Building a heap via divide & conquer:

Given an (unsorted) array $A$ of $n$ #s:

$\text{Build-heap}(A)$:

- Let $h$ be the smallest $h$ s.t. $n \leq 2^h - 1$ (Think: $n = 2^h - 1$)

- Divide:

  - $n_0 = 1$
  - $n_2 = 2^{h-1} - 1$
  - $n_1 = n - n_2 - 1$

  $n_0 + n_0 + n_2 = n$

  $n_1 \geq n_2$

- Combine:

  

  

  \[ T(n_0) + T(n_2) + \max\text{-heapify}(T) \]

  \[ \text{inplace division & combine} \]

  \[ + \max\text{-heapify} \]

  \[ \text{running time} \]

  \[ T(n) = T(n_1) + T(n_2) + O(\log n) \]

  \[ n_1 = \frac{n - n_2 - 1}{2} \text{ (perfect split)} \]

  \[ \text{approximates } T(n) \approx 2T(\frac{n}{2}) + O(\log n) \]

  \[ \Rightarrow T(n) = \Theta(n) \]

  \[ \text{master theorem} \]

  \[ \text{to get an } \Theta(n) \text{ (instead of } O(n \log n) \text{)} \]

  \[ \text{running time is crucial to make this } O(n^{1.5})! \]
Analysis of iterative Build-Heap:

**Key observation:** Max-heapify actually runs in $O(h^2)$ time, where $h$ = height of the subtree it is run on.

$\Rightarrow$ If this subtree has height $\ll \Theta(\log n)$, use again!

**Fact:** In a nearly complete binary tree with $m$ nodes, for any $h \geq 0$,
there is $\leq \frac{m}{2^h}$ nodes $u$ s.t. subtree rooted at $u$ has height $\geq h$.

$\Rightarrow$ Total time to execute all max-heapify calls:

$\leq \sum_{h=0}^{\log m} O(h^2) \cdot \frac{m}{2^h} = m \cdot \sum_{h=0}^{\log m} \frac{O(h^2)}{2^h} = n \cdot \Theta(1) = \Theta(n)$

Note: $\sum_{h=0}^{\log m} \frac{h^2}{2^h} \leq \sum_{h=0}^{\infty} \frac{h+1}{2^h} = \text{Const}$