

# Heaps & Priority Queues



Computer Science Department  
University of Central Florida

*COP 3502 – Computer Science I*



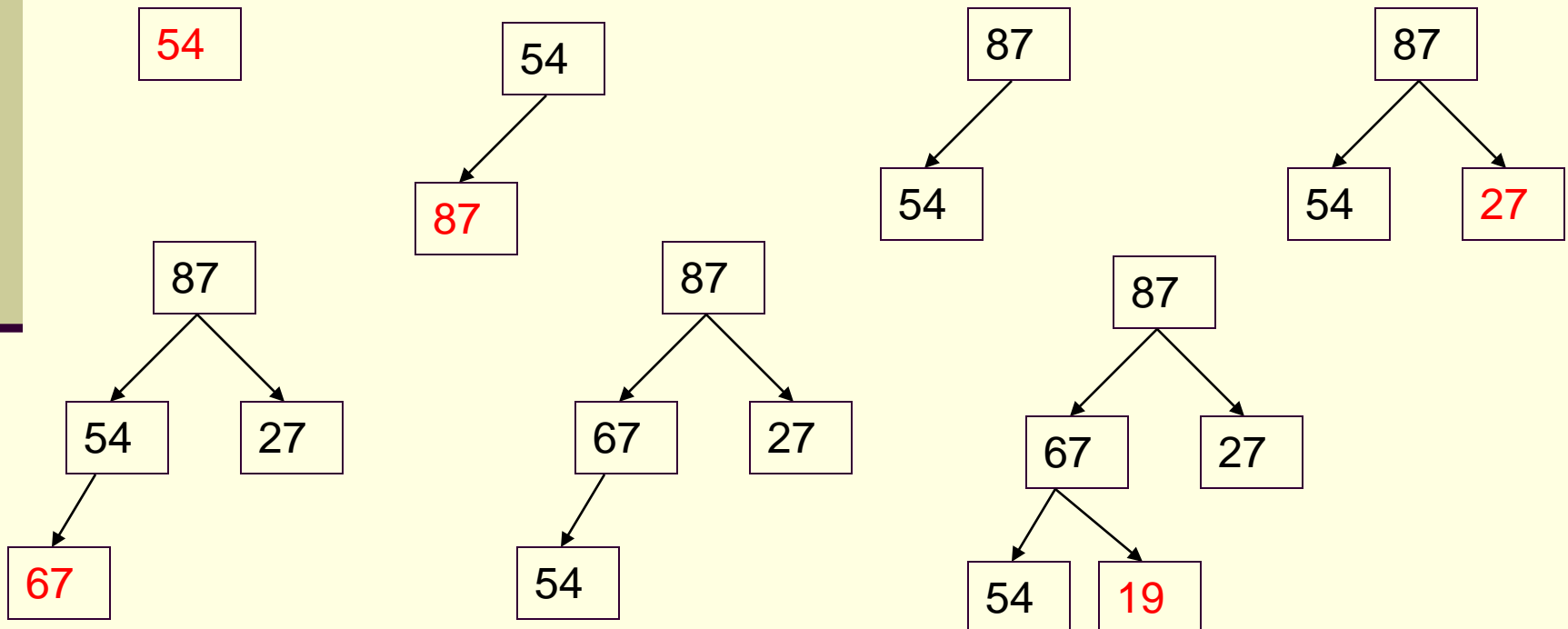
# Binary Heaps

- Building a Heap from scratch (a Max heap)
  - Given: an unsorted list of n values
    - **54, 87, 27, 67, 19, 31, 29, 18, 32, 56, 7, 12, 31**
  - How can we build a heap from these values?
    - It is really just a series of “insertions”
    - Simply insert the n elements into the heap in the order that they arrive (in our case, from left to right)
    - WHILE there are more elements:
      - 1) Insert the next element
      - 2) Percolate Up to a suitable position
  - Once all elements are inserted, we have our heap



# Binary Heaps

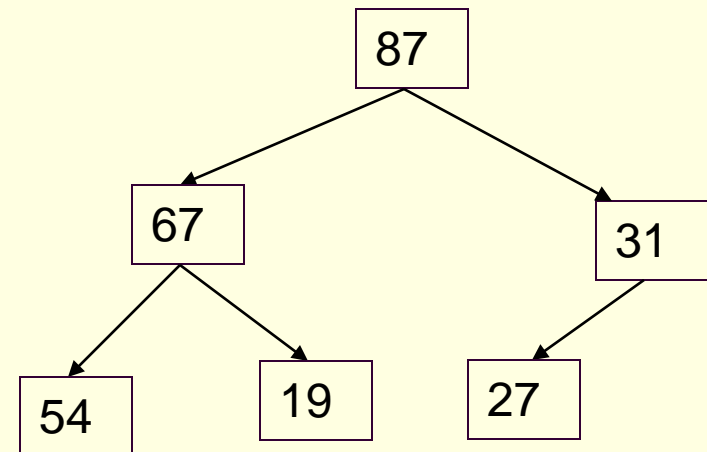
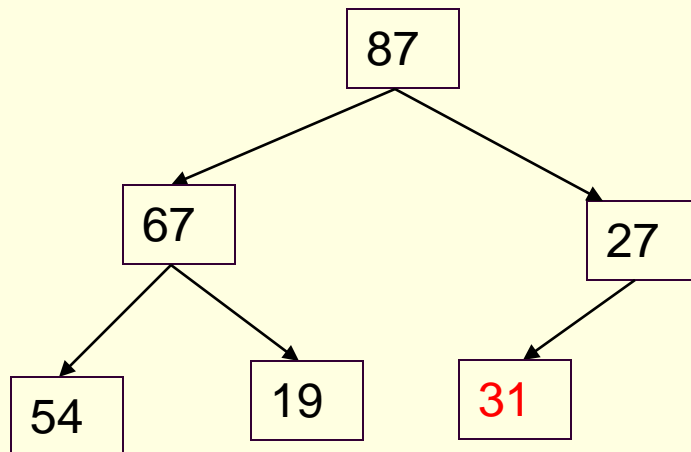
- Building a Heap from scratch (a Max heap)
  - Given: an unsorted list of n values
    - **54, 87, 27, 67, 19, 31, 29, 18, 32, 56, 7, 12, 31**





