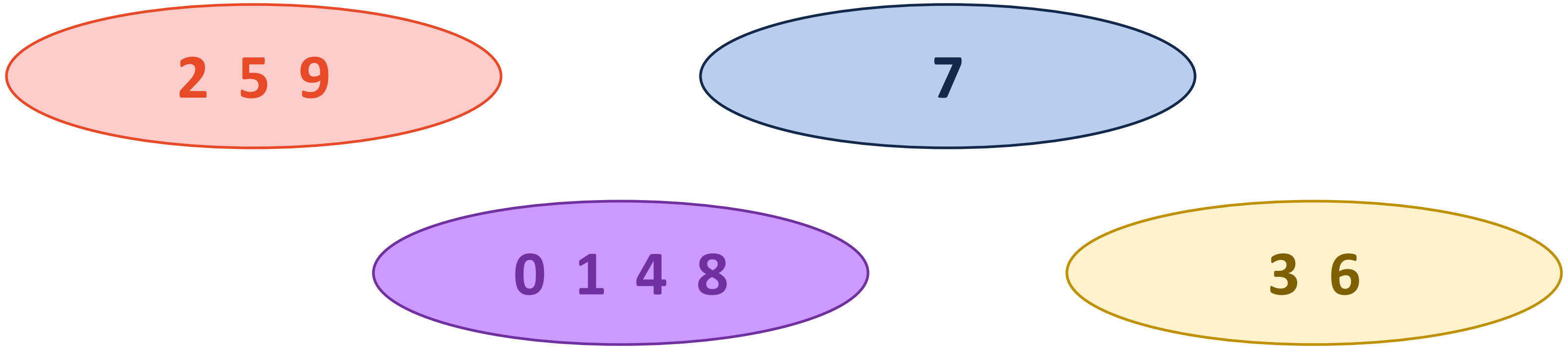
Let **R** be an equivalence relation on *us* where $(\mathbf{s}, \mathbf{t}) \in \mathbf{R}$ if **s** and **t** have the same favorite among:

{ ____, ____, _____, ____, _____, ____ }

Disjoint Sets

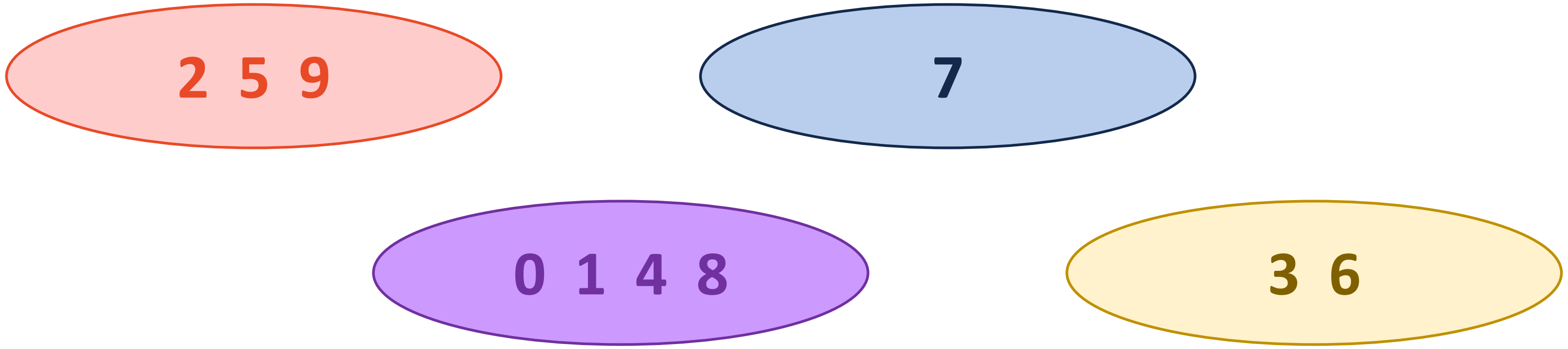


Disjoint Sets



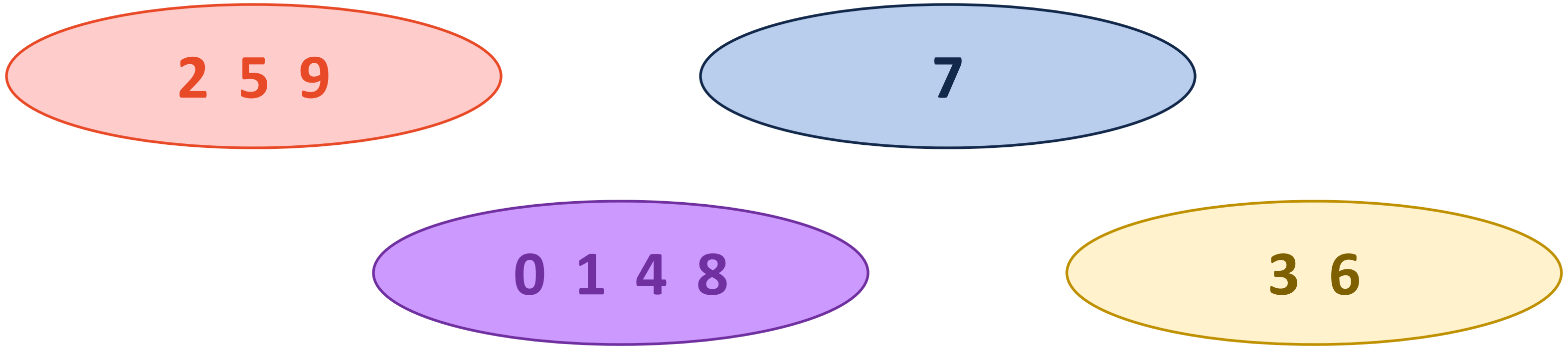
Operation: find(4)

