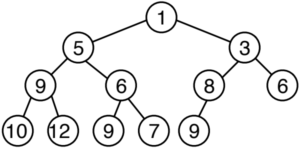
Priority Queues

Priority Queue operations:

- Insert
- Delete Min

Recall basic "heap" data structure

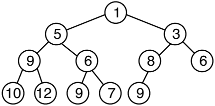


Structure: "nearly" perfect binary tree

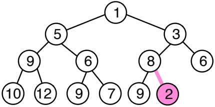
- $n \geq 2^h$, where $n :=$ # nodes, $h :=$ height

Heap condition: $key(v) \geq key(parent(v))$

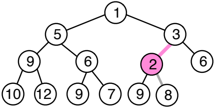
Insert:

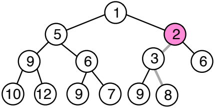


Insert 2

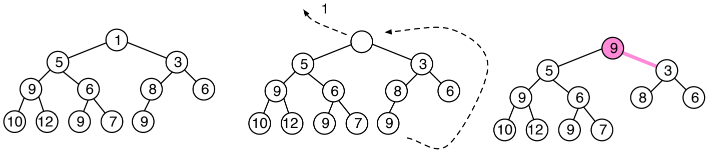


“float up”

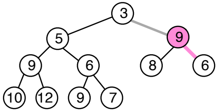


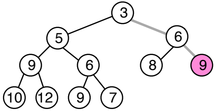


Delete Min:



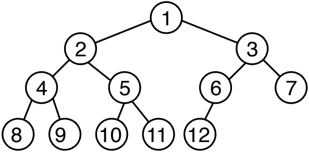
“sink down”





Insert and Delete Min: time $O(\log n)$

Array layout (an optimization)

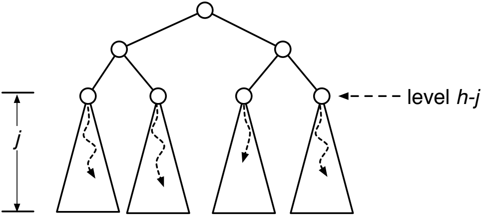


If array is indexed from 1:

- $LeftChild(i) = 2i$
- $RightChild(i) = 2i + 1$
- $parent(i) = \lfloor i/2 \rfloor$

Building a heap from scratch in time $O(n)$

- Put all keys in the array
- Let h be the height of the (implicit) tree
- Process nodes at levels $h-1, h-2, \ldots, 0$:
  - let the key at node v “sink” to its correct position in the subtree rooted at v (as in Delete Min)
- After processing level j, each node at level j is the root of a heap



- Cost for level $h-j$: $O(j2^{h-j})$
  - 2^{h-j} nodes at level $h-j$, each costs time $O(j)$ to process
- Total cost: $O(t)$, where $t = \sum_{j=1}^{h} j2^{h-j}$

- Total cost: $O(t)$, where

$$t = \sum_{j=1}^{h} j 2^{h-j}$$

$$= 2^h \sum_{j=1}^{h} j/2^j \leq n \sum_{j=1}^{h} j/2^j$$

Also, $\sum_{j=1}^{\infty} j/2^j = 2$:

$$\begin{array}{cccc} 1/2 & & & \\ 1/4 & 1/4 & & \\ 1/8 & 1/8 & 1/8 & \\ \vdots & \vdots & \vdots & \\ \hline 1 & 1/2 & 1/4 & \cdots \end{array}$$

$\therefore t \leq 2n$

Application: Heap Sort

- Build heap: cost $= O(n)$
- For $i = 1, \ldots, n$: Delete Min
  - each Delete Min costs $O(\log n)$
  - total cost $= O(n \log n)$
- Total cost $= O(n \log n)$

Mergeable Priority Queues

Operations:

- Insert
- Delete Min
- Merge two queues

Using heaps:

- need to re-build — time $O(n)$

Using 2-3 trees:

- Can support all 3 operations in time $O(\log n)$

Mergeable Priority Queues using 2-3 trees

Same tree structure as ordinary 2-3 trees

Keys stored at leaves, but

- duplicates allowed
- keys not in any particular order

Internal nodes contain “min key values” as guides

Insert: just make a new leaf (anywhere), and update guides

Delete Min: follow guides to find min, delete, and update guides

Merge: use Join procedure, and update guides

Implementation notes for heaps

Instead of keys, each array entry $A[i]$ may point to some object, one of whose fields acts as a key, say $A[i]$.priority

Each object also stores its position in the heap, so $A[i].\text{pos} = i$.

These objects may be accessible through other data structures besides the heap

If p points to such an object, we may modify p.priority directly

- $i = p.\text{pos}$ gives us p's position in the heap
- We can "float" or "sink" p as necessary to maintain heap condition
- All objects whose position in the heap changes must have their pos fields updated as well

All heap operations still take time $O(\log n)$

Similar techniques can also be used for 2-3-tree-based priority queues