# Binary Heaps

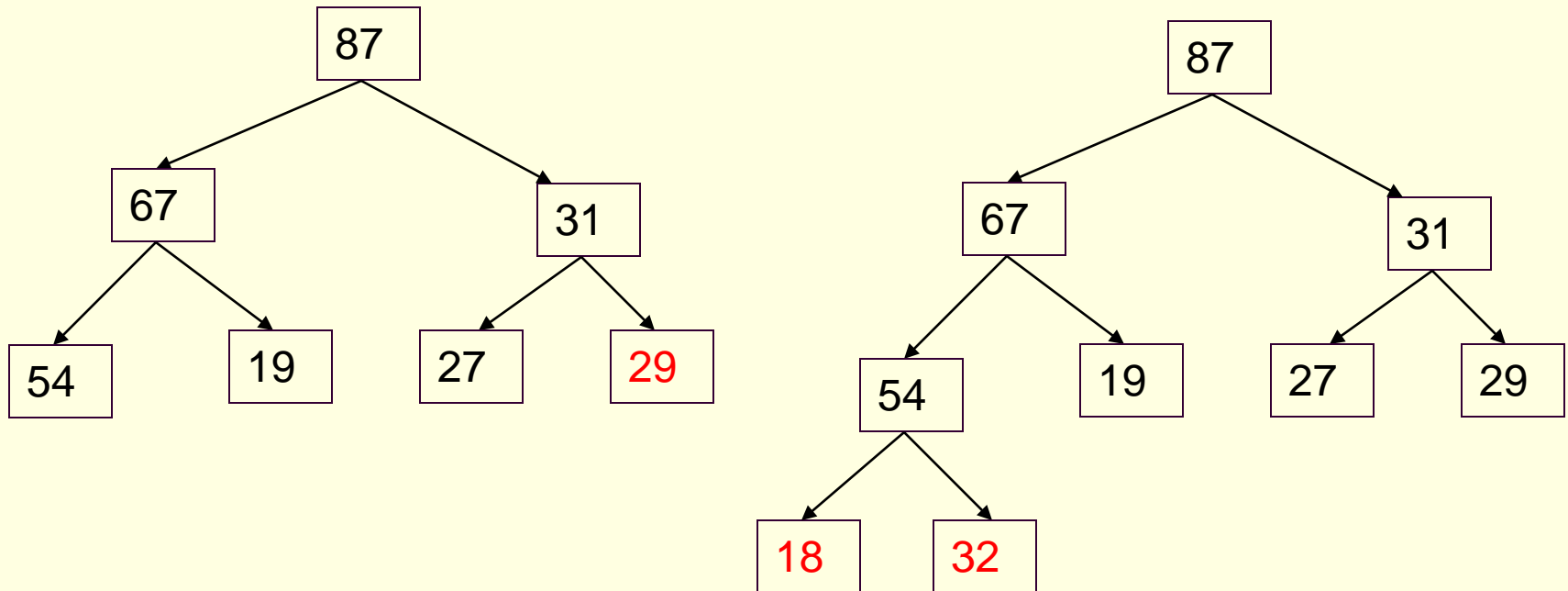
- Building a Heap from scratch (a Max heap)
  - Given: an unsorted list of n values
    - **54, 87, 27, 67, 19, 31, 29, 18, 32, 56, 7, 12, 31**





# Binary Heaps

- Building a Heap from scratch (a Max heap)
  - Given: an unsorted list of n values
    - **54, 87, 27, 67, 19, 31, 29, 18, 32, 56, 7, 12, 31**





# Binary Heaps

---

- Building a Heap from scratch
  - Running time:
    - How long does it take to do one insertion?
      - We just covered this!
      - An insertion takes  $O(\log n)$ 
        - As in the worst case, it has to Percolate all the way Up to root
    - And we have  $n$  elements to insert
    - **Running time to make a heap from  $n$  elements is  $O(n \log n)$**



# Binary Heaps

---

- Building a Heap from scratch
  - Can we do better than  $O(n \log n)$  time?
    - Turns out that we can
  - Start by arbitrarily placing your elements into a complete binary tree
  - Then, starting at the lowest level,
  - Perform a Percolate Down (if necessary)
  - So we work from the bottom and go up to the root
  - Performing a Percolate Down at each node
    - Only if necessary
  - This function is known as **Heapify**



# Binary Heaps

---

- Building a Heap from scratch
  - Running time:
    - Note:
      - Realize that for any given complete tree, that is completely filled, the lowest level has  $\frac{1}{2}$  of the total nodes in a tree
      - In a complete tree of 31 nodes, the lowest level has 16 nodes
        - And since they are already at the lowest level,
        - Those 16 nodes will NOT need to Percolate Down



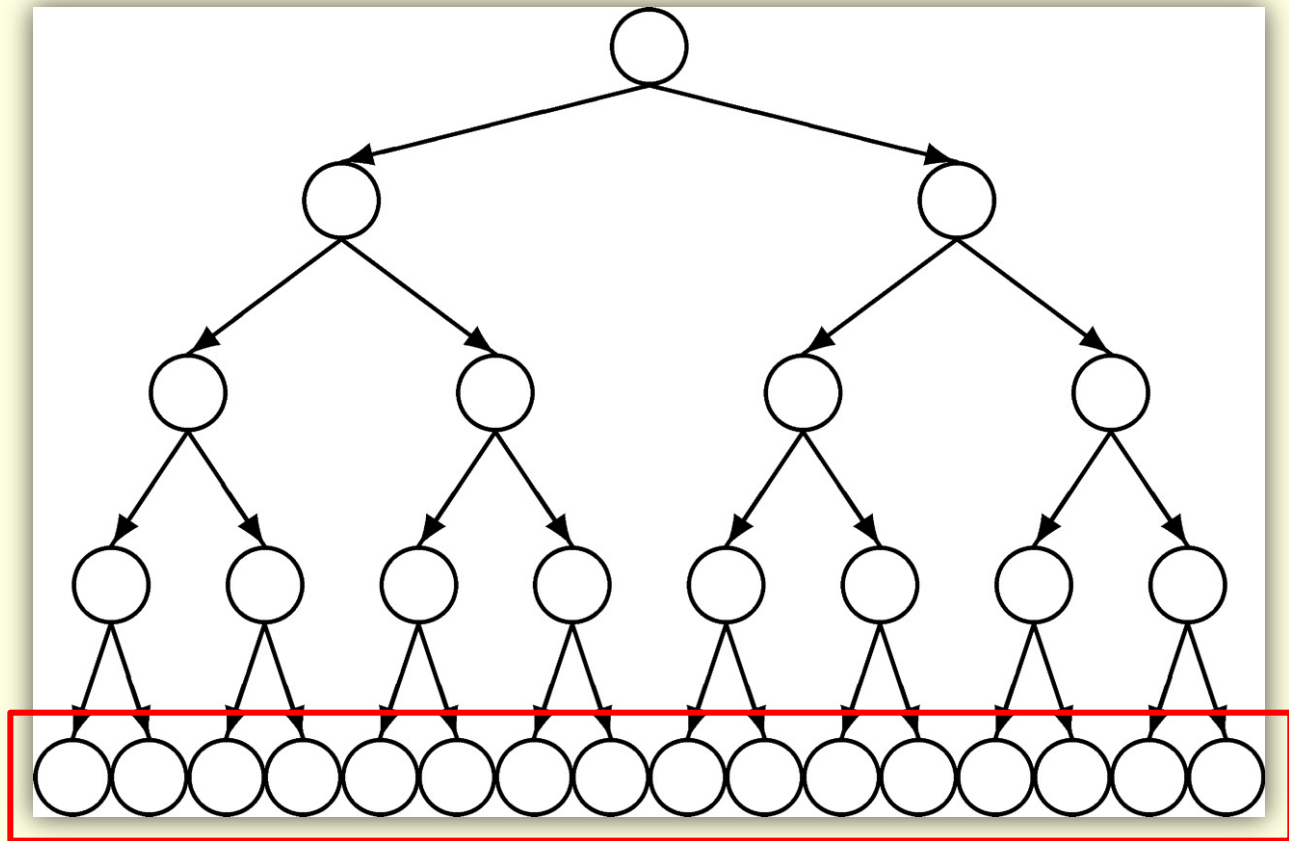


# Binary Heaps

## ■ Building a Heap from scratch

These nodes do NOT have to Percolate Down!

