(Supplementary notes)

Building a heap via divide & conquer:

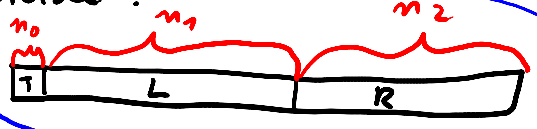
Given an (unsorted) array A of n #s:

Build_heap(A):

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

(Size of a complete binary tree of height h)

→ Divide:



$n_0 = 1$

$n_2 = 2^{h-1} - 1$

$n_1 = n - n_2 - 1$

$\Rightarrow$ $n_0 + n_1 + n_2 = n$, $n_1 \geq n_2$

→ Conquer:

$L' = \text{Build\_heap}(L)$ $T(n_1)$

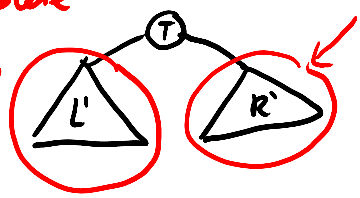
$R' = \text{Build\_heap}(R)$ $T(n_2)$

IMPORTANT: To make divide & combine steps be $\Theta(\log n)$ time (instead of $\Theta(n)$), one has to use an in-place division along the heap subtree structure, instead of the simple split shown here

→ Combine:

nearly complete binary tree of height $h-2 \leq \leq h-1$

Complete binary tree of height $h-2$



+ max-heapify(T)

in-place division & combine + max_heapify

Running time? $T(n) = T(n_1) + T(n_2) + \Theta(\log n)$

Assume that: $n_1 = n_2 = \frac{n-1}{2}$ (perfect split) ← not really needed

To get an $\Theta(n)$ (instead of $\Theta(n \log n)$) time it is crucial to make this $O(n^{1-\varepsilon})$!

By Master Theorem:

$$T(n) \approx 2T(\tfrac{n}{2}) + \Theta(\log n)$$

$$\Rightarrow \boxed{T(n) = \Theta(n)}$$

Analysis of iterative Build_Heap:

Key observation: Max_heapify actually runs in $O(h+1)$ time, where h = height of the subtree it is run on

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

Fact: In a nearly complete binary tree with n nodes, for any $h \geq 0$, there is $\leq \frac{n}{2^{h+1}}$ nodes v s.t. subtree rooted at v has height $\geq h$

$\Rightarrow$ Total time to execute all max_heapify calls:

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

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