ECEN 758 Data Mining and Analysis Heaps and Binary Search Trees

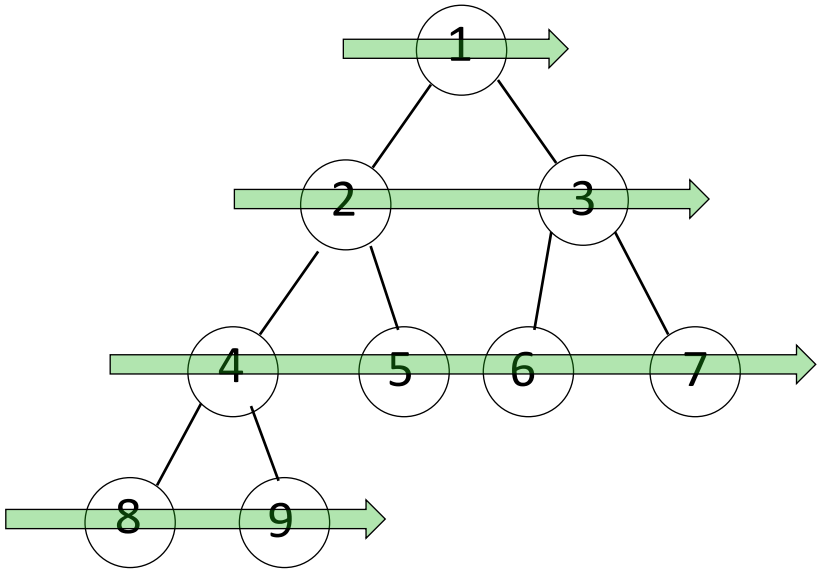
Nick Duffield

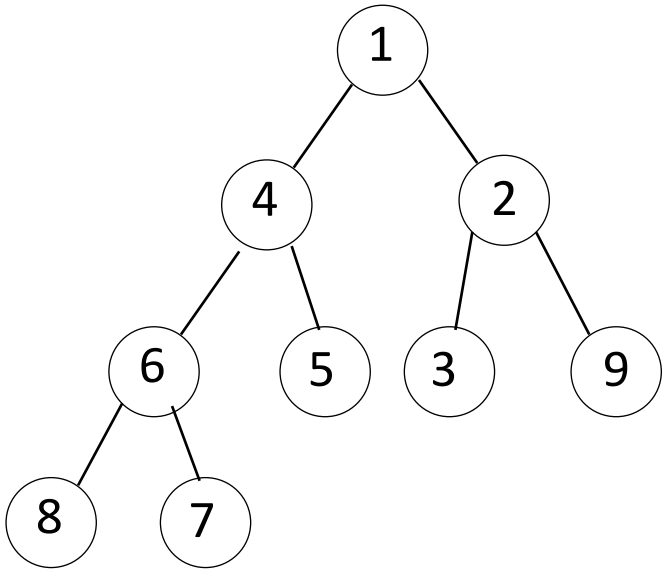
Department of Electrical & Computer Engineering

Texas A&M Institute of Data Science

Heaps

- **Heap (low priority version)**
  - Each node has at most two children
  - Parents have lower priority than children
  - Each depth of is filled in order (top to bottom, left to right)
- **Two possible heaps storing the numbers {1,..9}**