They are already at the bottom most level.





# Binary Heaps

---

- Building a Heap from scratch

- Running time:

- Note:

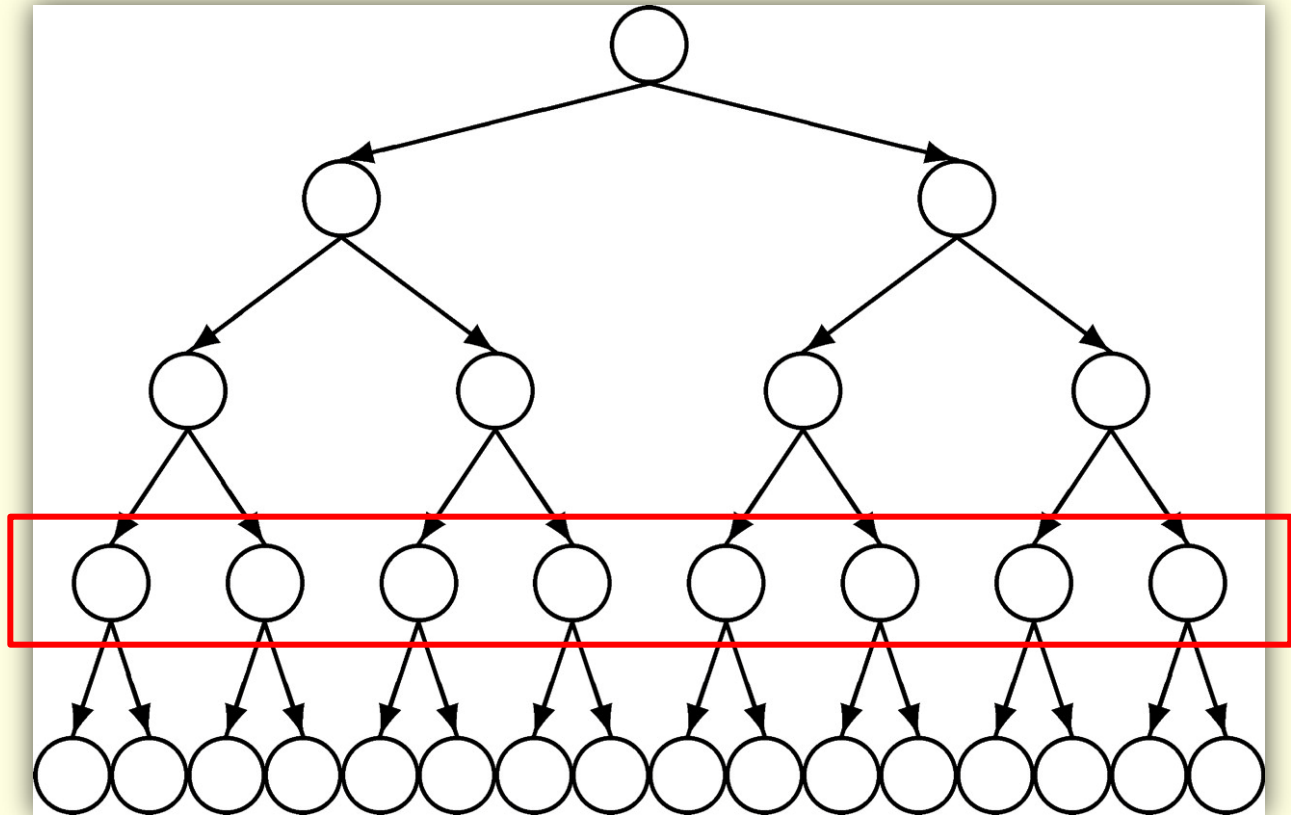
- Realize that for any given complete tree, that is completely filled, the lowest level has  $\frac{1}{2}$  of the total nodes in a tree
      - In a complete tree of 31 nodes, the lowest level has 16 nodes
        - And since they are already at the lowest level,
        - Those 16 nodes will NOT need to Percolate Down
      - The level above the 16 nodes has 8 nodes
      - What can we say about those 8 nodes?
      - Notice that, at MOST, those 8 nodes will have to Percolate Down only one level



# Binary Heaps

## ■ Building a Heap from scratch

These nodes  
only have to  
Percolate Down  
one level.





# Binary Heaps

---

## ■ Building a Heap from scratch

### ■ Running time:

#### ■ Note:

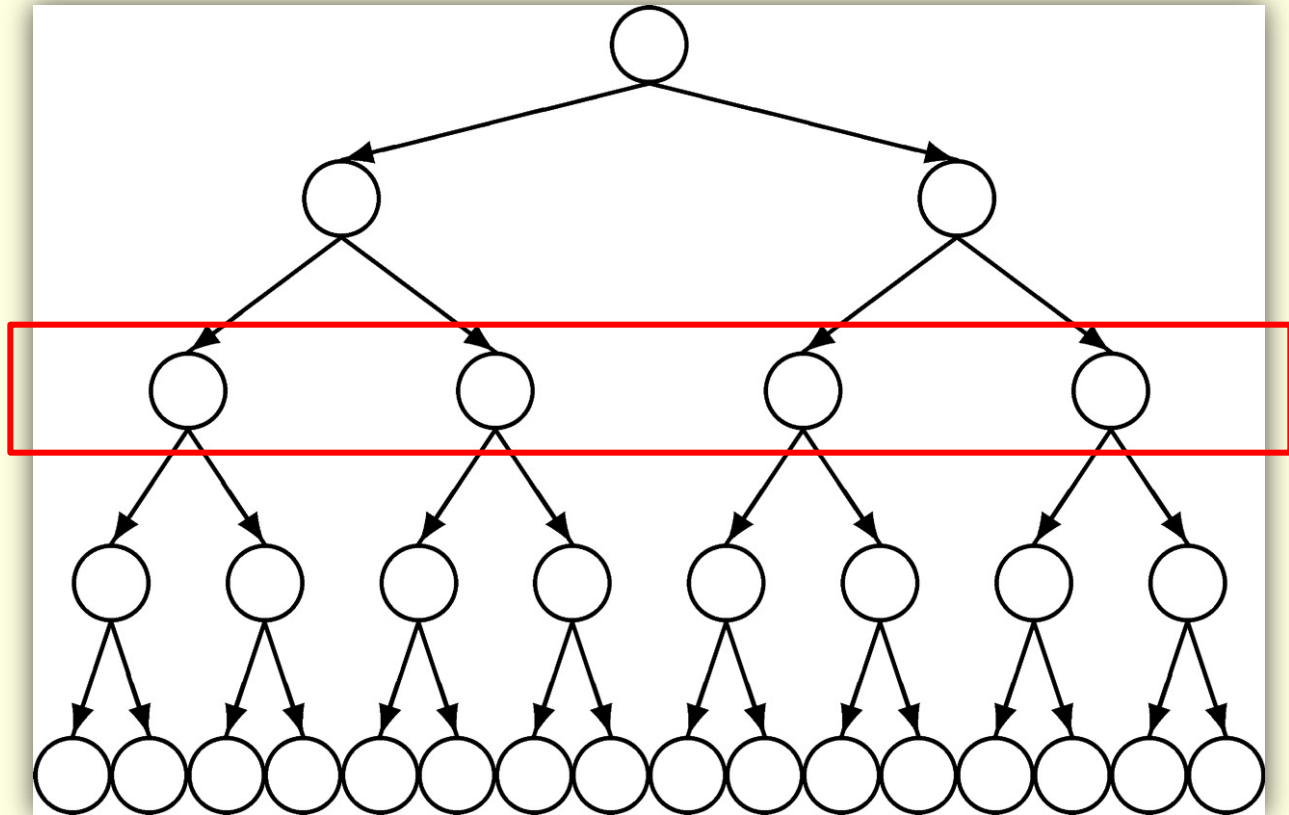
- Realize that for any given complete tree, that is completely filled, the lowest level has  $\frac{1}{2}$  of the total nodes in a tree
- In a complete tree of 31 nodes, the lowest level has 16 nodes
  - And since they are already at the lowest level,
  - Those 16 nodes will NOT need to Percolate Down
- The level above the 16 nodes has 8 nodes
- What can we say about those 8 nodes?
- Notice that, at MOST, those 8 nodes will have to Percolate Down only one level
- And the level above the 8 nodes has 4 nodes
- Those 4 nodes, at most, percolate down 2 levels, etc, etc.



