Heaps

A heap is a completely filled binary tree (ok, maybe not full leaves), leaves filled left to right.

- Max-Heap
  For all nodes, value of parent ≥ value of self

- Min-Heap
  For all nodes, value of parent ≤ value of self
How long to find $1$ in the heap?
worst-case search: $n$
worst-case height: $\log n$

What is easy to find in a max-heap?
\[ \text{max is } O(1) \]
Insert into a heap

due to the structure of a heap, you always know the shape a heap will have after it goes from n to n+1 elements.

example:
Insert into a heap

- so, to insert a node into a heap,

1. add new node to leftmost open position in leaves

2. adjust as needed to ensure heap property is met.

   a. compare new node to parent.
   b. swap if new node bigger.
   c. repeat until swap not needed or new node is root.
Example:

Insert 11.

Step 1.

11 > 3
Step 2b.

11 > 9
Step 2b again.
Step 2: called "bubbling up" the new node.

Insert runtime:
- worst case: bubble new node up all the way to root

$O(\log n)$

length of path from new node to root
Create a heap from an array of elements (Method I):

- one way is to insert $n$ elements into the heap one after the other

$n$ calls to insert.

Upper bound $O(n \log n)$
Delete Root.

We know the shape of a heap with \( n-1 \) elements.
Delete Root

1. Swap root with rightmost leaf
2. Delete old root
3. Adjust to maintain heap property.

4. Let \( n = \text{new root}, \ i = \emptyset \)
5. Let \( \text{largest} = \max(n, \ n.\text{left}, \ n.\text{right}) \)

6. If \( \text{largest} > n \) {
   7. \( \text{swap}(\text{max}, \ n) \) // \( n \) now on level \( i+1 \)
   }
8. Repeat until no more swaps
    or \( n \) is a leaf
Step 3 called "Bubbling Down"

Step 3b is two comparisons.
worst case - bubble down all the way from root to leaf.
max path length $O(\log n)$
so runtime is $O(\log n)$
Another name for "Bubble Down" is Max-HEAPIFY.

1. Assume children of root are both heaps
2. Bubble down root if needed so whole data structure maintains heap property.
Heap Sort (Array $A[1...n]$) \{
    1. Turn $A$ into a max-heap.
    2. Extract the Max using Delete Root.
    3. Repeat step 2 until only one element left in heap. That element is min.
\}\.

Steps 2 and 3 called "harvesting the heap"
Runtime of Heapsort.

Step 1: Build the heap.

We know upper bound is \( n \log n \).

Is there a way to build in place?

Treat away as a heap:

\[ \text{AC[1]} \quad \text{AC[2]} \quad \text{AC[3]} \quad \text{AC[4]} \]

\[ \text{AC[0]} \quad \text{AC[2]} \quad \text{AC[3]} \quad \text{AC[4]} \]
Storing a tree in an array...

Index of left child: $2 \times$ index of parent

Index of right child: $(2 \times$ index of parent $)+1$

(one-based indexing)
BUILD-MAX-HEAP

1. Treat array like a candidate heap.
2. All leaves are heaps, so start at level before leaves. Go from left to right.
   Calling MAX-HEAPIFY on each node until you finish calling MAX-HEAPIFY on root.
3. Repeat on all higher levels.

So you call MAX-HEAPIFY around \( \frac{n}{2} \) times.
Example of Build-Max-Heap:

3 6 2 1 7 9 8

Step 1:

Step 2:

Step 2 again:

bubble down 6 and 2

bubble down 3
Steps 2 and 3: Call `DeleteRoot` repeatedly (harvest heap)

Is there a way to harvest heap in place?
Yes.

Let \( \text{last} = n \)

Put max in \( A[\text{last}] \).
Treat heap as \( A[1 \ldots (\text{last}-1)] \)
\( \text{last}--; \)
Repeat.
Runtime of Heapsort?

1. Build heap in place.
2. Harvest heap in place.

Worst case of Step 1: \( \Theta(n \log n) \).

But note, when at low levels, performance of \texttt{MAX-HEAPIFY} is not \( O(\log n) \), because height of subtree rooted at lower level nodes \( < \log n \).
Runtime of Build-MAX-HEAP is $\sum_{h=0}^{\log_2 n} (\text{# nodes with height } h) \in O(n)$.
\[ \log_2 n \leq a \sum_{h=0}^{n} \left\lfloor \frac{n}{2^{h+1}} \right\rfloor (ah) \]

( Ignore ceiling )

\[ = \frac{1}{2} an \sum_{h=0}^{\infty} h \left( \frac{1}{2} \right)^h \]

\[ \leq \frac{1}{2} an \sum_{h=0}^{\infty} h \left( \frac{1}{2} \right)^h = \frac{1}{2} an \left[ \frac{\frac{1}{2}}{(1 - \frac{1}{2})^2} \right] \]
\[
\left(1 - \frac{(1-x)^2}{2}\right)(-1) = 1 - x
\]

\[
x \geq 1
\]

For the function \( f(x) = 1 - x \),

\[
x \geq 1
\]
\[ \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \]
Runtime of HeapSort?

① Build heap in place: $O(n)$

② Harvest heap in place:

Call Delete Root $O(n)$ times.
Each call to `Delete Root` starts with a new node at the root which could bubble all the way down to a leaf.
So path length for all Delete Root calls is $\log_2 n$.

So harvesting runtime is $O(n \log n)$.

Runtime of Heapsort?

1. Build heap in place: $O(n)$
2. Harvest heap in place: $O(n \log n)$

**TOTAL RUNTIME**: $O(n \log n)$
Another use of a heap is a priority queue.

Convenient because if you have a multitheaded machine you can return max to one thread that needs it and heapify in another thread.