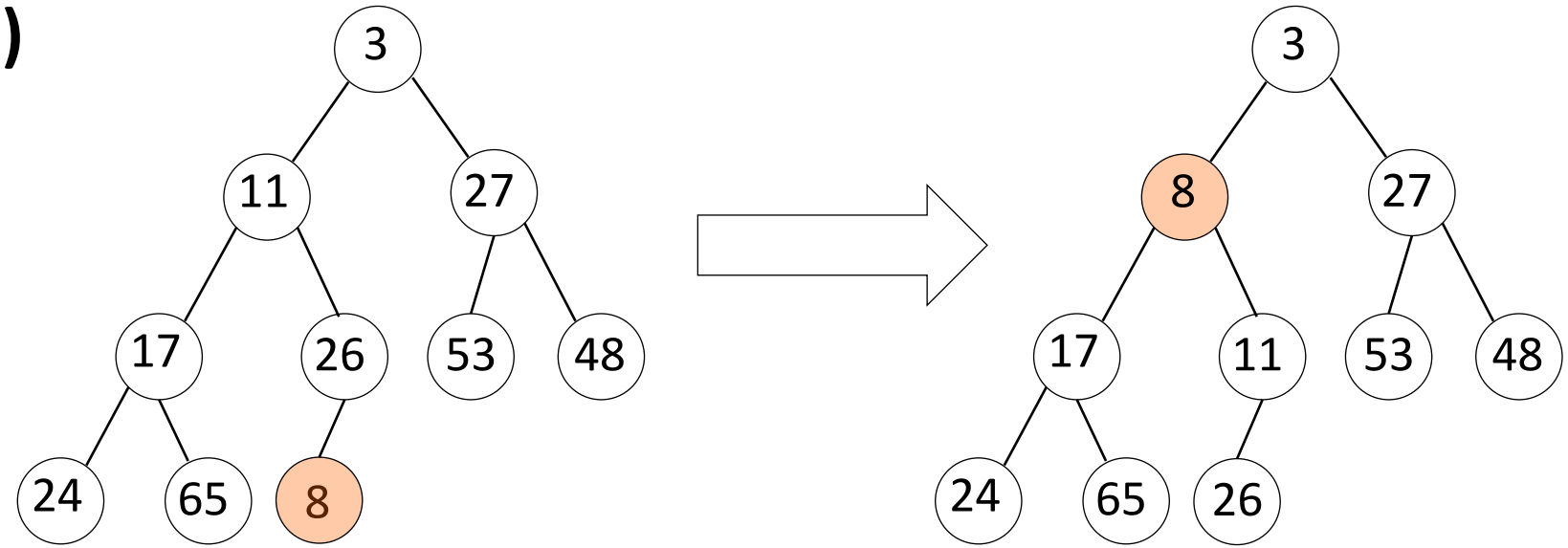


- **Heaps of same size have same topology; balanced binary**

Heap Insertion

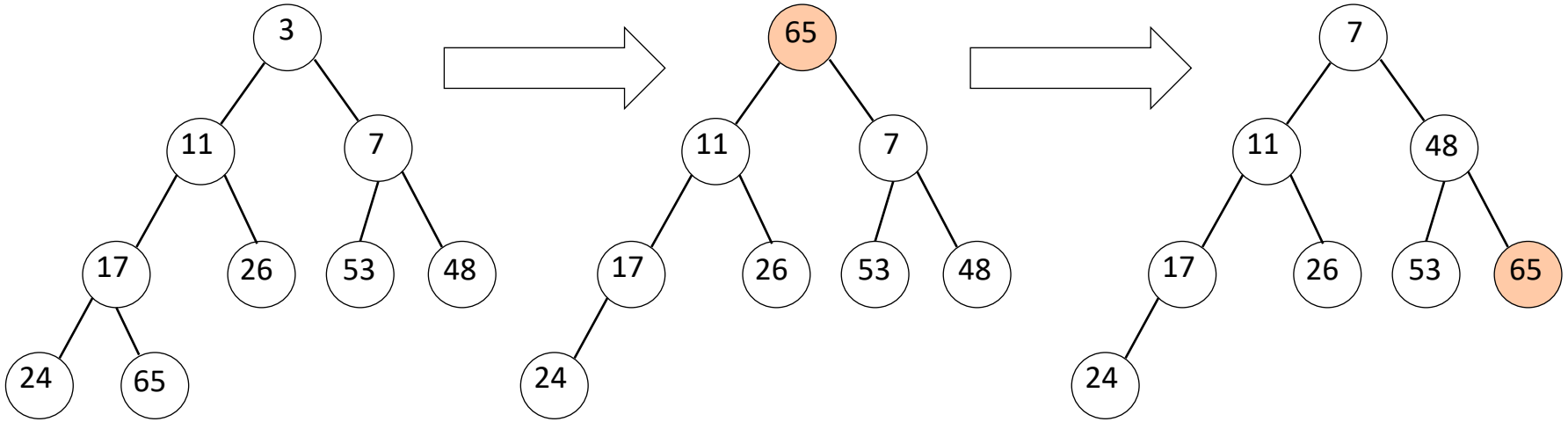
- **Insert new item at next free position**
- **Bubble up by swapping with until a heap is obtained**
- **Have to do at most h swaps, where h = O(log n) is tree depth**