# Binary Heaps

- Building a Heap from scratch

These nodes only have to Percolate Down two levels.





# Binary Heaps

## ■ Building a Heap from scratch

### ■ Running time:

- So only  $\frac{1}{2}$  of the nodes in a tree may need to be percolated down one level or more
- Only  $\frac{1}{2}$  of those ( $\frac{1}{4}$  of the total) may have to be percolated down two or more levels
- Only  $\frac{1}{2}$  of those ( $\frac{1}{8}$  of the total) may have to be percolated down three or more levels, etc., etc.
- So if we add up the total number of swaps, we get:
- $(\frac{1}{2}) * n + (\frac{1}{4}) * n + (\frac{1}{8}) * n + \dots \approx n$
- **So this Heapify function runs in  $O(n)$  time**



# Brief Interlude: FAIL Picture





# Daily UCF Bike FAIL



Courtesy of  
Kyle Perez





# Binary Heaps

---

- Implementing a Binary Heap
  - Remember:
    - a binary heap is a complete binary tree
  - So we can implement this binary tree as an array!
  - How?
    - If a tree is “complete”,
      - The root would be the 1<sup>st</sup> position of the array (index 1)
      - The two children of the node would be in index 2 and 3
      - The 4 nodes on the next level would be in index 4 – 7
      - The 8 nodes on the next level would be in index 8 - 15
      - and so on



# Binary Heaps

---

## ■ Implementing a Binary Heap

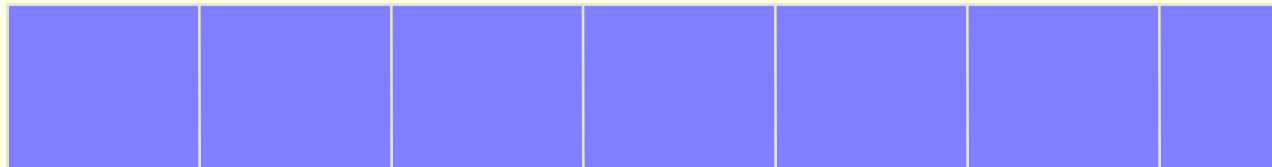
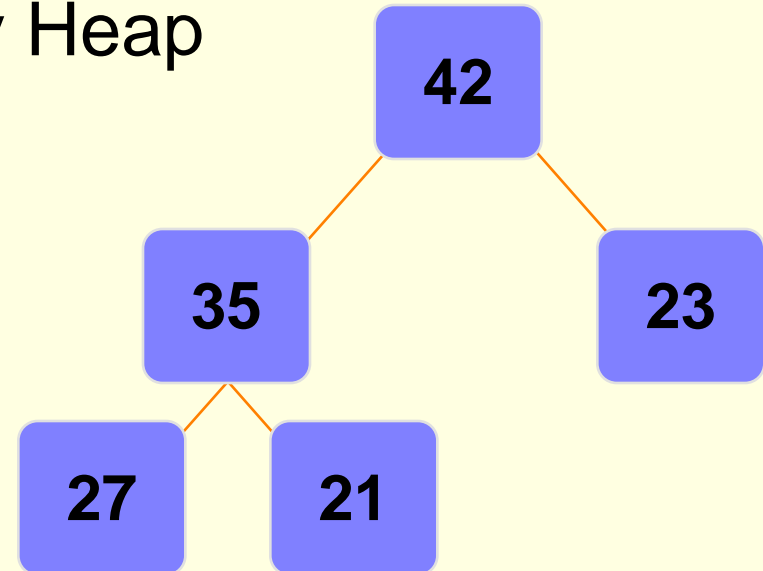
### ■ Notes:

- So we are wanting to implement one ADT
  - A Priority Queue
- To do so, we utilize another ADT
  - A Heap
- And to implement the actual Heap, which, in turn, implements the Priority Queue
  - **We use an array!**
- So after all of this, we simply use an array
- **And the way we dereference the array and manipulate the data is what makes “the array a tree”**



# Binary Heaps

- Implementing a Binary Heap
- We store the data from the nodes in a partially-filled array.



