Heapsort($A$)

Buildheap($A$) $\mathcal{O}(n)$

for $i = n$ downto 1 $n$

output $A[1]$ (i.e. output the root) $\Theta(1)$

$A[1] = A[i]$ (i.e. put the last leaf at the root) $\Theta(1)$

Heapify($A[1]$) (i.e. heapify the root) $\Theta(lg\ n)$

Analysis: $T(n) = \mathcal{O}(n) + \Theta(n\ lg\ n) = \Theta(n\ lg\ n)$
Analysis of Buildheap:
So, each node’s height is at most \( \lg n \), and there are \( n \) nodes. Since Heapify takes \( \Theta(h) \) time to process a node at height \( h \), the total Buildheap running time is \( O(n \lg n) \)

Is that the best bound we can get?
No.
Tree contains \( n \) nodes, how many are at height \( h \)?

homework: to show that it is at most \[
\left\lfloor \frac{n}{2^{h+1}} \right\rfloor
\]

Since Heapify takes \( \Theta(h) \) time to process node at height \( h \), total time for Buildheap is

\[
\sum_{h=0}^{\lfloor \lg n \rfloor} \Theta(h) = \Theta\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \leq O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right)
\]

From Chapter 3, the summation can be simplified to \[
\frac{1/2}{(1 - 1/2)^2} = 2
\]

And so, the total time becomes \( O(2n) = O(n) \)
Buildheap(A): given an array A, make a heap out of it

for h=0 to height(root)
    heapify all nodes at height h

Analysis of Buildheap: what is the height of the root of a heap?

<table>
<thead>
<tr>
<th>depth</th>
<th># nodes at this depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>$2^d$</td>
</tr>
</tbody>
</table>

$1+2+4+...+2^d = n$, and we want to compute $d$

$$\sum_{i=0}^{d} 2^i = n$$, which becomes $2^{d+1} - 1 = n$

Thus, $d = \log (n+1) - 1 \leq \log n$

In other words, the depth of any node is at most $\log n$. Then, the height of any node is also at most $\log n$. 
Analysis of Heapify: if node $i$ is at height $h$

largest = $\max\{i, \text{left}(i), \text{right}(i)\}$  \(\Theta(1)\)

if largest = $i$ then done  \(\Theta(1)\)

else if largest = left($i$) then switch i & left($i$)  \(\Theta(1)\)

\[\text{Heapify(left($i$))} \quad T(h-1)\]

else switch i & right($i$)  \(\Theta(1)\)

\[\text{Heapify(right($i$))} \quad T(h-1)\]

\[T(h) \leq \max\{ \Theta(1), T(h-1) + \Theta(1), T(h-1) + \Theta(1) \} + \Theta(1)\]

\[= T(h-1) + \Theta(1) = \Theta(h) \quad \text{using iteration method}\]

Thus, Heapify runs in time $\Theta(h)$
Heapify(A, i): given a tree that is a heap except possibly at position i, make the entire tree a heap

largest = max \{ i, left(i), right(i) \}

if largest = i then done
else if largest = left(i) then switch i with left(i) and Heapify(left(i))
else switch i with right(i) and Heapify(right(i))
Binary Heaps

described as a binary tree with heap property

Parent(i): return \( \lfloor i/2 \rfloor \)
Left-Child(i): return 2i
Right-Child(i): return 2i+1

Heap Property: node i has a value that is no greater than the value of i’s parent
Lecture 4: Heapsort

Binary Tree

height of a node: number of edges on the longest path from the node to a leaf (shown to the side of nodes)

depth of a node: number of edges on the path from node to root

leaves are double circled