Heaps – Implementation

Arrays are the traditional implementation for heaps.

- same big-Oh as linked structure, but avoids space overhead of parent/child pointers

Running time:

- insert – O(log n)
  - O(1) to put element in array, update `last`
  - O(log n) to bubble up
- remove min – O(log n)
  - O(1) to swap with last, remove last, update `last`
  - O(log n) to bubble down
- find min – O(1)
  - min element is at root (index 0)

Heaps – Implementation

We didn't improve the big-Oh over the balanced search tree implementation for PQs.

But –

- reduced storage overhead (no parent, child pointers)
- reduced difficulty of implementation
  - array + bubble up, bubble down vs. linked structure + balanced search tree operations
  - traded maintaining 'min' reference for incrementing/decrementing 'last' index
- reduced constant factors
  - traded O(log n) maintenance of 'min' reference for O(1) maintenance of 'last' index

Building a Heap

How to build a heap?

- repeatedly insert each element $\sum_{i=0}^{n-1} \log(i)$ = Θ(n log n)

Building a Heap

Or...if you already have an array of elements...

- for any n elements in an array, the heap order property is at most broken only for the first n/2 elements

Heapify idea.

- for each index n/2 down to 0, bubble down that element

Running time.

- bubble down takes O(h) time
  - n/2 elements are leaves (already in place – no change)
  - n/4 elements are one level above leaf (at most 1 swap)
  - n/8 elements are two levels above leaf (at most 2 swaps)
  - …
- = $\sum_{i=1}^{\log n} (i-1)(\frac{n}{2^i})$ = n Θ(1) = Θ(n)