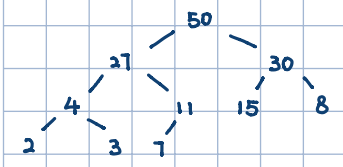
Heaps & Heapsort



In place sorting algorithm: Does not require more memory other than the array that contains the #s

Properties of a heap:

It's a binary tree with:

a). Heap shape property

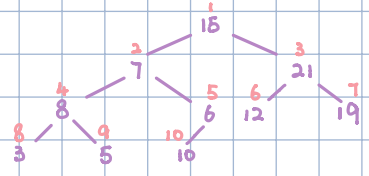
b). Heap ordering: key parent > key child

Do not confuse with BST!

For node i → parent is $(i/2)$
→ right child is $2i+1$
→ left child is $2i$

Example:

	1	2	3	4	5	6	7	8	9	10
A	15	7	21	8	6	12	19	3	5	10



Heap Operations

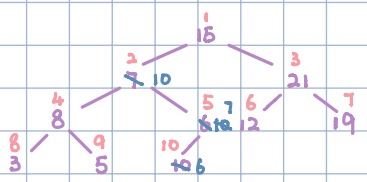
Heapify (or "bubble down")

Heapify(A, i)

Compare A(i) with A(2i), A(2i+1)

swap with larger

repeat till ∄ swap



Heapify(5): compare 10 with 6 and swap them

Heapify(4): compare 8 with 3 & 5 and do nothing

Heapify(2): compare 7 with 10 and swap

compare 7 with 6 and do nothing

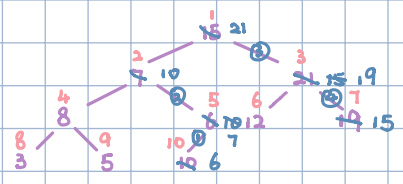
Build Heap(A)

For $i = \frac{\text{length}(A)}{2} \to 1$

Heapify(A, i)

Example:

	1	2	3	4	5	6	7	8	9	10
A	21	10	19	8	7	12	15	3	5	6



Where is the max element in a heap?

Max Heap: at the root

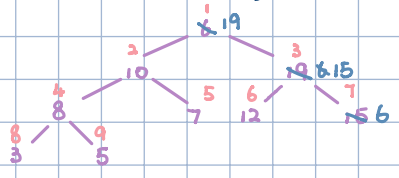
(Min Heap: Smallest at root)

Delete Max(A)

swap(A(1) ↔ A(length(A))

length(A) = length(A) − 1

Heapify(A, 1)

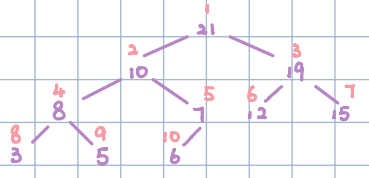


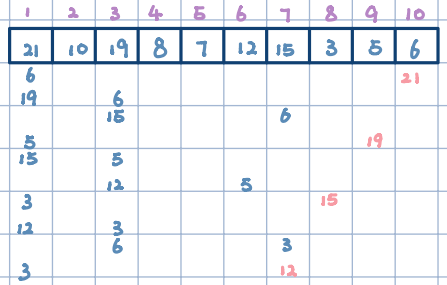
Heapsort(A)

BuildHeap(A)

For $i = 1 \cdots$ length(A)

Delete Max(A)





Runtime:

Heapify: $O(\log n)$

Buid Heap: $O(n \log n)$

Delete Max: $O(\log^n)$

Heapsort: $O(n \log n)$

Tight bound $O(n)$ for Build Heap

Lemma: There are at most $\frac{n}{2^{h+1}}$ nodes of height h in a heap

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

But $\sum_{h=0}^{\infty} h x^h = \frac{x}{(1-x)^2} \overset{x=\frac{1}{2}}{=} \frac{1/2}{(1-\frac{1}{2})^2} = 2$

⇒ Rewrite (*):

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

Priority Queues

Can be implemented easily with heaps.

- Insert (bubble up)
- Extract_Max