6.3 Build-Heap

We can create a heap from an unordered array by calling Heapify on the internal nodes starting at the bottom and working our way up.

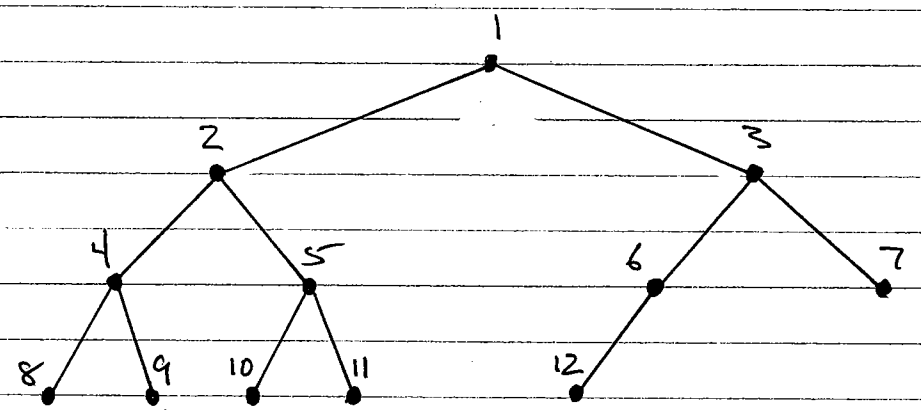
Let $n = heap\text{-}size[A] = length[A]$ and observe that $\lfloor \frac{n}{2} \rfloor$ is the index of the last (i.e. rightmost) internal node, since it is the parent of the last leaf.

Thus the elements of $A[(\lfloor n/2 \rfloor + 1) \ldots n]$ are all leaves and hence each is already a heap. We process the internal nodes $A[1 \ldots \lfloor n/2 \rfloor]$ in an order which guarantees that the subtree rooted at each child is already a heap.

Build-Heap(A)

1.) $n \leftarrow heap\text{-}size[A] \leftarrow length[A]$
2.) for $i \leftarrow \lfloor n/2 \rfloor$ down to 1
3.) Heapify(A, i)

Ex.



$A_1 \ldots\ldots A_6$ $A_7 \ldots\ldots A_{12}$

internal nodes leaves

THE RUN TIME OF BUILD-HEAP is $O(n \lg n)$ SINCE EACH CALL TO Heapify COSTS $O(\lg n)$ AND THERE ARE $\lfloor n/2 \rfloor = \Theta(n)$ SUCH CALLS.

THIS BOUND is NOT TIGHT HOWEVER. OBSERVE THAT THE COST OF CALLING Heapify ON A NODE OF HEIGHT h is $\Theta(h)$, AND THE HEIGHTS OF MOST NODES ARE SMALL.

LEMMA (PROBLEM 6.3-3, P. 135)
AN ACRT ON n NODES HAS AT MOST $\lceil n/2^{h+1} \rceil$ NODES AT HEIGHT h.

PROOF
RECALL THE LAST (i.e. RIGHTMOST) INTERNAL NODE is THE PARENT OF THE LAST LEAF, AND HENCE HAS INDEX $\lfloor n/2 \rfloor$.

$$\underbrace{A_1, \ldots, A_{\lfloor n/2 \rfloor}}_{\text{INTERNAL NODES}} \quad \underbrace{A_{\lfloor n/2 \rfloor + 1}, \ldots, A_n}_{\text{LEAVES}}$$

THEREFORE THE NUMBER OF LEAVES (NODES AT HEIGHT 0) is

$$n - \lfloor n/2 \rfloor$$

IF WE DELETE THESE LEAVES WE OBTAIN AN ACRT ON $\lfloor n/2 \rfloor$ NODES WHICH THEREFORE

HAS $\lfloor n/2 \rfloor - \lfloor \lfloor n/2 \rfloor / 2 \rfloor = \lfloor n/2 \rfloor - \lfloor n/2^2 \rfloor$ LEAVES. EACH OF THESE LEAVES WAS AT HEIGHT 1 IN THE ORIGINAL ACBT. THEREFORE AN ACBT ON n NODES HAS EXACTLY

$$\lfloor n/2 \rfloor - \lfloor n/2^2 \rfloor$$

NODES AT HEIGHT 1. NOW DELETE ALL LEAVES IN THIS NEW TREE TO OBTAIN AN ACBT ON $\lfloor n/2^2 \rfloor$ NODES WHICH THEREFORE HAS $\lfloor n/2^2 \rfloor - \lfloor \lfloor n/2^2 \rfloor / 2 \rfloor = \lfloor n/2^2 \rfloor - \lfloor n/2^3 \rfloor$ LEAVES, WHICH ARE PRECISELY THOSE NODES AT HEIGHT 2 IN THE ORIGINAL TREE.

