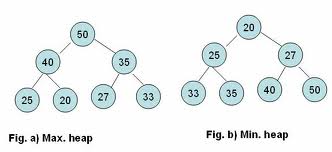
**Topic Outline – Heaps and Priority Queues**

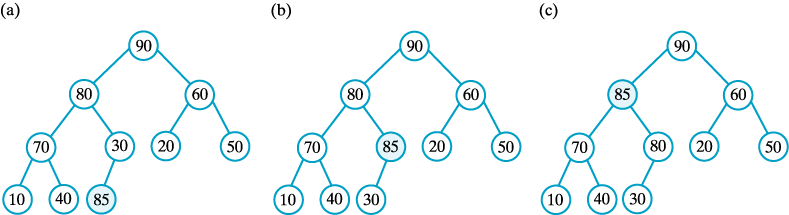
1. Overview of Heaps
   1. Heaps are binary trees with two additional properties
      1. It is a complete tree (filled and balanced – see pictures)
      2. Each node is less than or equal to both the left and right children (min heap)

Or

* + 1. Each node is greater than or equal to both the left and right children (max heap)



1. Adding new nodes to a Heap (We will assume we’re doing a min heap)
   1. \*\*Because a heap is a complete tree, there is only one correct location for the insertion of a new node, and that is either the next open position from the left at level h or the first position on the left at level h + 1 if level h is full.
   2. Once the node is added, we have to maintain the proper ordering (min heap rules).
      1. If the new value is less than the parent node’s value, then swap the values. Continue this process up the tree until the new value either is greater than its parents or is in the root of the heap.
      2. This is illustrated well in your book on page 337 (figure 11.5)
      3. Another example – adding the value of 85 into a max heap

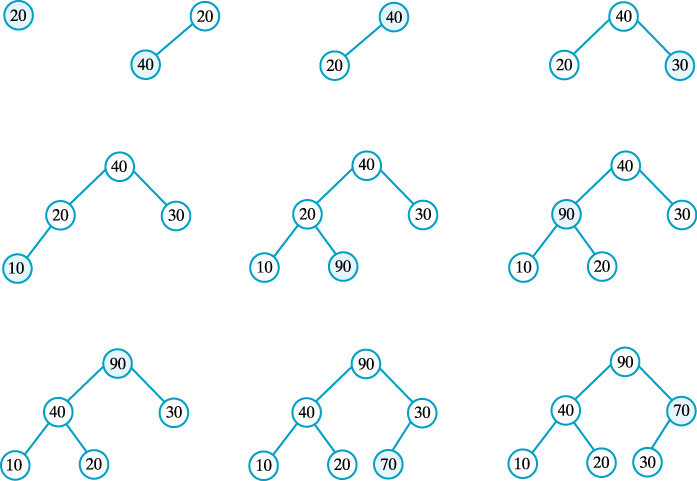


* + 1. *It is important to note that we are swapping only the values contained in the nodes, not the nodes themselves.*

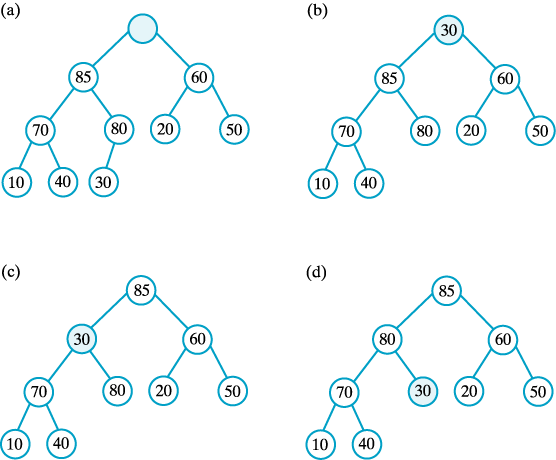
1. Sample Problem - *Create a max heap out of the following values (in the order that they appear). Draw your heap state at each insertion and swap point (yes, that is a lot of heap drawings… I know).*

**Values – 20, 40, 30, 10, 90, 70**

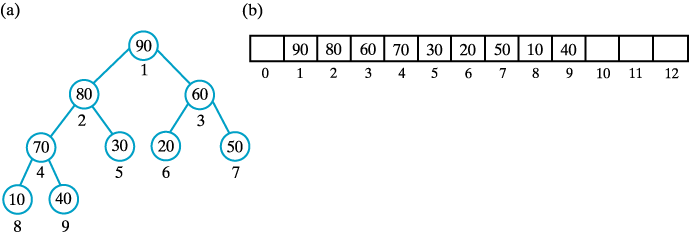
**Solution:**



1. Removing the minimum (min heap) or maximum (max heap) element.
   1. We’re using a heap, which means we care only about the lowest or highest value (otherwise we’d be using a regular binary tree, queue, or stack). So if we need to pop off (remove) a value, it’s going to be the one at the root position since that’s our largest/smallest value.
   2. To remove the root element, replace it with the element that was added last (our heap keeps a reference to this object just like it does for the root element and just like linked lists keep track of the first element in the list), then continually compare the value now at the root with its two children, swapping as needed until the value is either greater (max heap) or less (min heap) than both the children or it is in a leaf position.
   3. Step-by-step illustration of removing the max (root) value of a max heap



1. Implementing Heaps
   1. Links vs. Arrays
   2. Array Representation of a Heap - a complete binary tree with its nodes numbered in level order
      1. Parent of a node at *i* is found at *i/2*   
         (unless *i*  is 1)
      2. Children of node at *i* found at indices   
         *2i* and *2i + 1*



* 1. How do you determine where to add new values to the heap?

1. Priority Queues
   1. Priority Queues have two ordering rules:
      1. Items with higher priority go first (duh!) instead of just using FIFO like regular queues
      2. Useful when new items \*sometimes\* need to be remove/processed before existing ones
      3. Items with the same priority use a first in, first out (FIFO) method to determine ordering.
   2. Min Heaps are commonly used to implement priority queues
2. BREAK
3. Coding a heap in java
4. Programming Assignment #2 overview