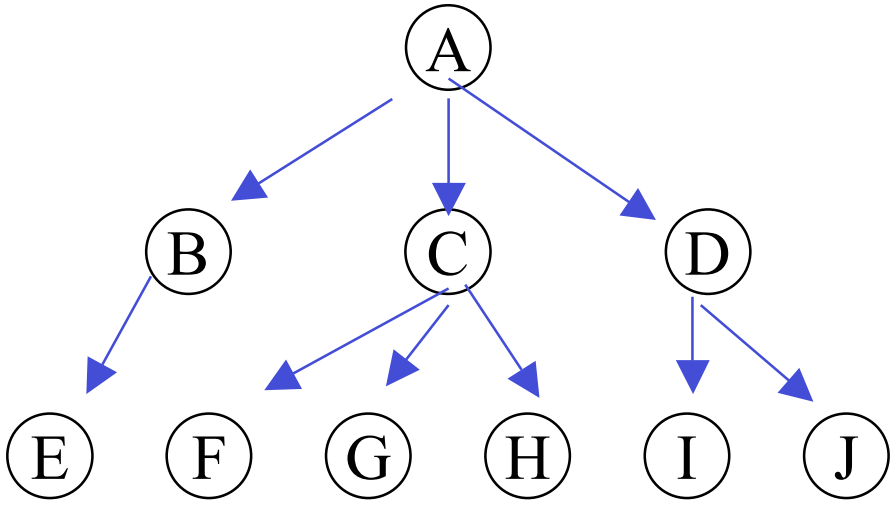
CS112: Data Structures

Lecture 10

Heaps

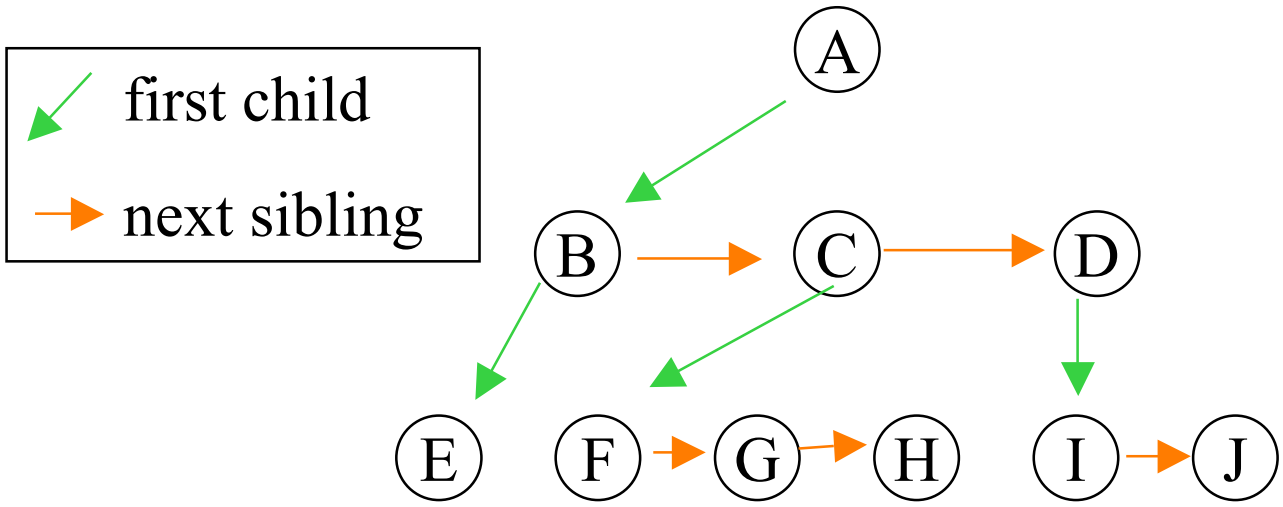
Review: General Trees

- **Each node has an arbitrary number of children**
- **Problem: representation of a node**