CONTINUING IN THIS MANNER WE SEE THAT AN ACBT ON n NODES HAS EXACTLY

$$\lfloor n/2^h \rfloor - \lfloor n/2^{h+1} \rfloor$$

NODES AT HEIGHT h. (THIS PART OF THE PROOF COULD BE PHRASED AS A FINITE INDUCTION ON h FOR $h = 0, 1, \ldots, \lfloor \lg n \rfloor$.)

EXERCISE
SHOW $\forall x \in \mathbb{R}: \lfloor x \rfloor \leq \lfloor x/2 \rfloor + \lceil x/2 \rceil \leq \lceil x \rceil$.

IN PARTICULAR $\lfloor x \rfloor - \lfloor x/2 \rfloor \leq \lceil x/2 \rceil$, SO LETTING $x = n/2^h$ WE SEE THAT IN AN ACBT ON n NODES:

$$(\#\text{NODES AT HEIGHT } h) = \lfloor n/2^h \rfloor - \lfloor n/2^{h+1} \rfloor \leq \lceil n/2^{h+1} \rceil. \quad ///$$

LET $T(n)$ DENOTE THE WORST CASE RUN TIME OF BUILD-HEAP ON AN ARRAY OF LENGTH n. BUILD-HEAP CALLS HEAPIFY ON THE LAST INTERNAL NODE ($h=1$), THEN BACKS THROUGH THE ARRAY TO THE ROOT ($h = \lfloor \lg n \rfloor$). EACH CALL TO HEAPIFY ON A NODE AT HEIGHT h COSTS $\Theta(h)$, AND THERE ARE AT MOST $\lceil n/2^{h+1} \rceil$ SUCH NODES. THEREFORE

$$T(n) \leq \sum_{h=1}^{\lfloor \lg n \rfloor} \lceil n/2^{h+1} \rceil \cdot \Theta(h)$$

NOW $\Theta(h)$ STANDS FOR A FUNCTION WHICH IS BOUNDED ABOVE BY ch FOR LARGE h, AND $\lceil x \rceil \leq 2x$ FOR $x \geq 1$. THUS

$$T(n) \leq \sum_{h=1}^{\lfloor \lg n \rfloor} \frac{n}{2^h} \cdot ch \leq cn \sum_{h=1}^{\infty} h \cdot \left(\frac{1}{2}\right)^h$$

FORMULA A.8 ON P. 1061 SAYS:

$$\sum_{k=1}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

EXERCISE: PROVE THIS.

THUS $T(n) \leq cn \cdot \dfrac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2cn = O(n)$.

$\therefore T(n) = O(n)$.

EXERCISE: SHOW $T(n) = \Omega(n)$, WHENCE $T(n) = \Theta(n)$.

6.4 HEAPSORT

HEAPSORT TAKES AS INPUT AN UNORDERED ARRAY A. FIRST, IT CALLS BUILD-HEAP(A) TO ORDER A AS A HEAP. IT THEN EXCHANGES THE LARGEST ELEMENT A_1 WITH A_n (WHERE $n = length[A]$), THEN EXCLUDES A_n FROM THE HEAP BY DECREMENTING heap-size[A] TO $n-1$.

AT THIS POINT THE SUBARRAY $A[1 \ldots (n-1)]$ MAY NOT BE A HEAP, BUT THE LEFT AND RIGHT SUBTREES ROOTED AT A_2 AND A_3 ARE HEAPS (SINCE THEY WERE BEFORE THE SWAP.) THUS A SINGLE CALL Heapify(A, 1) MAKES $A[1 \ldots (n-1)]$ INTO A HEAP.

NOW EXCHANGE $A_1 \leftrightarrow A_{n-1}$, DECREMENT heap-size[A] TO $n-2$, CALL Heapify(A, 1), AND REPEAT THE PROCESS UNTIL heap-size[A] IS 1.

Heapsort(A)

1.) Build-Heap(A)
2.) for $i \leftarrow length[A]$ DOWN TO 2
3.) $\quad A[1] \leftrightarrow A[i]$
4.) $\quad heap\text{-}size[A] \leftarrow (heap\text{-}size[A] - 1)$
5.) $\quad$ Heapify(A, 1)