# Heaps

The idea: a heap is a
- binary tree +
- an ordering property to aid in searching +
- a structural property to aid in efficiency of implementation

Heap ordering property:  (min heap)
- every node's key is ≤ the keys of its children
  - smallest element is at the root

Structural constraint:
- the tree is a complete binary tree
  - the only empty spots are the rightmost elements in the last level

Note: the height of a complete binary tree is O(log n).



# Heaps

Insert the elements 42, 50, 40, 60, 30, 80 into an initially-empty max heap. Draw the state of the heap after each insertion.

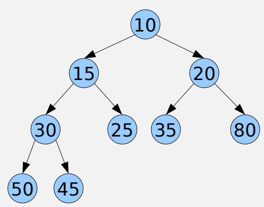
Carry out four "remove max" operations, starting with the heap resulting from the previous question. Draw the state of the heap after each removal.

Strategy –
- insert/remove as dictated by the structural property
- fix up the ordering property if broken by bubbling element down or up



# Heaps – FindMin

The ordering property means that the min element is at the root.

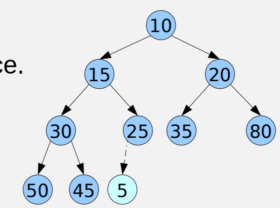





# Heaps – Insertion

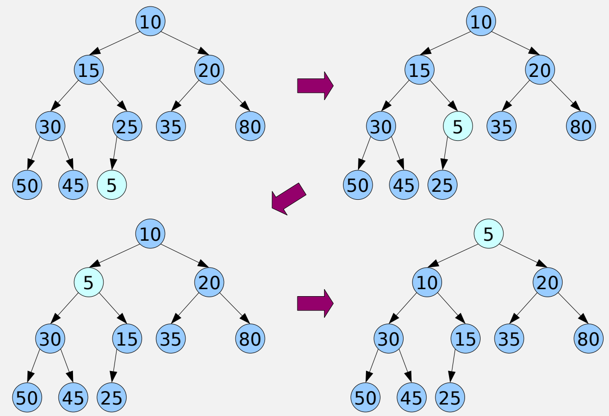
The structural property means insertion can only occur in one place.

Strategy: insert in the only possible place, then fix up the ordering property if broken.




Thus:
- insert element in the first available slot
- "bubble up" until ordering property is restored
  - element is only out of order with respect to parent

## Heaps – Insertion

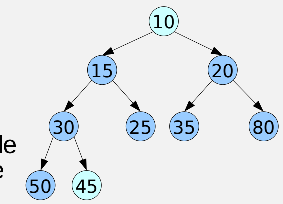





## Heaps – RemoveMin

The structural property means removal can only occur from one place.

Strategy: swap element to delete with the element in the only possible position for removal, then fix up the ordering property if broken.

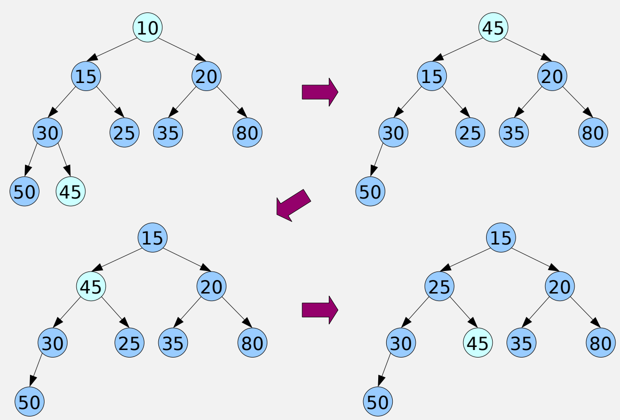



Thus:

- swap root with last slot
- remove element in last slot
- "bubble down" until ordering property is restored
  - swap with smaller child



## Heaps – RemoveMin






## Heaps – Implementation

The standard implementation for binary trees is a linked structure.

  - tree node stores element + pointers to parent, left child, right child

Running time and space –

- find min is O(1) – min element is at the root
- inserting and removing require knowing the location of the last element in the tree
  - O(n) to find – don't know which child will have the last leaf
  - solution – maintain a `last` pointer!
  - updating `last` after insertion/removal can require O(log n) time
    - bubbling is already O(log n) so this is just a constant factor
- space is O(n)
  - but there is overhead of three pointers per element (same as BST)

## Heaps – Implementation

Assessment –

- same big-Oh running time as balanced BST
- space is similar
  - need parent pointers, though many balanced BST implementations have overhead beyond the binary tree structure
- implementation is simpler

Can we do better?

- reduce space overhead
  - array eliminates overhead of pointers...if structural information can be encoded in the indices
- reduce time to build heap from n elements
  - O(n log n) for n insert operations



## Heaps – Implementation

The alternative to a linked structure is an array.

- calculate parent/child index instead of storing
  - root stored at slot 0
  - left child of node with index i is in slot 2i+1, right child in slot 2i+2
  - parent of node with index j is in slot (j-1)/2

Running time and space –

- find min is O(1) – min element is in slot 0
- inserting and removing require knowing the location of the last element in the tree
  - at `size-1` (i.e. maintain a `last` index)
  - updating this value after insertion/removal takes only O(1) time
    - just increment or decrement
- space is O(n)
  - only have to store elements (no additional pointers)
  - complete binary tree fills consecutive slots – no gaps



## 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 BST implementation for PQs.

But –

- reduced storage overhead (no parent, child pointers)
- reduced difficulty of implementation
  - array + bubble up, bubble down vs. linked structure + balanced BST 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)