- **Insert(8)**



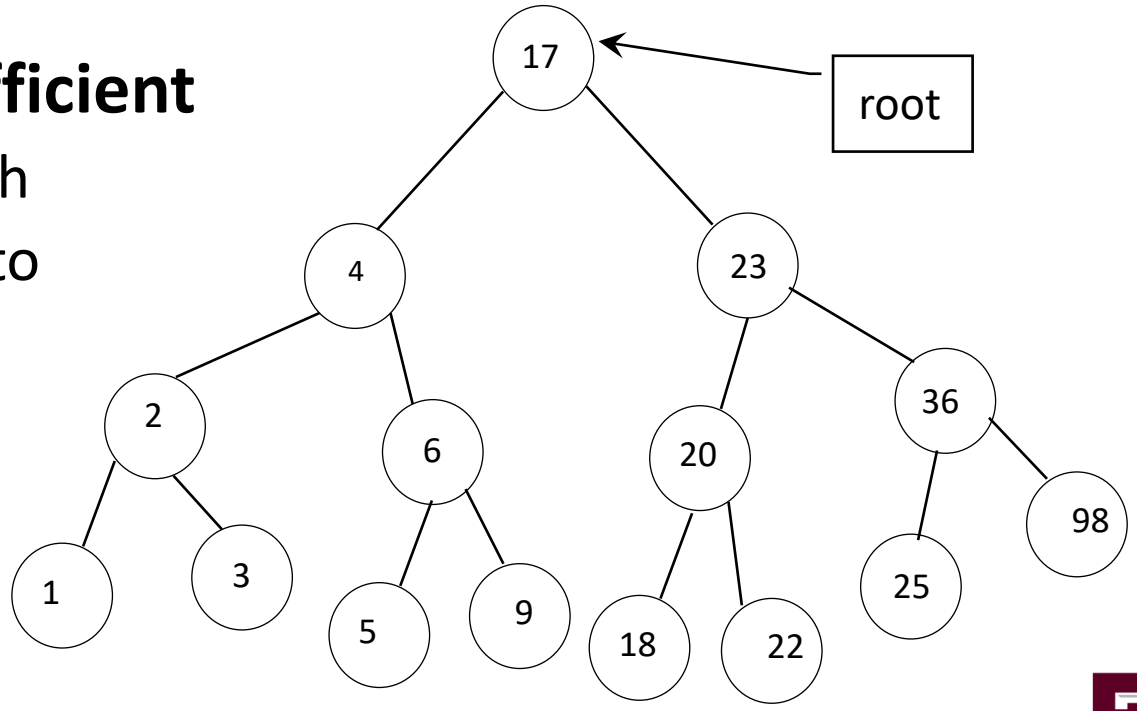
Heap Removal

- **Only the root node is removable**
  - Lowest priority item when implementing a (low)-priority queue
- **Remove root node**
- **Move node in last position to root**
- **Bubble down by swapping with smaller child until heap formed**
  - Takes at most h swaps, where h = O(log n) is tree depth



