Readings Ch.6 (App. A + C)

Self test 6.3-1, 6.1-4, 6.2-4

Lecture 03

Priority Queues

Like a queue except every item has a "priority" (usually a number) that determines retrieval order. More formally, a priority queue consists of a set of elements S, where each element has a priority.

`insert(S,x)`: insert x in the set (priority of x stored in `x.priority`)
`maximum(S)`: return element from S with largest priority
`extractMax(S)`: remove and return element S with largest priority

Applications:

- job scheduling in an operating system
- printer queues
- event-driven simulation algorithms
- etc.

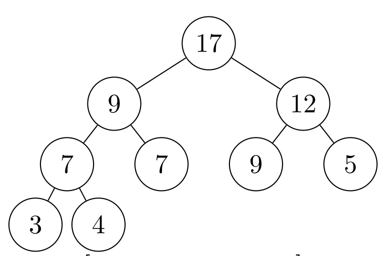
19	5	4	20	-2	0	5	$\rightarrow$	-2	0	4	5	5	19	20

Data structures:

- Unsorted list? $\rightarrow \Theta(n)$ for `extractMax`
- Sorted list (by priorities)? $\rightarrow \Theta(n)$ for `insert`
- Special case: If only a fixed number k priorities $\{p_1, p_2, \ldots, p_k\}$ and k is small, then keep an array with k positions (one for each priority) and store items in a linked list at each position. Then,
  `insert` $\in \Theta(1)$
  `maximum, extractMax` $\in \Theta(k)$
  `increasePriority` $\in \Theta(1)$

Heaps

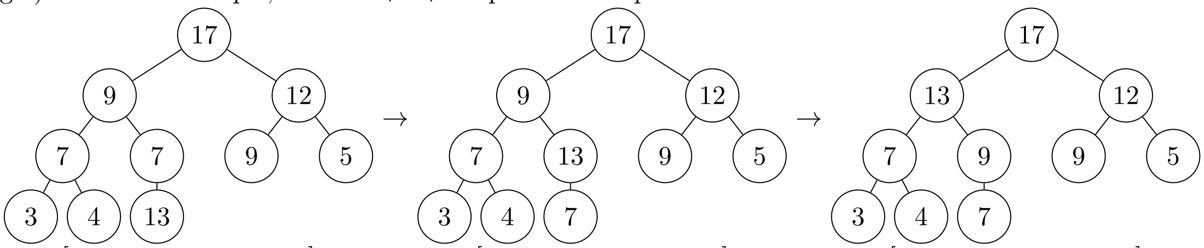
Simple data structures used to represent priority queues. Stores items in *complete* binary trees (each level contains maximum # of nodes, except possibly last level, and nodes in last level "as far left" as possible), and in "heap order": every node has priority greater than or equal to priorities of its immidiate children.
Implication: every subtree of a heap is also a heap.
Complete tree: ensures height is small.
Heap order: supports faster heap operations.
Intuition: tree is partially sorted. Enough to query operations fast while not requiring full sorting after each update.



[17,9,12,7,7,9,5,3,4]

insert

Increment “heapsize”, add element at new index “heapsize”. Result might violate heap property: “bubble” element up (exchange it with its parent) until priority no greater than priority of parent. $\Theta(\text{height}) = \Theta(\log n)$ time. For example, `insert(13)` on previous heap:



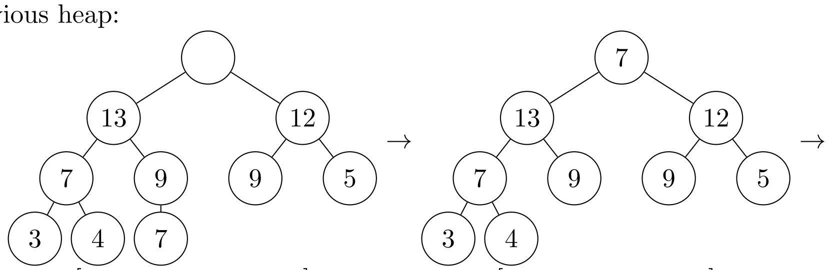
[17,9,12,7,7,9,5,3,4,13] → [17,9,12,7,13,9,5,3,4,7] → [17,13,12,7,9,9,5,3,4,7]

maximum

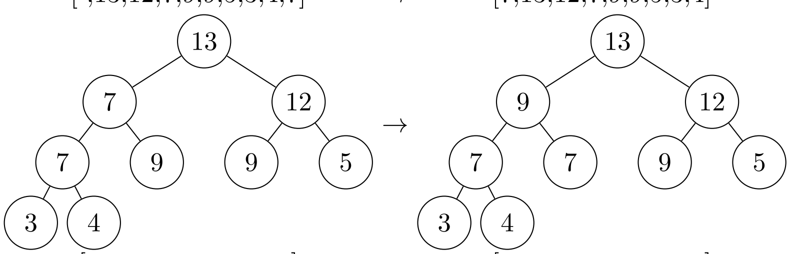
Return element at index 1 (if heapsize ≥ 1). $\Theta(1)$ time.

extractMax

Decrement “heapsize”, remove element at index 1. This leaves “hole” at index 1: move element at “heapsize + 1” into index 1. Restore heap order by “percolating down” (exchange with highest priority child until priority is greater than or equal to both children, or leaf is reached). $\Theta(\log n)$ time. For example, `extractMax` on previous heap:



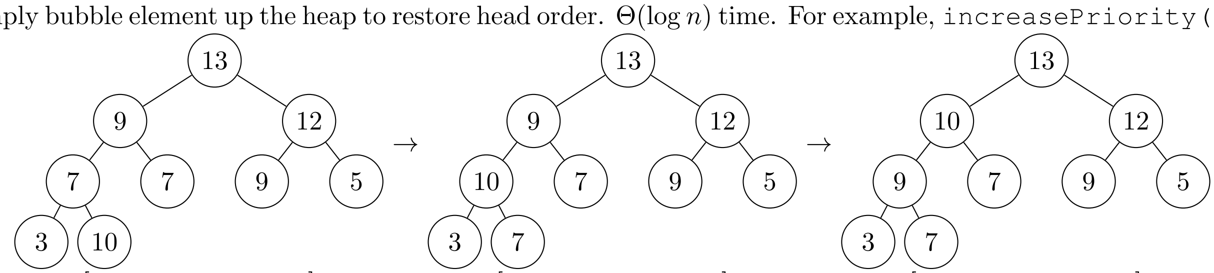
[,13,12,7,9,9,5,3,4,7] → [7,13,12,7,9,9,5,3,4] →



[13,7,12,7,9,9,5,3,4] → [13,9,12,7,7,9,5,3,4]

increasePriority

Simply bubble element up the heap to restore head order. $\Theta(\log n)$ time. For example, `increasePriority(4,10)`:



[13,9,12,7,7,9,5,3,10] → [13,9,12,10,7,9,5,3,7] → [13,10,12,9,7,9,5,3,7]