CSC263 Tutorial Jan 19, 2018

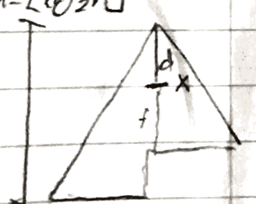
- ADT: Priority Queue P
  - ↳ Operations: INSERT(P, x) EXTRACT MIN/MAX(P)

- Heaps = implementation of Priority Queue
  - ↳ Heap-order property - values of children of node x are (max: less than or equal to) (min: greater than or equal to) value of x

- if there is only one node that violates heap-order property, recursively compare to children & swap with largest child
- max-heapify(A, i) - subprocedure used in INSERT & EXTRACT
  - Precondition: subtrees rooted at Left(i) & Right(i) are heaps
  - Psuedocode of max-heapify in textbook
  - max-heapify $\in O(\log n)$ // at most $\log n$ constant time calls

- Build-max-heap: for $i = \lfloor |A|/2 \rfloor$ to 1 : max-heapify(A, i)
  1. express array as complete binary tree
  2. iterate from leafs to root performing heapify
     - heapify precondition true for only leafs ∴ start heapify on second level of nodes
     - after heapify performed on level, can assume subtrees starting at that level satisfy heap precondition

$h = \lfloor \log_2 n \rfloor$



f = node height

- intuitively (BMH calls $O(\log n)$ procedure n times) in $O(n \log n)$ but does not actually iterate through whole height of tree n times!

O(n):
- each node at depth d has height of at most $h-d$ ∴ "heapification" cost of each node is at most $h-d$

∴ cost to heapify each level $\leq 2^d(h-d)$ // $2^d$ nodes at depth d

∴ cost to heapify whole tree $\leq \sum_{d=0}^{h-1} 2^d(h-d)$ // h-1 because leaves skipped

Set $i = h-d$ ∴ $d = h-i$, write $\sum_{i=1}^{h} 2^{h-i} \cdot i = 2^h \sum_{i=1}^{h} i/2^i \leq 2^h \boxed{\sum_{i=1}^{\infty} i/2^i} = n \boxed{2}$.

- Deleting from Binary Search Trees
  A) x has no children: set parent's pointer to null, removed
  B) x has one child: set parent's pointer to x's child, removed
  C) x has two children:
    - successor: smallest number bigger than x
    - go into right subtree of x, then all the way left to find successor
    - set x value to value of successor, delete successor