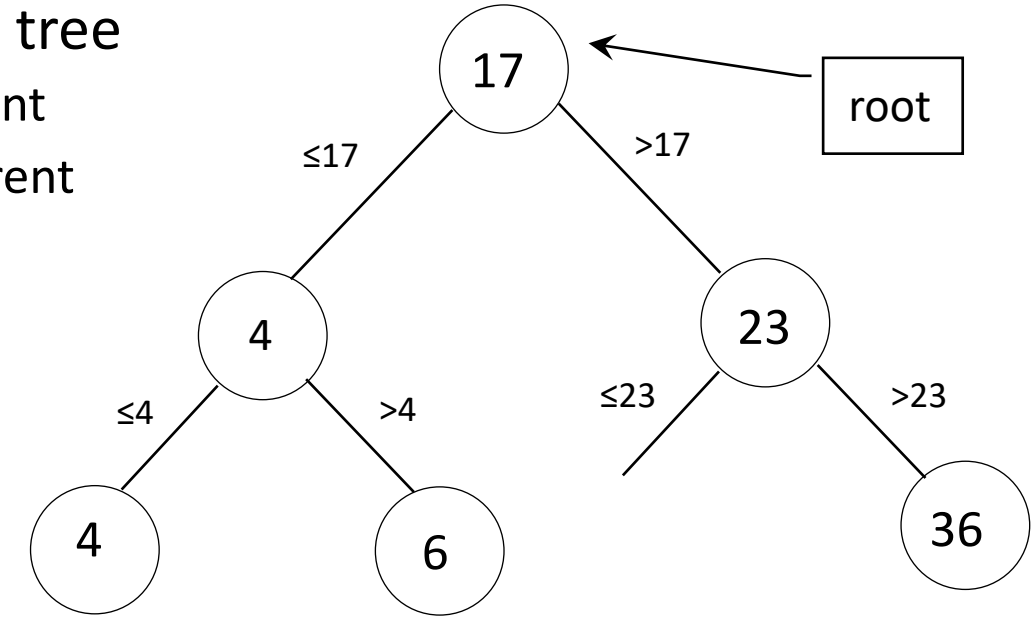
Binary Search Trees

- **Data structure to store and retrieve points in ordered set**
  - e.g. numbers with order of “<“
- **Points do not need to be added in any particular order**
  - No presorting needed
- **Tree like structure is very efficient**
  - Stores n= 2^h-1 items if depth h
  - Computational cost O(log n) to retrieve any item



Storing in a Binary Search Tree

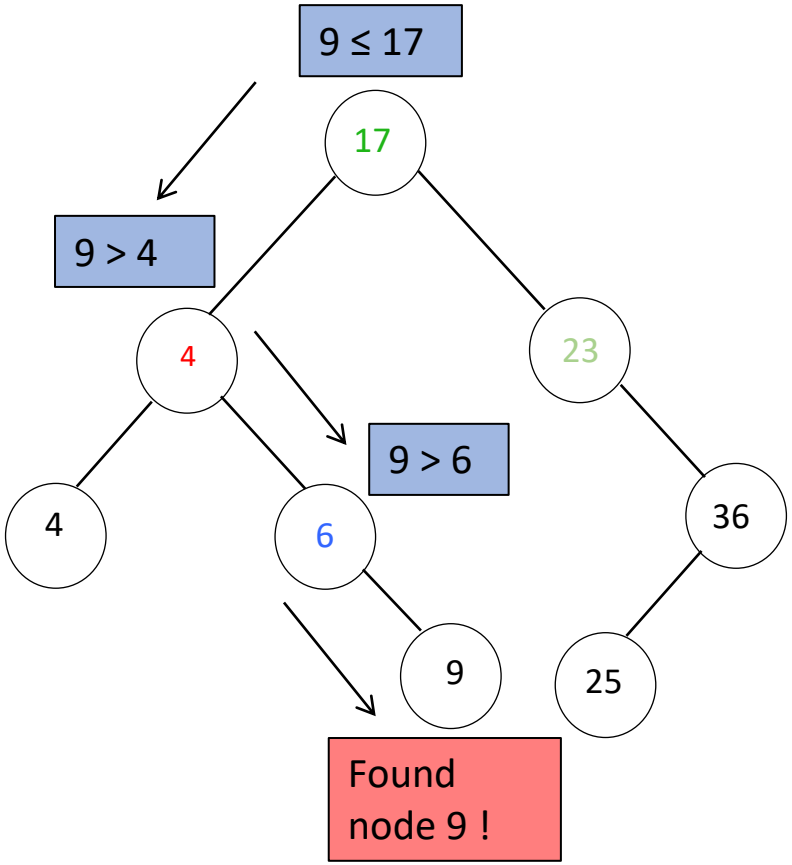
- **Storing an ordered set, e.g., {17, 4, 6, 23, 36, 4} with > order**
  - Store first element at root: 17
  - Pass further elements down tree
    - To left child if element ≤ parent
    - To right child if element > parent



- Tree is depth O(log m) for m items
- Computational complexity to add, delete or retrieve item is O(log m)
- Binary search easy: go left/right until found, O(log m) steps
- Need to **rebalance**: maintain bounded depth (i.e. approx. symmetry)

Search on a Binary Tree

- **Search for some node (and whatever information attached)**
  - say node 9
- **Start at root**
- **Iterate until found:**
  - Branch left if 9 ≤ node
  - Branch right if 9 > node



References

- **Binary trees in general:**
  - Goodrich: Data Structures and Algorithms in Python. Ch. 8
    - TAMU Library, online
  - These notes describe insertion and search; BST also provide for deletion, rebalance and other operations.