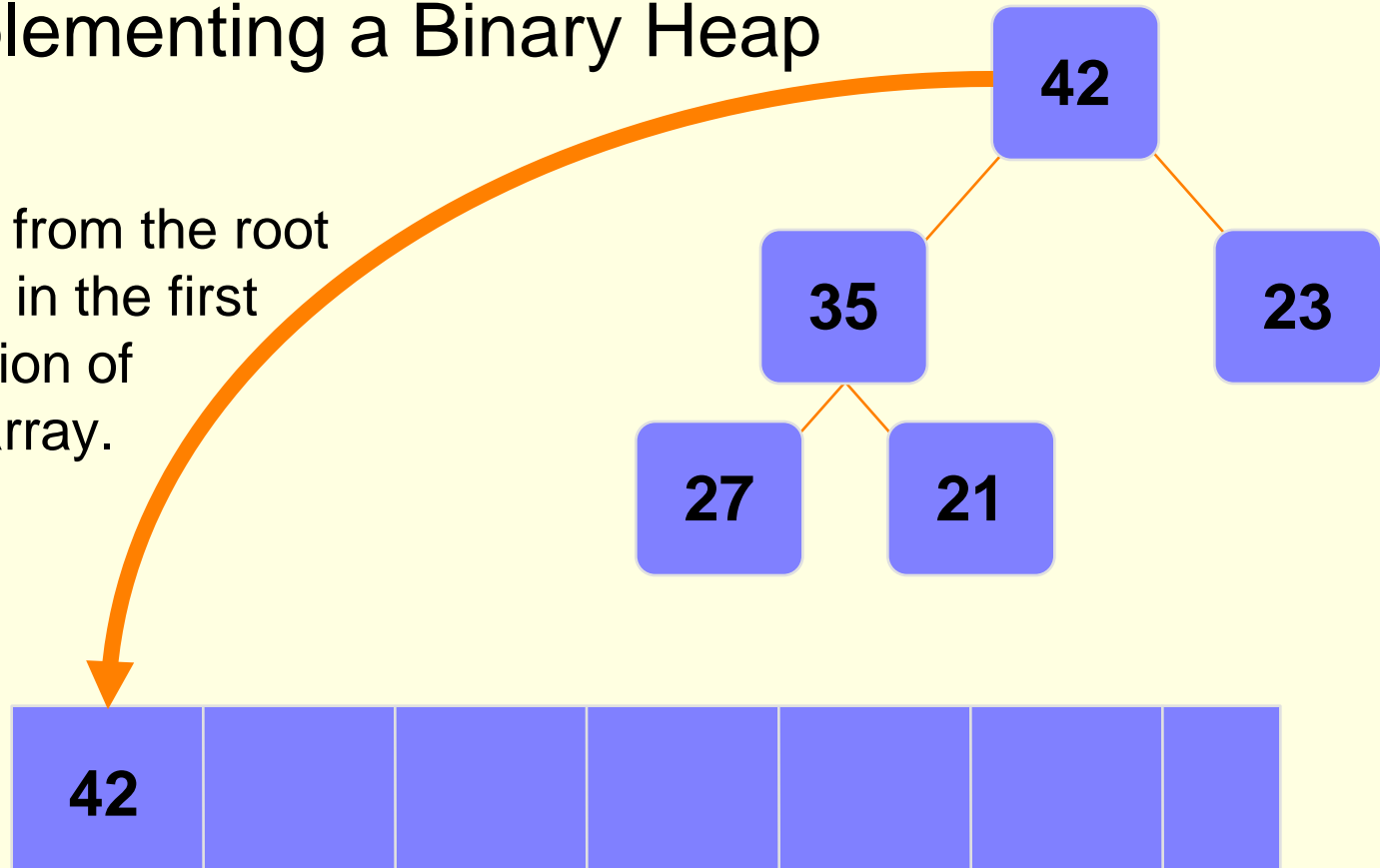
An array of data



# Binary Heaps

- Implementing a Binary Heap

- Data from the root goes in the first location of the array.

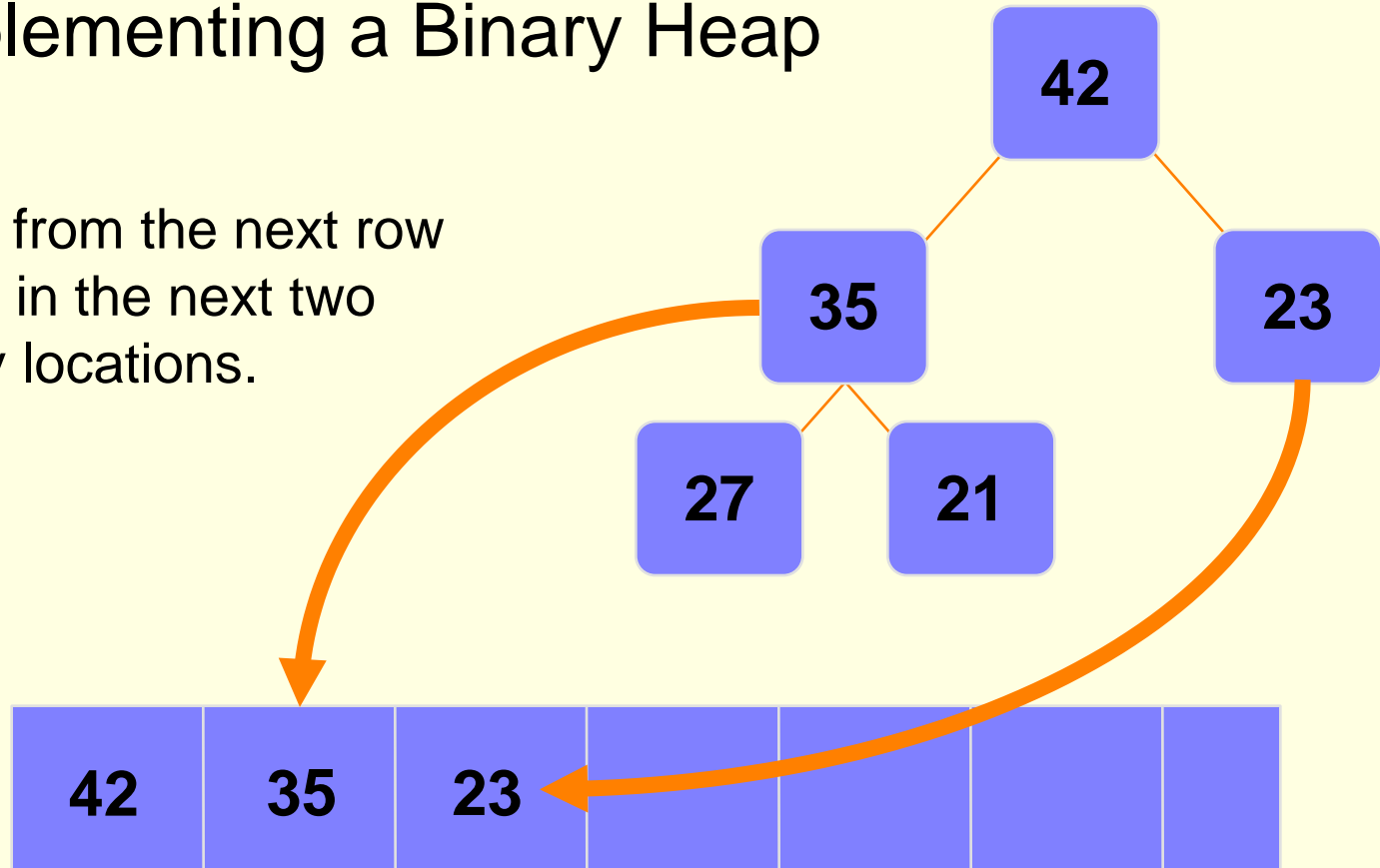


An array of data



# Binary Heaps

- Implementing a Binary Heap
- Data from the next row goes in the next two array locations.



