Heap

Heap is a complete binary tree.

Operations:
- Insert
- Delete
- Extract

Properties:
- All nodes are in order from root to leaf.
- Parent node is greater than or equal to its children.

Example:

```
        1
       /\```
```
   2   3```
```
  4  5  6```
```
Exercise 7.1 and 7.2

Properties to explain, true \( O(n) \) in time

- All operations on a heap run in time
- Heap of a complete tree is \( O(n) \)
- Height of the tree = height of the root

In a heap, a node: A parent of a node is the node with the smallest value from node to

\[
\begin{array}{c}
\text{Input: An array } A \text{ and an index } i \text{ in array } A

\text{Heapsify maintains heap propety}
\end{array}
\]

Five basic procedures

1. Insert \( \text{ new heap entry} \)
2. Extract-Min \( \text{ number where the root is used as priority} \)
3. Heap-Remove \( \text{ number in place} \)
4. Heap-Insert \( \text{ entry in heap} \)
5. Heap-Delete-Min \( \text{ entry in heap} \)

\[ \text{A heap} \]
One can prove running time is in $O(n)$ node eliminates, or merge nodes and small
O operation. Can be call to heapify depends on height of
Higher node. And can be computed.

Upper bound correct? But not tighter

Running time $\Theta(n \log n)$

These are all lower $O(n \log n)$ calls

Each call to heapify costs $O(\log n)$

Simple upper bound on running time.