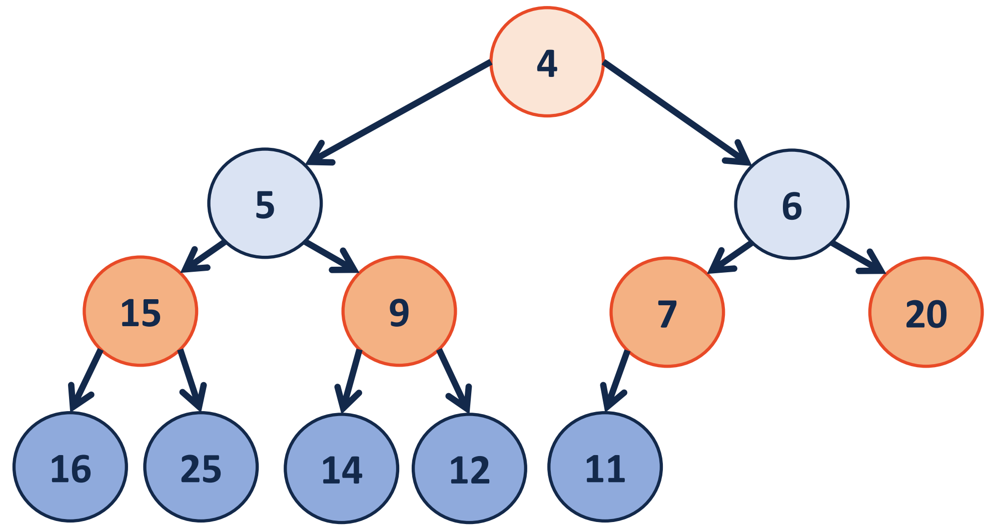
CS 225

Data Structures

March 10 – Heaps

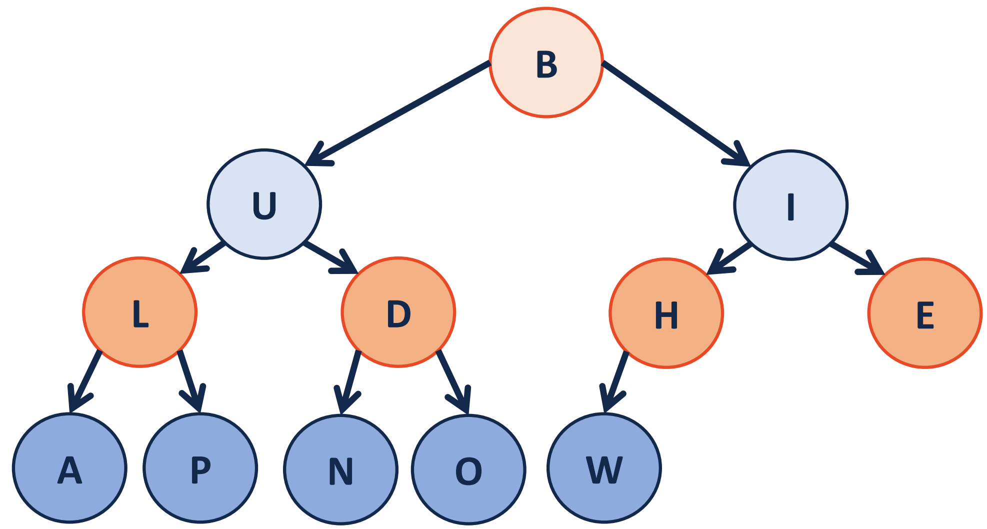
G Carl Evans

(min)Heap



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

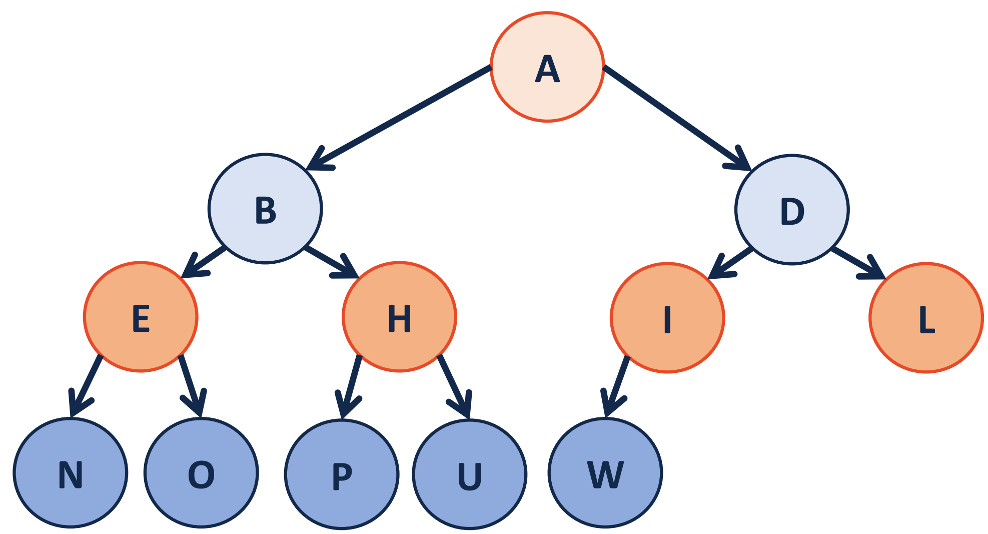
buildHeap



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

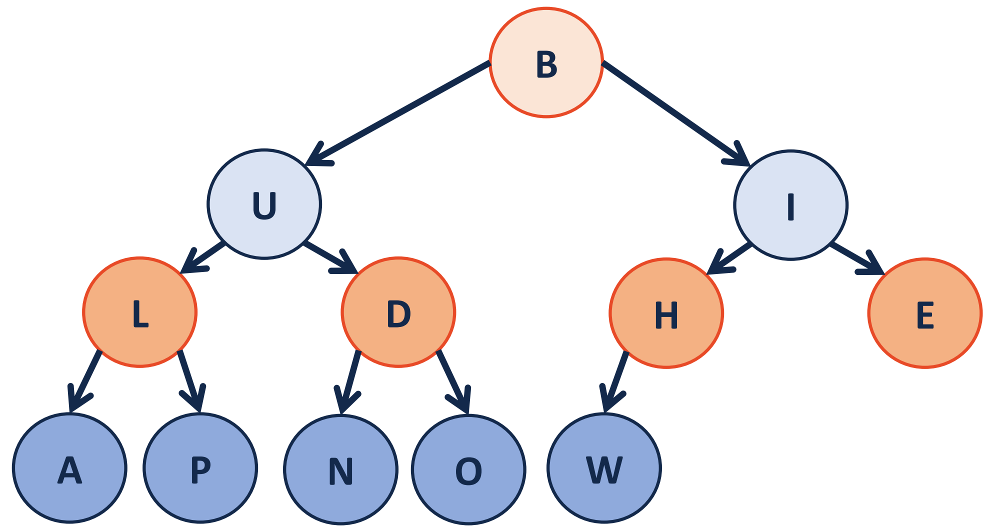
buildHeap – sorted array

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



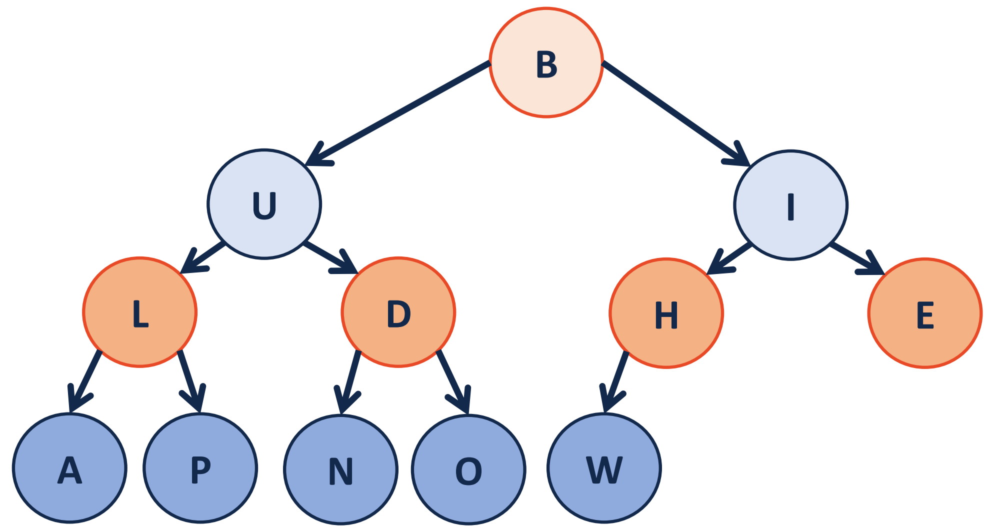
	A	B	D	E	H	I	L	N	O	P	U	W			