Disjoint Sets



Operation: find(4) == find(8)

Disjoint Sets



Operation:

```
if ( find(2) != find(7) ) {
    union(   find(2), find(7)    );
}
```

Disjoint Sets ADT

- Maintain a collection $S = \{s_0, s_1, \ldots s_k\}$

- Each set has a representative member.

- API:
```
void makeSet(const T & t);
void union(const T & k1, const T & k2);
T & find(const T & k);
```

Implementation #1

0 1 4

2 7

3 5 6

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

Find(k):

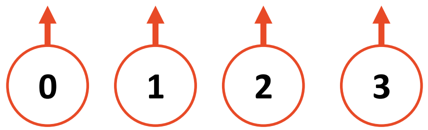
Union(k1, k2):

Implementation #2

- We will continue to use an array where the index is the key

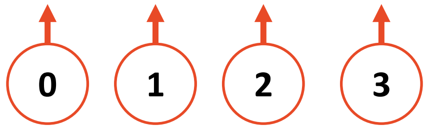
- The value of the array is:
  - **-1,** if we have found the representative element
  - **The index of the parent**, if we haven't found the rep. element

- We will call theses **UpTrees**:



0	1	2	3
-1	**-1**	**-1**	**-1**

UpTrees



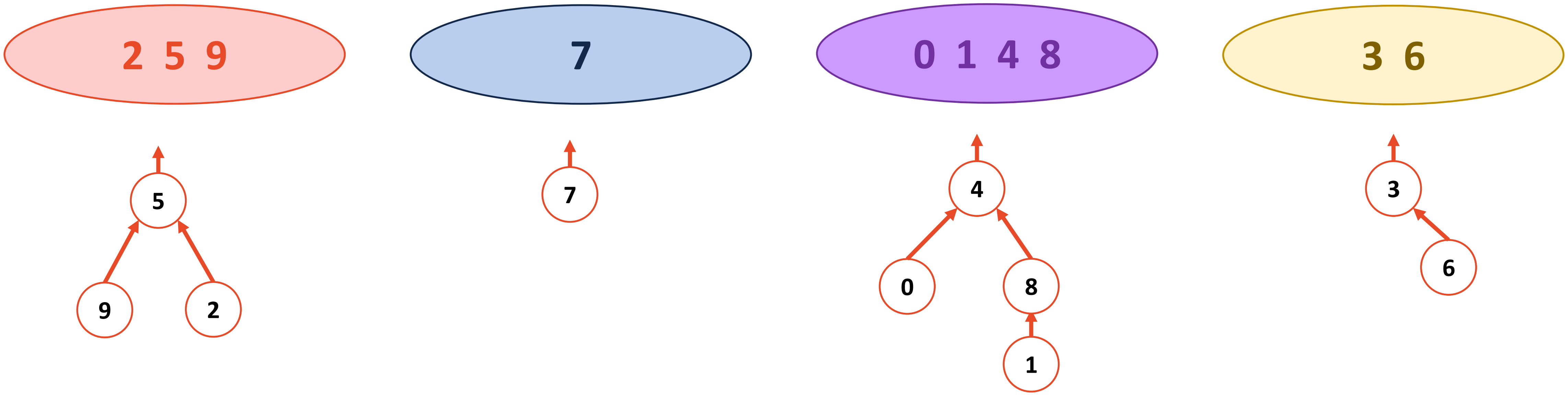
0	1	2	3
-1	-1	-1	-1

0	1	2	3

0	1	2	3

0	1	2	3

Disjoint Sets



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

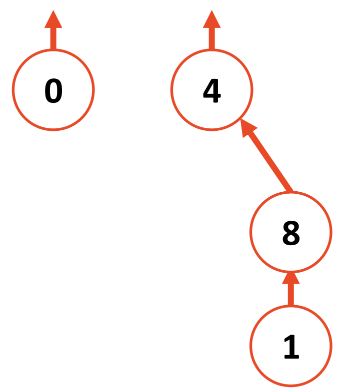
Disjoint Sets Find

```
int DisjointSets::find() {
  if ( s[i] < 0 ) { return i; }
  else { return _find( s[i] ); }
}
```

Running time?

```
void DisjointSets::union(int r1, int r2) {


}
```



CS 225 – Things To Be Doing

Exam 9 (theory, trees) is ongoing!

More Info: https://courses.engr.illinois.edu/cs225/fa2017/exams/

MP6: One week MP*

Due Monday, Nov. 17 at 11:59pm

Lab: lab released today

Due Sunday, Nov. 12 at 11:59pm

POTD

Every Monday-Friday – *Worth +1 Extra Credit /problem (up to +40 total)*