QuickSort – one last tine

- like MergeSort is divide and Conquer
  - so like MergeSort is O(n log(n))
    - but analysis is far more complex because much depends on good pivot selection
- “Really worst case” analysis
  - What is the worst possible pivot selection?
  - if you always choose the worst pivot how long does QuickSort take?

Priory Queue -- recall

- A priority queue is a data structure that offers convenient access to the data item with the smallest (or largest) key.
- Implemented with with linked list or ordered array:
  - Search/Deletion O(1)
    - go to the first item
  - Insertion O(n)
- Alternative implementations?

Heaps – a priority implementation

- A heap is a binary tree.
- Insertion and deletion are both log(n).
- It is the method of choice for implementing priority queues when speed is important and there are many insertions.

Heap Characteristics

- A heap is a binary tree with these criteria:
  - It is complete – although the last row need not be full
  - It is (usually) implemented with an array.
  - Each node in a heap satisfies the *heap condition*, which states that every node's key is larger than or equal to the keys of its children.

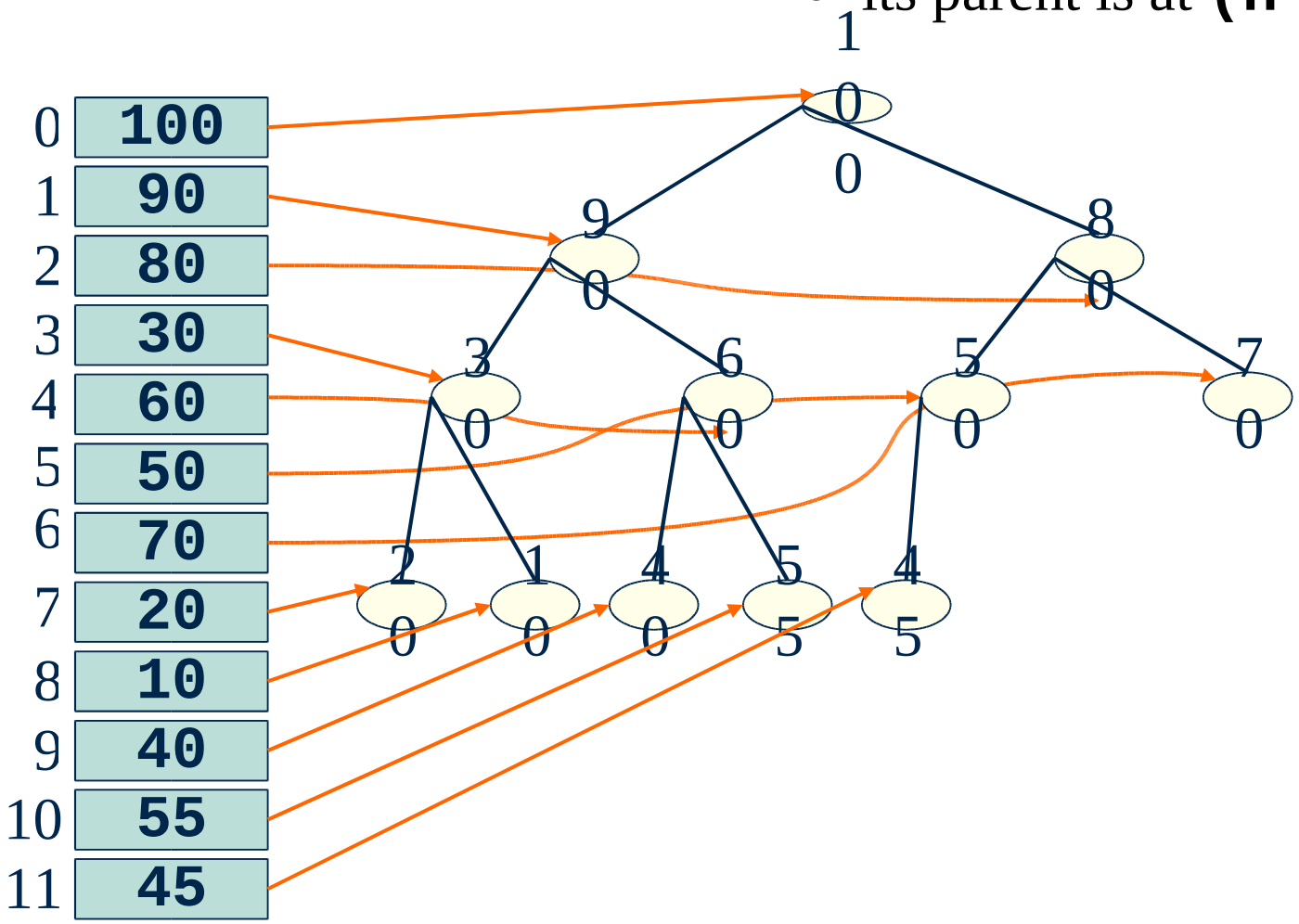
Representing a Binary Tree with Arrays

- So far we used references to find a node's left and right children.
- In the array approach, the nodes are stored in an array, and are not linked.
- The position of the node in the array corresponds to its position in the tree.
- The node at index 0 is the root, index 1 is the root's left child and so on, progressing from left to right for every level of the tree.

Heap as Array

if a node is stored at index **n**

- its left child is at **2*n+1**
- its right child is at **2*n+2**
- its parent is at **(n-1)/2**



Weakly Ordered

- A binary tree has a stronger ordering than the heap, that is, there's ordering among siblings as well as between parents and children.
- Thus traversing heap nodes in order is difficult.
- Also can not search for a specific key.

Heap Ordering and Priority Queue

- Recall that we wanted the heap to simulate the priority queue.
- The weaker ordering of the heap, however, is just enough to allow the quick removal of a maximum node as well as quick insertion.
- The node with the max key is always the root in a heap.

Deletion

- `maxNode = heapArray[0];`
- Now we must fill the hole that is the root.
  - Move the last node in the array to the root

    `heapArray[0] = heapArray[--N];`
  - Trickle (swap) the node down until it's below a larger node and above a smaller one.
    - compare the node with the larger of its children
    - swap if smaller

Insertion

- Insertion calls for trickling up instead of down.
  - Insert the node as the last item in the array.

    `heapArray[N++] = newNode;`
  - Compare node with parent.
  - Swap if bigger.

Heapsort

- Even though it is very difficult to traverse the heap in order, there is a simple way to sort the items using a heap.
- Insert items into the a heap normally.
- Repeated deletion of all items will then remove them in order!

```
for(i=0; i<size; i++)
  theHeap.insert(myArray[i]);
for(i=0; i<size; i++)
  myArrray[i] = theHeap.delete();
```

Heapsort Efficiency

- Insertion and deletion are both log(n).
- They are each applied n times.
- Efficiency of heapsort is therefore 2*n*log (n), which is O(nlog(n)).
- However, even though it is rated the same as quicksort, it is slower because of the trickling that happens in the inner loop.
- Also uses more memory.