A complete binary tree $T$ is a min-heap if:

- $T = \emptyset$ or
- $T = \{r, T_L, T_R\}$, where $r$ is less than the roots of $\{T_L, T_R\}$ and $\{T_L, T_R\}$ are min-heaps.
buildHeap
buildHeap – sorted array
buildHeap - heapifyUp
buildHeap - heapifyDown
**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);
       }
   }
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);
     }
   }
$O(h) = 2$
O(h) = 3
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 \( h \).

\[ S(0) = \]
\[ S(1) = \]
\[ S(2) = \]
\[ S(h) = \]
Proving buildHeap Running Time

**Proof the recurrence:**

Base Case:

IH:

General Case:
Proving buildHeap Running Time

From $S(h)$ to $\text{RunningTime}(n)$:

$S(h)$:

Since $h \leq \lg(n)$:

$\text{RunningTime}(n) \leq$
Heap Sort

Running Time?

Why do we care about another sort?
Heap Sort

Running Time?

Why do we care about another sort?
Another throwback to CS 173...

Let $R$ be an equivalence relation on $us$ where $(s, t) \in R$ 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:

```java
if ( find(2) != find(7) ) {
    union( find(2), find(7) );
}
```
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 makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);