CSCE 750 Heaps
9/8/2022

Def: A max-priority queue is a data structure that stores a collection of objects from some linearly ordered set (Comparable interface in Java, for example) and supports these operations:

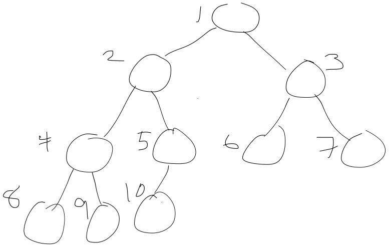
Insert(H, x) — insert an element x into heap H

FindMax(H) — return the element with maximum (key) value

DeleteMax(H) — remove the max element from H

Binary heap implementation

Heap items are stored in an array A[1 .. n]



$\text{parent}(i) = \lfloor \frac{i}{2} \rfloor$

$\text{left}(i) = 2i$

$\text{right}(i) = 2i+1$

Max-heap condition: for every $i > 1$, $A[i] \leq A[\text{parent}(i)]$

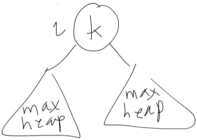
FindMax(H) —
    return A[1]

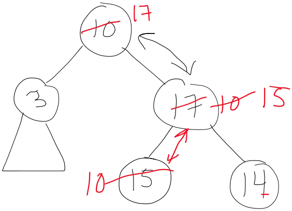
Define BuildMaxHeap
    convert an arbitrarily ordered array A[1 .. n] in max-heap order

calls a subroutine
    MaxHeapify(A, n, i)
        A[1 .. n] is the array
        n = # of elements

$1 \le i \le n$.

Turns the subtree rooted at index i into max-heap order, assuming the two subtrees of i are in max heap order

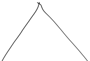


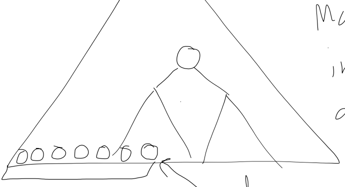


cascading down:
Swap k with the largest of its children until no child of k is > k.

Time for MaxHeapify:
lg n worst case
actually h, where h is the height of the subtree rooted at i
This is $O(\lg n)$

BuildMaxHeap(H)





MaxHeapify in order of descending index, starting at $\lfloor n/2 \rfloor$ until 1.

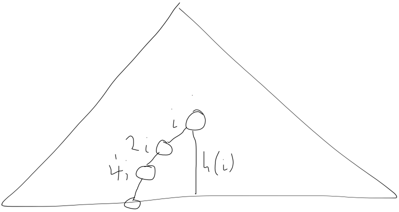
Running time of BuildMaxHeap:

calling MaxHeapify $\frac{n}{2}$ times
O(lg n) worst case

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

But this is not tight

In fact, $T(n) = \Theta(n)$.



Tight analysis: Let $h(i)$ be the height of node i

Have $2^{h(i)} i \le n$

$$2^{h(i)} \le \frac{n}{i}$$

$$\longrightarrow h(i) \le \lg\left(\frac{n}{i}\right)$$

$$T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} \left(\underline{O}(h) \left\lceil \frac{n}{2^{h+1}} \right\rceil \right)$$

$$\le cn \underline{\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}}$$

mixed arith.-geom. sum with ratio $\frac{1}{2}$

$$\le cn \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{\left(1-\frac{1}{2}\right)^2} cn$$

$$= O(n)$$

Obviously, $T(n) = \Omega(n)$

$$\therefore \underset{\underset{\text{time for BuildMaxHeap}}{\parallel}}{T(n)} = \Theta(n).$$

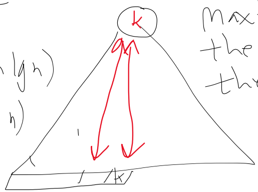
HeapSort(A, n) — sort elements $A[1], \dots, A[n]$ in place

BuildMaxHeap(A, n)

$\Theta(n)$

$+ \Theta(n \lg n)$

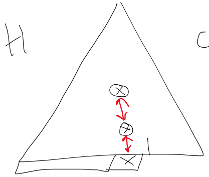
$= \Theta(n \lg n)$



MaxHeapify the root with the (smaller) heap

Priority Queue ops:

Insert(H, x)

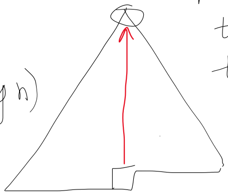


cascading up — swap x with parent until x is the root or x is ≤ its parent

Time (worst-case) $\Theta(\lg n)$

FindMax(H)
  return H[1]   Time $O(1)$

DeleteMax(H)



Time $O(\lg n)$

move H[n] to root, then MaxHeapify at the root.

Define a min-priority queue and a min-heap in the same way by just reversing the sense of the comparisons.

Fibonacci Heap performs better than this.
We won't cover this. Instead, I'll cover Binomial Heaps (also support efficient merging).