buildHeap - heapifyUp



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

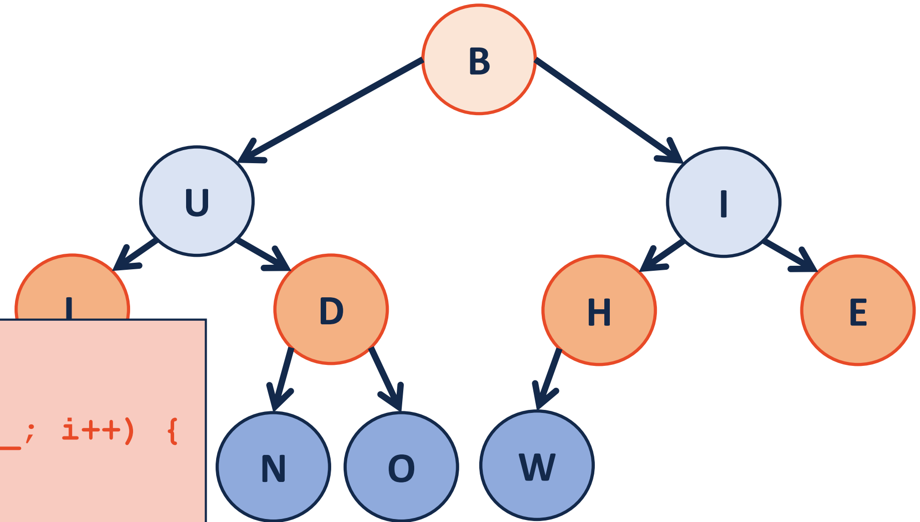
buildHeap - heapifyDown



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

buildHeap

1. Sort the array – it's a heap!



2.

```
template <class T>
void Heap<T>::buildHeap() {
  for (unsigned i = 2; 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: __________.

Strategy:

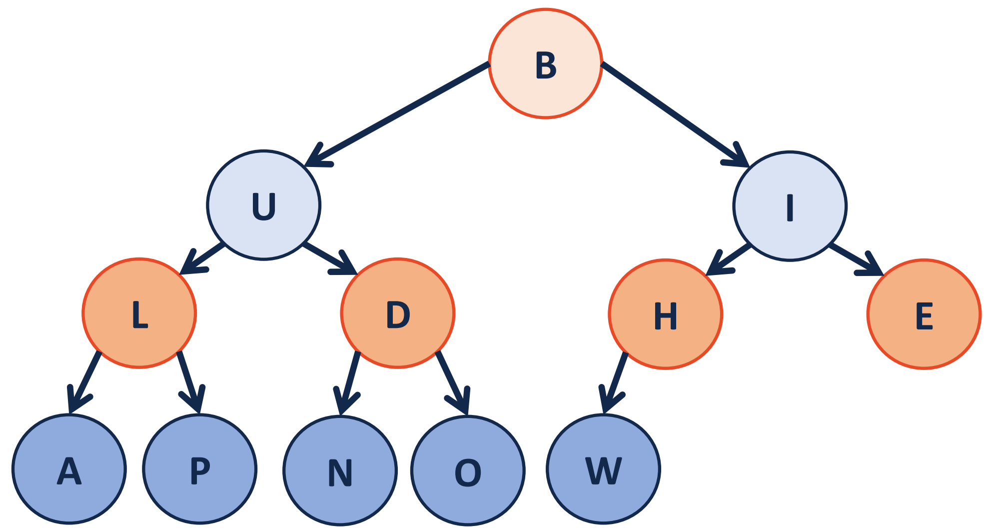
-

-

-

Proving buildHeap Running Time