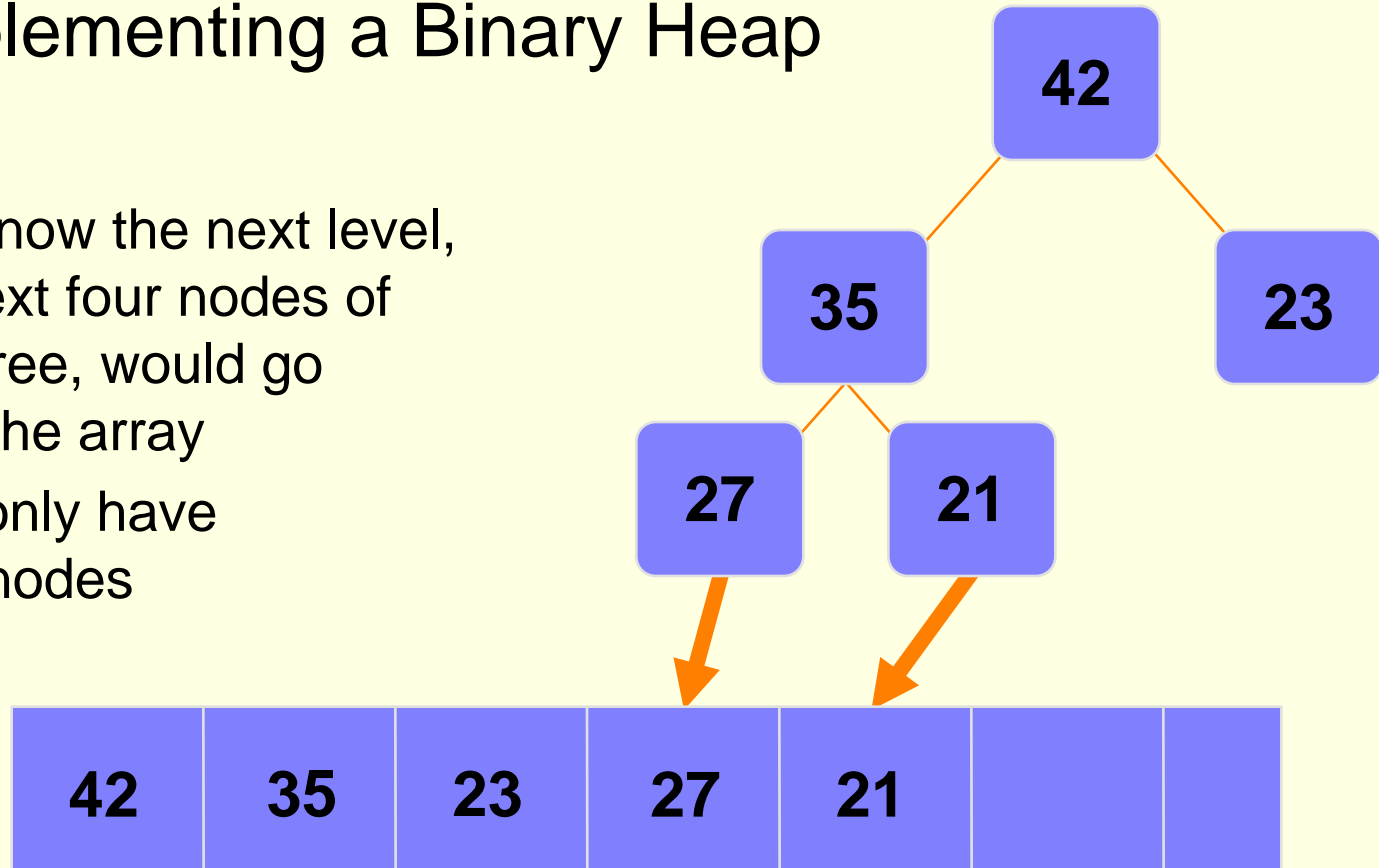
An array of data



# Binary Heaps

- Implementing a Binary Heap

- And now the next level, or next four nodes of the tree, would go into the array
- We only have two nodes



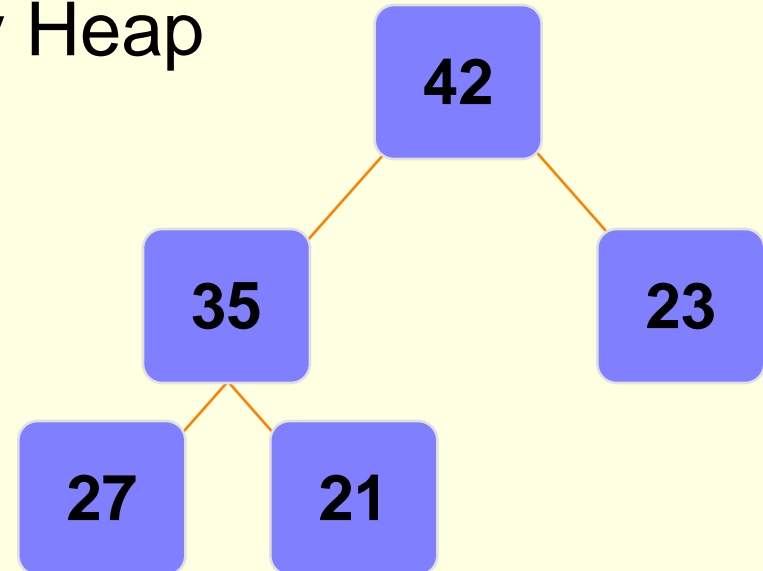
An array of data



# Binary Heaps

- Implementing a Binary Heap

- We are only concerned with the front part of the array
- If the tree has 5 nodes, then we only care about the first five spots of the array



An array of data

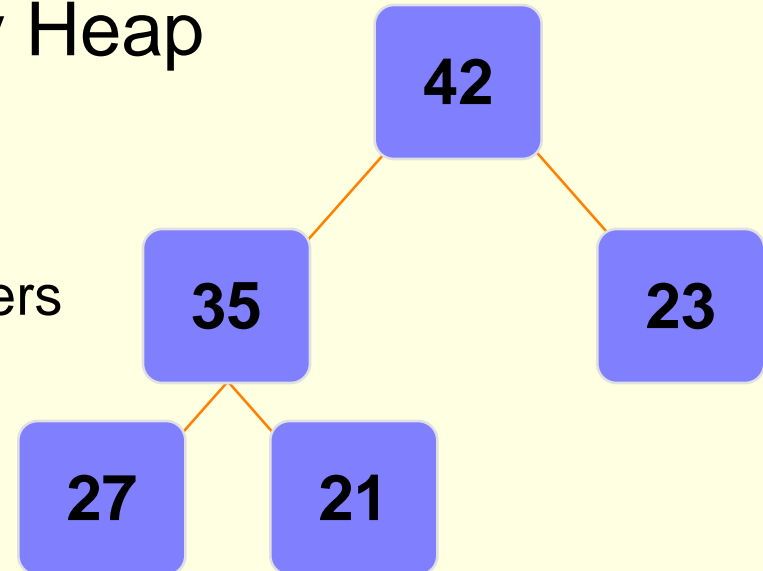
We don't care what's in this part of the array.



# Binary Heaps

- Implementing a Binary Heap

- The links between the tree's nodes are not stored as pointers
- The only way we “know” that the “array is a tree” is based on how we choose to manipulate the array



