Q: An optimal buildHeap operation:

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

Theorem: The running time of buildHeap on array of size n is:

_______.

Strategy:

Define S(h):

\[ S(h) = \]

Disjoint Sets

Let \( R \) be an equivalence relation. We represent \( R \) as several disjoint sets. Two 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: \( \text{find}(4) == \text{find}(8) \)

Building Disjoint Sets:

- Maintain a collection \( S = \{s_0, s_1, \ldots, s_k\} \)
- Each set has a representative member.
- ADT:
  
  ```
  void makeSet(const T & t);
  void union(const T & k1, const T & k2);
  T & find(const T & k);
  ```
Implementation #1: Representative Member Array

Operation: find(k)  
...running time?

Operation: union(k1, k2)  
...running time?

Implementation #2: UpTrees

- 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

Step-by-step construction of UpTrees:

Example:

Implementation – DisjointSets::find

```
DisjointSets.cpp (partial)
1 int DisjointSets::find(int i) {
2     if ( s[i] < 0 ) { return i; }
3     else { return _find( s[i] ); }
4 }
```

What is the running time of find?

What is the ideal UpTree?

Implementation – DisjointSets::union

```
DisjointSets.cpp (partial)
1 void DisjointSets::union(int r1, int r2) {
2     // code here
3 }
```

How do we want to union the two UpTrees?

CS 225 – Things To Be Doing:

1. Theory Exam 3 starts Thursday; Practice Exam Available!
2. MP5 due tonight at 11:59pm
3. Lab Section: new lab coming up this week in lab!
4. Daily POTDs are ongoing!