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



S(0) =

S(1) =

S(h) =

Proving buildHeap Running Time

Proof the recurrence:

Base Case:

General Case:

Proving buildHeap Running Time

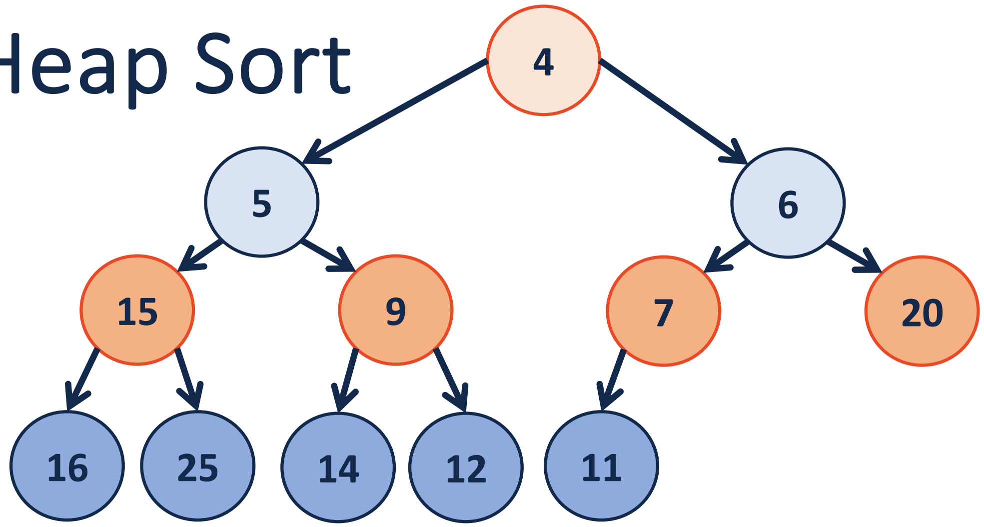
From S(h) to RunningTime(n):

S(h):

Since h ≤ lg(n):

RunningTime(n) ≤

Heap Sort



1.

2.

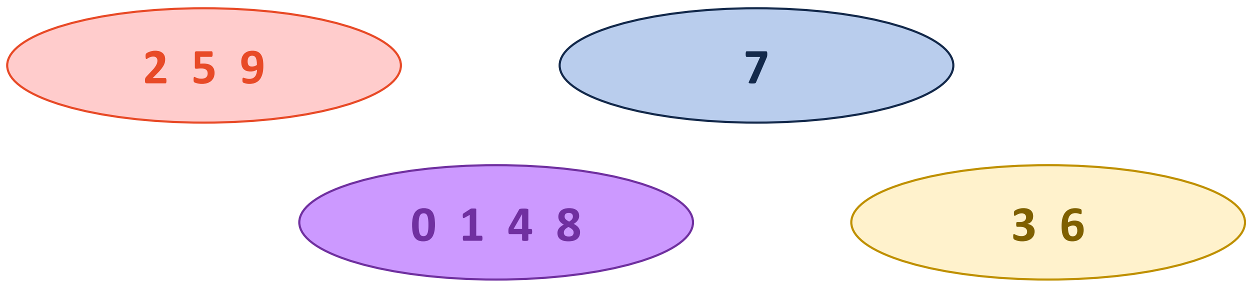
3.

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

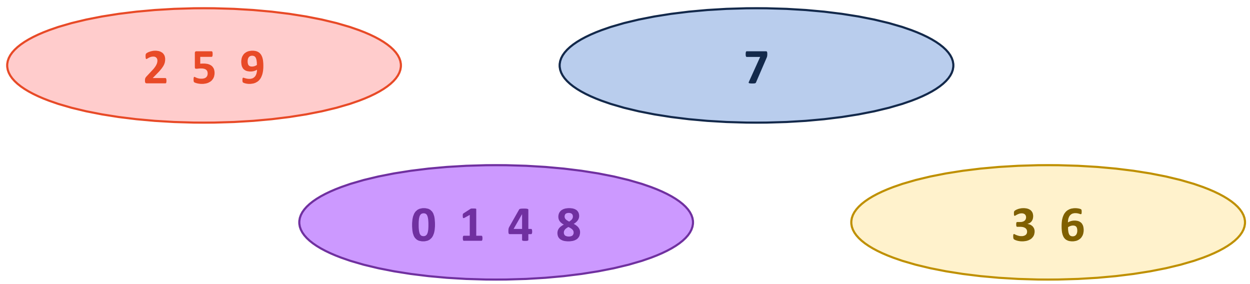
Running Time?

Why do we care about another sort?