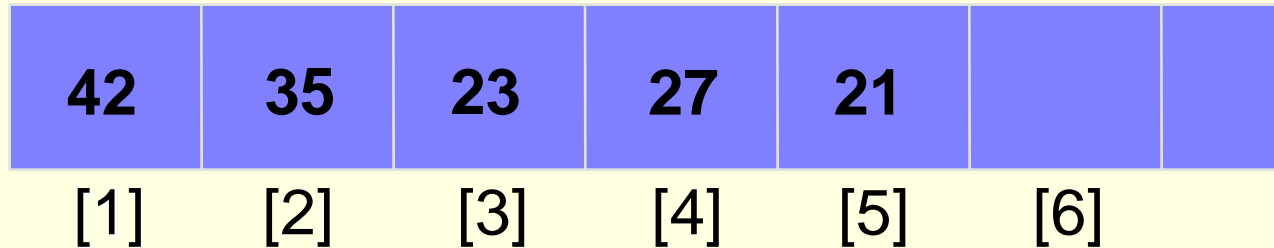
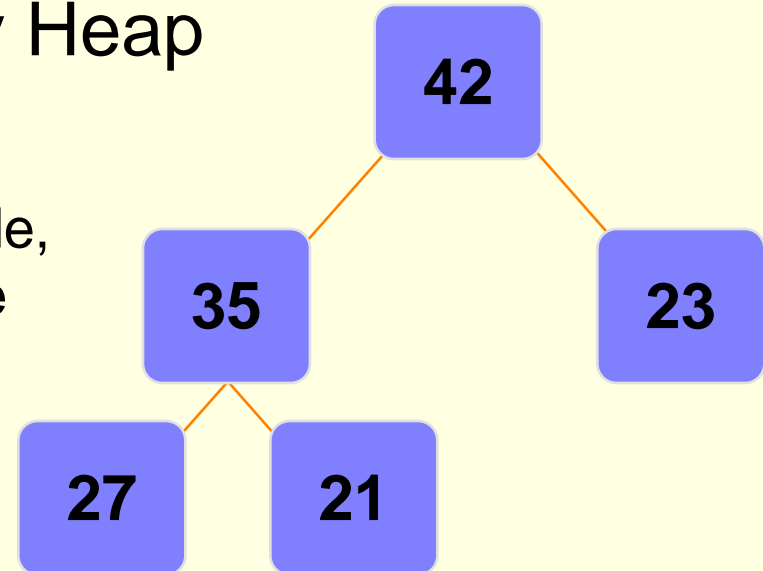
An array of data





# Binary Heaps

- Implementing a Binary Heap
- If you know the index of a node, then it is easy to figure out the index of that node's parent or children

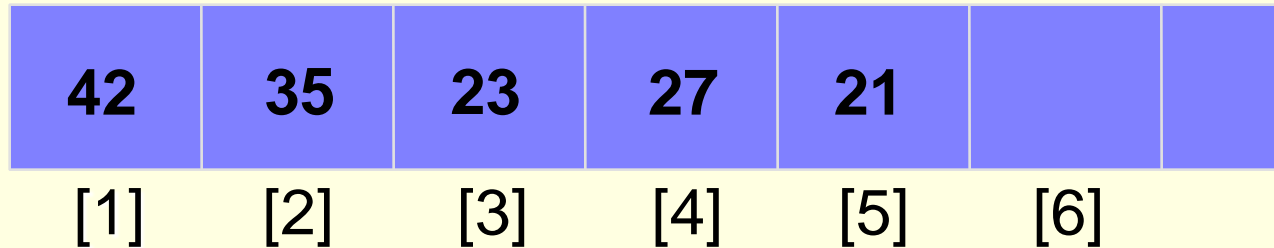
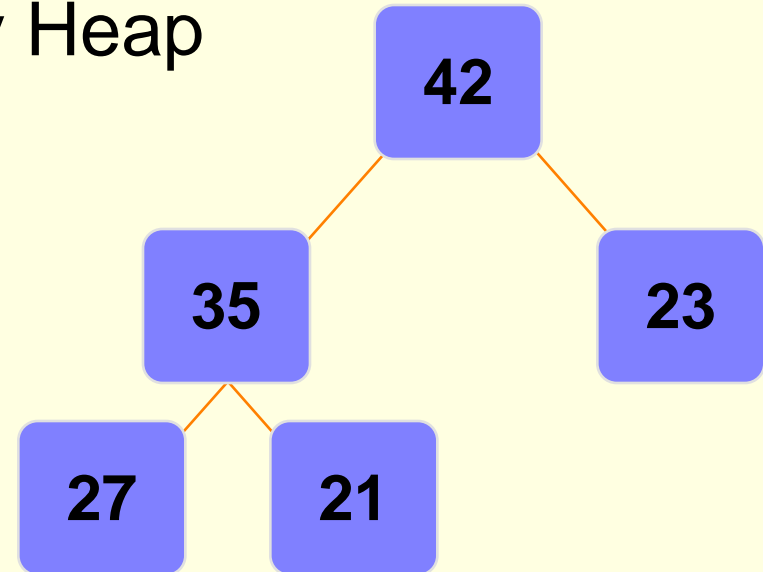




# Binary Heaps

## ■ Implementing a Binary Heap

- The name of our array is  $A[]$
- Root is at position  $A[1]$
- Left child of  $A[i] = A[2i]$
- Right child of  $A[i] = A[2i+1]$
- Parent of  $A[i] = A[i/2]$

