Tighter Build-Heap Running time

- We build the heap from the bottom-up
- Tree starts small & grows bigger
- The worst case for any `heapify` is moving `i` to bottom. This is height of `i`.

Build-Heap \( H \)

for \( i \leq \lceil \frac{N}{2} \rceil \) down-to

\[ \text{do heapify}(H, i) \]

\( O(N \log N) \)  

\( O(N) \)  

Tight bound
The total time is:

\[ S = \sum_{i=1}^{N} \text{height}(i) \]

(i is the index)

Assume a maximum tree of height \( h = 2^{h+1} - 1 \)

- 1 node at height \( h=0 \)
- 2 nodes at height \( h=1 \)
- 4 nodes at height \( h=2 \)
- \( 2^i \) nodes at height \( h=i \)

\[ S = \sum_{i=0}^{h} (h-i) \cdot 2^i \]

\[ S = h \cdot 2^{h+1} + 4(2^{h-2}) + \cdots + 2 \cdot 2^1 + 2^0 \]

\[ 2S = 2h \cdot 2^{h+1} + 4 \cdot 2^{h-1} + \cdots + 2 \cdot 2^1 + 2^0 \]

\[ \Rightarrow S = h^2 + 4 + 8 + \cdots + 2^{h-1} + 2^h \]
\[= -h + \frac{2}{1-2} (1-2^h)\]

\[= (2-1)2^{-h} \leq 2^{h+1} \geq N\]

\[= O(N)\]
Other operations:
- **Increasekey** \((H, i)\), \(i\) is an index
  - Heapify \((H, i)\)
- **Decreasekey** \((H, i)\)
  - insertion

Diagram:
- Node with index \(i\) in the heap array
- Deletion:
  - Last element of the heap
  - Heapify \((H, i)\)
* Heap Sort

\[
\text{HeapSort}(H) \\
\begin{align*}
&\{ \text{Build-Heap}(H) \} \\
&\text{for } (i \leftarrow \text{length}(H) \text{ down to } 2) \\
&\quad \text{do swap } H[i] \leftrightarrow H[?] \\
&\quad \text{heapsize}(H) -= \\
&\text{heapify}(H, 1)
\end{align*}
\]

\(O(N)\) \(\rightarrow\) \(O(N \log N)\)

* Selection algorithm: find the \(k\)th smallest element

Extract \(\text{MIN}\) for \(k\) times \(O(K \log N)\)