Disjoint Sets

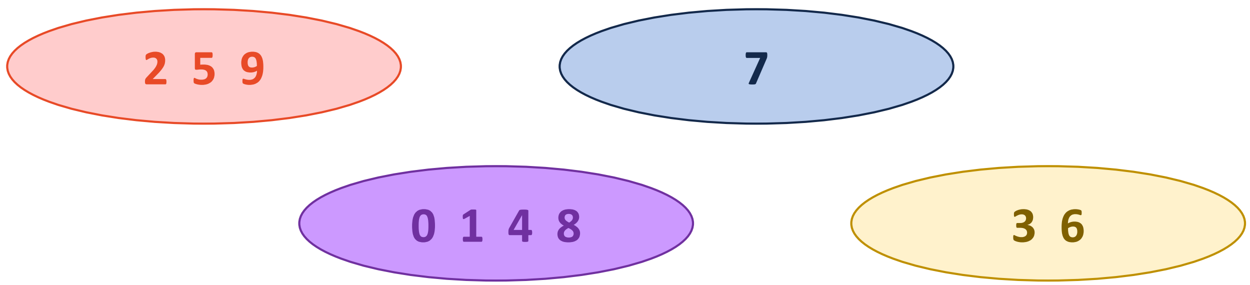


Disjoint Sets



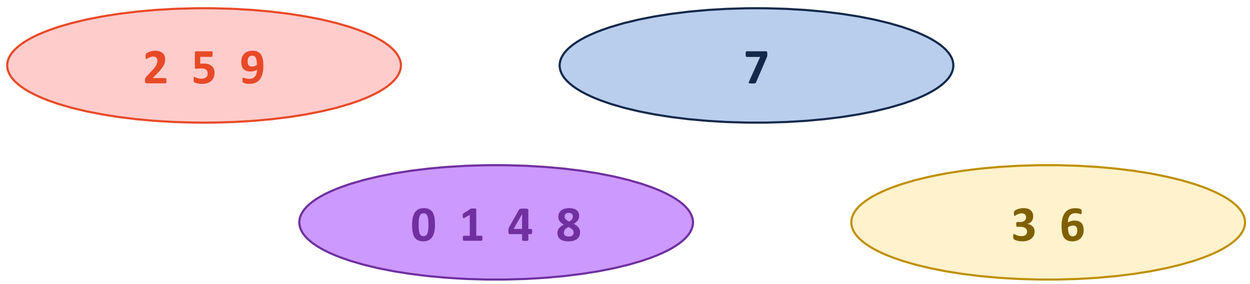
Operation: find(4)

Disjoint Sets



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

Disjoint Sets



Key Ideas:

- Each element exists in exactly one set.
- Every set is an equitant representation.
  - Mathematically: $4 \in [0]_R \rightarrow 8 \in [0]_R$
  - Programmatically: find(4) == find(8)

Disjoint Sets ADT

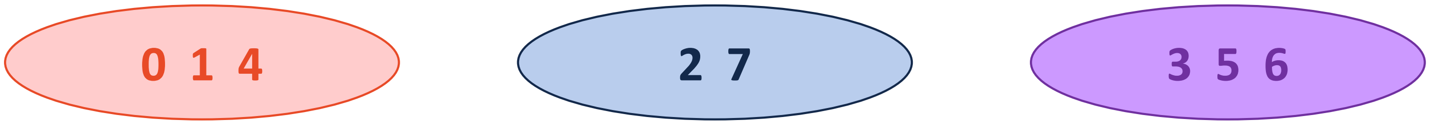
- Maintain a collection $S = \{s_0, s_1, \ldots s_k\}$

- Each set has a representative member.

- API:

```
void makeSets(int number);
void union(int k1, const int k2);
int  find(int k);
```

Implementation #1



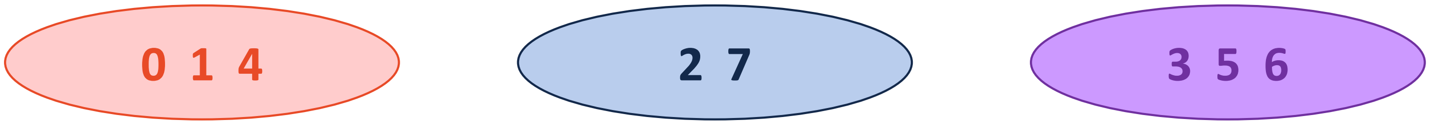
0	1	2	3	4	5	6	7

Find(k):

Union(k1, k2):


YOU EXPECTED A NEW DATA STRUCTURE
BUT IT WAS ME, TREE ALL ALONG
imgflip.com

Implementation #2



0	1	2	3	4	5	6	7

Find(k):

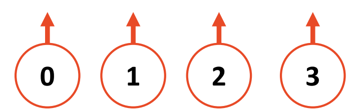
Union(k1, k2):

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**