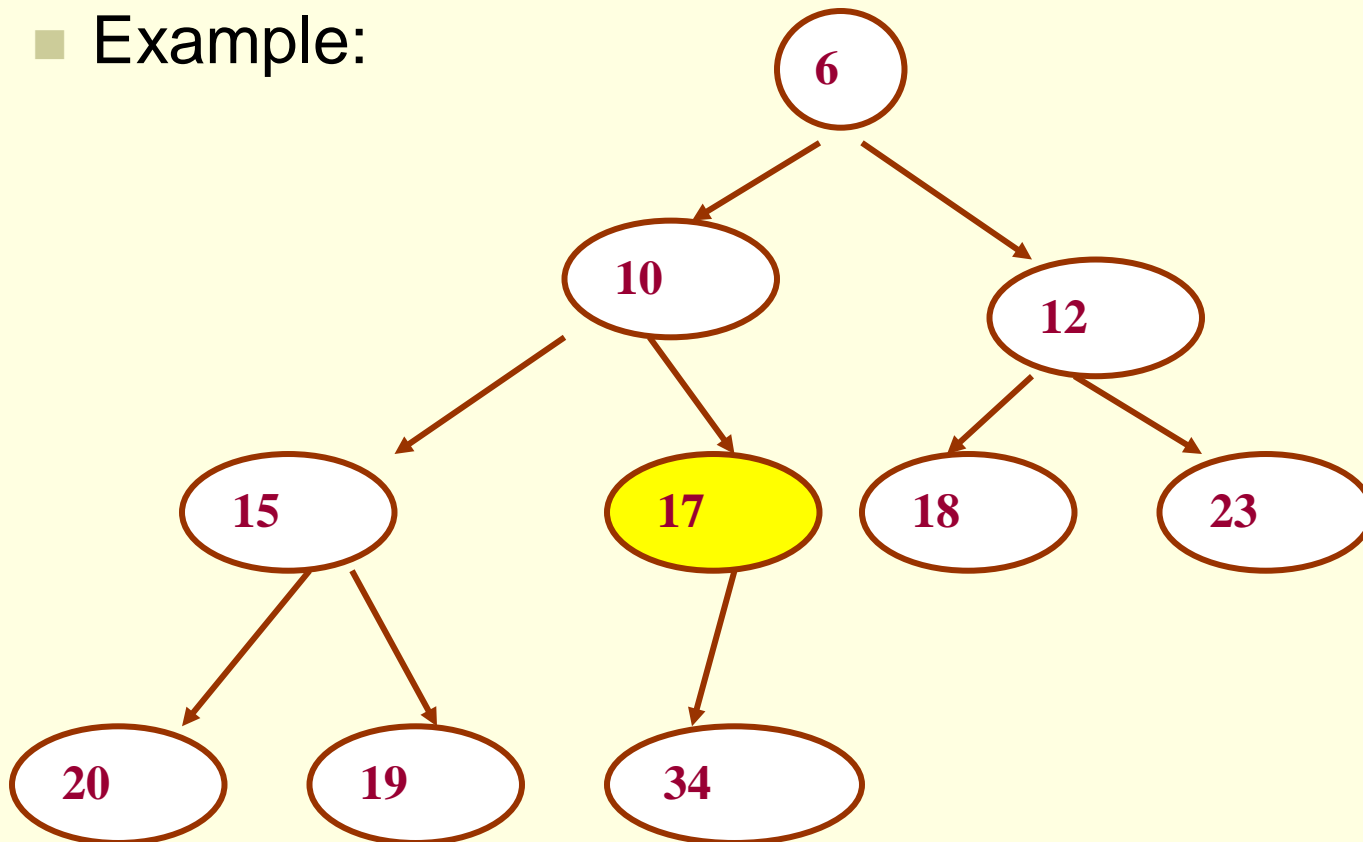




# Binary Heaps

- Implementing a Binary Heap

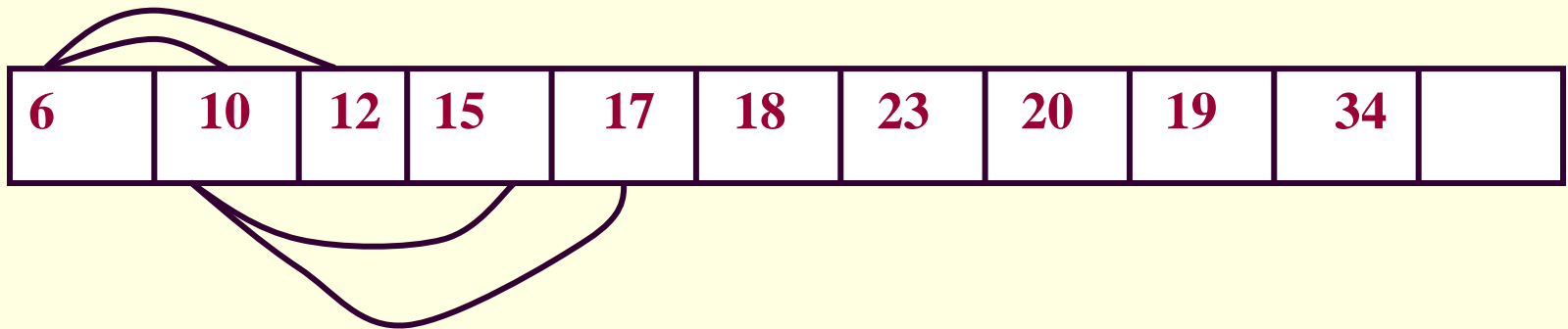
- Example:





# Binary Heaps

- Implementing a Binary Heap
  - Example:



- Consider node 17:
  - Position in the array: 5
  - It's parent is 10, and is located at position  $[5/2] = 2$
  - 17's left child is node 34, and located at position  $5*2 = 10$
  - 17 has no right child. Position  $(2*5 + 1) = 11$  (empty)



# Binary Heaps

## ■ Heapsort

- We can use heaps to sort our data
- Here's the algorithm:
  - Build a heap with all the  $n$  items
    - Takes  $O(n)$  time (cuz we add to a binary tree and run **Heapify**)
  - Extract the minimum item (if a Min-heap)
    - $O(1)$
  - Fix the heap after extraction
    - $O(\log n)$
  - Perform this extraction  $n$  times for all the elements
  - Store these removed items, sequentially, in an array
  - Running time:  $O(n \log n)$



# Binary Heaps

---

- Summary:
  - A binary heap is a tree that satisfies 2 properties:
    - The Heap Property
      - Max-heap OR Min-heap
    - The Shape Property
      - Must be a complete binary tree
  - To add elements to a heap
    - Place element at next available spot and Percolate Up
  - To remove elements from a heap,
    - Delete root, as it is always the one you want to remove
    - Then copy last element to root's position
    - Finally, Percolate Down



# Binary Heaps

---

- Summary:
  - The purpose of a heap is essentially to implement a Priority Queue
  - So we use one ADT to implement another ADT
  - And then, at the end of it all, we simply implement the Heap as an array!
    - We know our array is a Heap (a tree) based on how we dereference the array and how we choose to manipulate the data



# Binary Heaps & Priority Queues

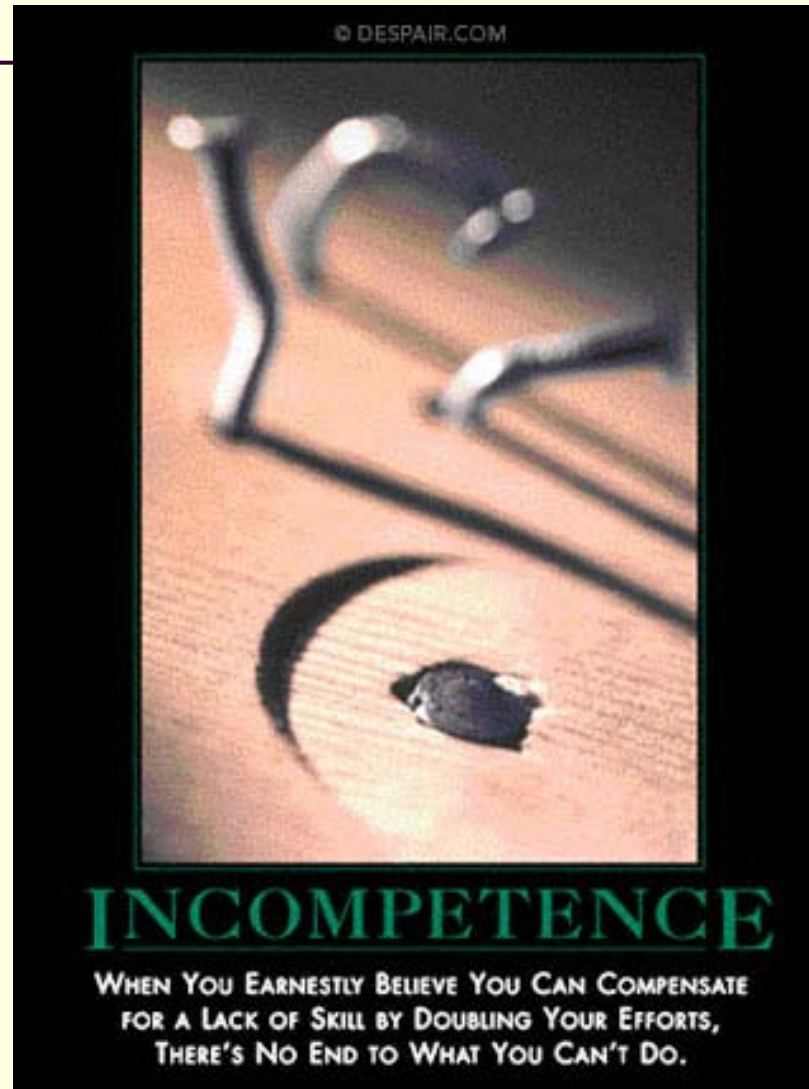
---

**WASN'T  
THAT  
PRODIGIOUS!**





# Daily Demotivator



# Heaps & Priority Queues



Computer Science Department  
University of Central Florida

*COP 3502 – Computer Science I*