What is a heap?

**Structure:**
- binary tree with keys at the nodes
- all levels full, except possible the lowest;
  nodes in lowest level as far left as possible

**Heap property:**
- whenever $y$ is a child of $x$,
  $\text{key}(y) \leq \text{key}(x)$

![Diagram of a heap and a non-heap tree]

a heap

not a heap: why not?
Implementing a heap in an array

If node $x$ is in location $i$

- $\text{left}_\text{child}(x) \rightarrow 2i$
- $\text{right}_\text{child}(x) \rightarrow 2i + 1$
- $\text{parent}(x) \rightarrow \lfloor i/2 \rfloor$
The Heapify operation (example)

“Fixes” a tree that is a heap except possibly at the root

(root filters down to its correct position)
Heapify (the algorithm)

Heapify(x) -- assume the tree rooted at x is a heap
-- except possibly at the root
-- and turn it into a heap

if x is not a leaf then
  let y be the child of x with maximum key
  if key(x) ≤ key(y) then
    swap keys of x and y
    Heapify(y)

Maximum number of comparisons for Heapify(x)?

2 times the height of x in the tree.
Analysis of heapify

(to be filled in later)
The Build-Heap operation (example)

1. Place elements arbitrarily in tree (array)
2. Build-Heap: Heapify($x$) as $x$ runs through all nodes, bottom to top, right to left (last to first in array)
Build-Heap (the algorithm)

Array implementation:

\[
\text{for } i \leftarrow n \text{ downto } 1 \text{ do} \\
\quad \text{Heapify}(i)
\]

Total number of comparisons:
(to be filled in later)
Analysis of Build-Heap

A better upper bound:
(to be filled in later, based on analysis on page 145
Application: Heapsort

1. Build-Heap
2. \textbf{while} heap is not empty \textbf{do}
   \begin{itemize}
   \item remove max element (root)
   \item move last element to root
   \item Heapify(root)
   \end{itemize}
Application: Priority queue

Abstract Data Type

Objects: Records with keys from a totally ordered set

Operations:

- **MAX:** returns record with maximum key
- **EXTRACT-MAX:** returns and deletes record with maximum key
- **INSERT:** adds a new record

Implement using a heap

- implement each operation
- worst-case time (comparisons) for each operation