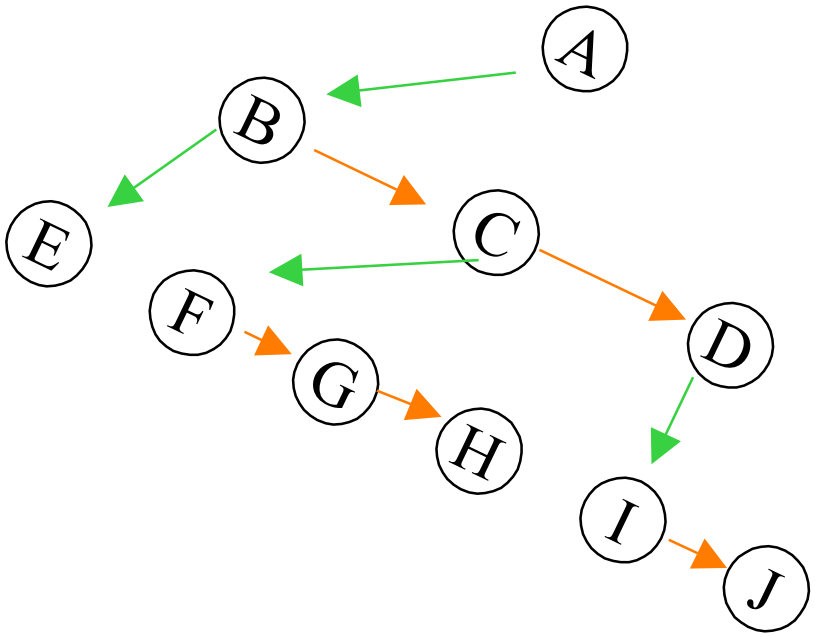
General Trees

- **Each node has an arbitrary number of children**
- **Problem: representation**
- **Solution: linked list of children**



General Tree as Binary

- **First child <=> Left child**
- **Next sib <=> Right child**



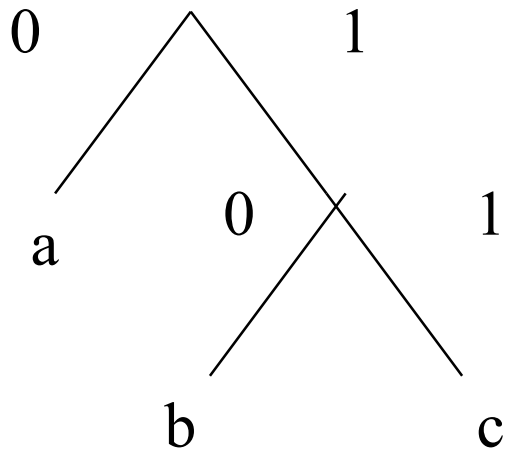
Data Compression

- **In most data some symbols appear more often than others**
  - **Eg English text ‘e’ appears more often than ‘q’**
- **In ascii code, each character is 8 bits.**
- **Suppose we had a code in which common symbols took fewer bits and uncommon symbols took more bits**

Huffman Code

- **EG 3 symbols: a, b, c, with a most frequent**
- **Code: 0 = a, 10 = b, 11 = c**
  - **abacac = 010011011**
  - **9 bits = 1.5 bits/character,**
  - **Versus 2 bits/character for fixed length code**
- **Decode: 11010 = cab**
- **Suppose code was 1 = a, 10 = b, 11 = c**
  - **Is 111 ca or ac or aaa?**
  - **No character's code can be prefix of another**

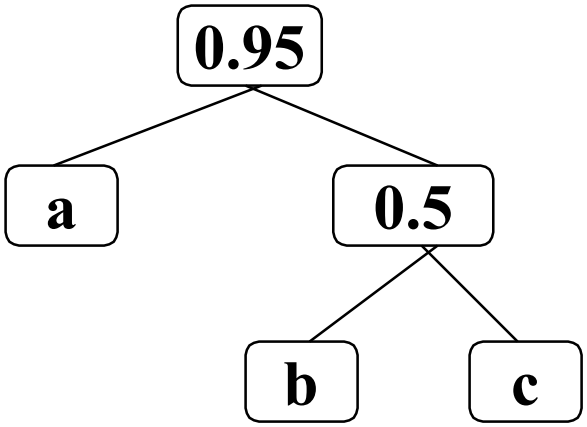
Huffman Code as a Tree



Symbols only at leaves

Algorithm

- 2 queues: S, T
- Contents of each queue: Tree
  - A leaves stores a symbol
  - A non-leaf mode stores total frequency of all symbols at leaves under this node
- E.g., for frequencies a: 0.45, b: 0.3, C: 0.2:



Algorithm

- **2 queues:**
  - **S initially holds 1-node trees for all symbols, least likely first**
  - **T empty**

while not (S empty and t length == 1)

find two least-weight trees in S, T and dequeue them

make a tree with these two as subtrees

enqueue on T

Example

A	**.05**
B	**.1**
C	**.1**
D	**.2**
E	**.25**
F	**.3**

Book code

- **See Huffman.java and HuffmanDriver.java in dsoi.progs.src.zip in apps/tree/**

Priority Queues

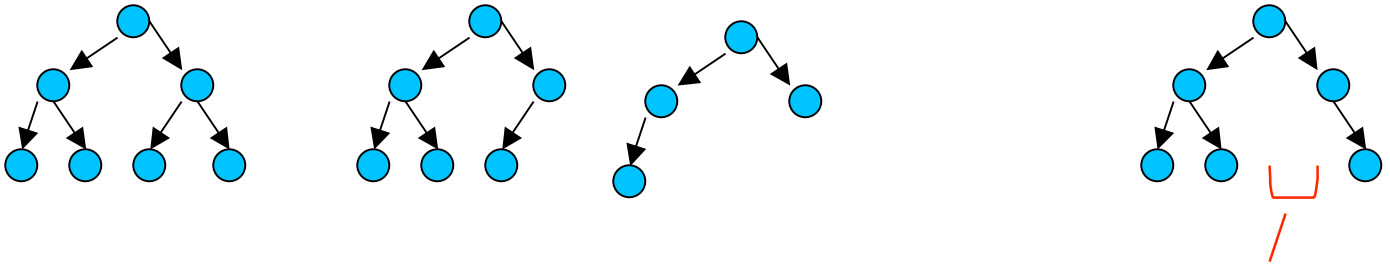
- **Each data item has a priority**
- **Add items to queue in any order**
- **Remove items in priority order**
  - **add A:5, B:3, C:6**
  - **remove C**
  - **add D:8**
  - **remove D, remove A**

Implement as an array

- **Unsorted array:**
  - **Insert one item O(1)**
  - **Remove one item O(n)**
- **Sorted array:**
  - **Insert one item: O(n)**
  - **Remove one item: O(1)**

Heap

- **A heap is a way to implement a priority queue with O(log n) complexity**
- **A heap is a complete binary tree**
  - **all levels except maybe the last are full**
  - **last level filled from left to right**

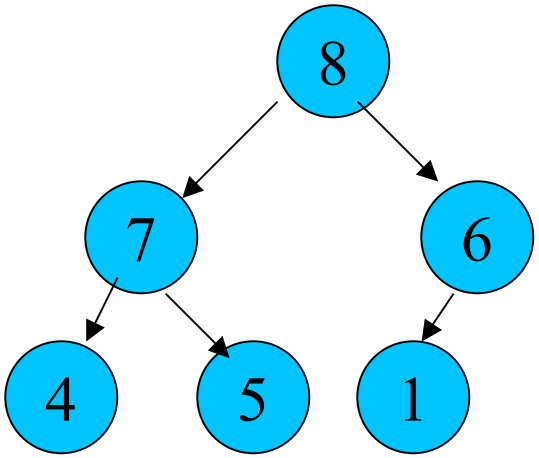


good

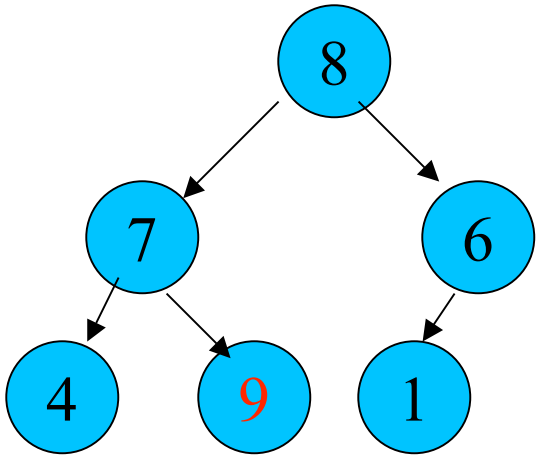
bad

Heap

- **The number at a node is greater than the number at any descendant**



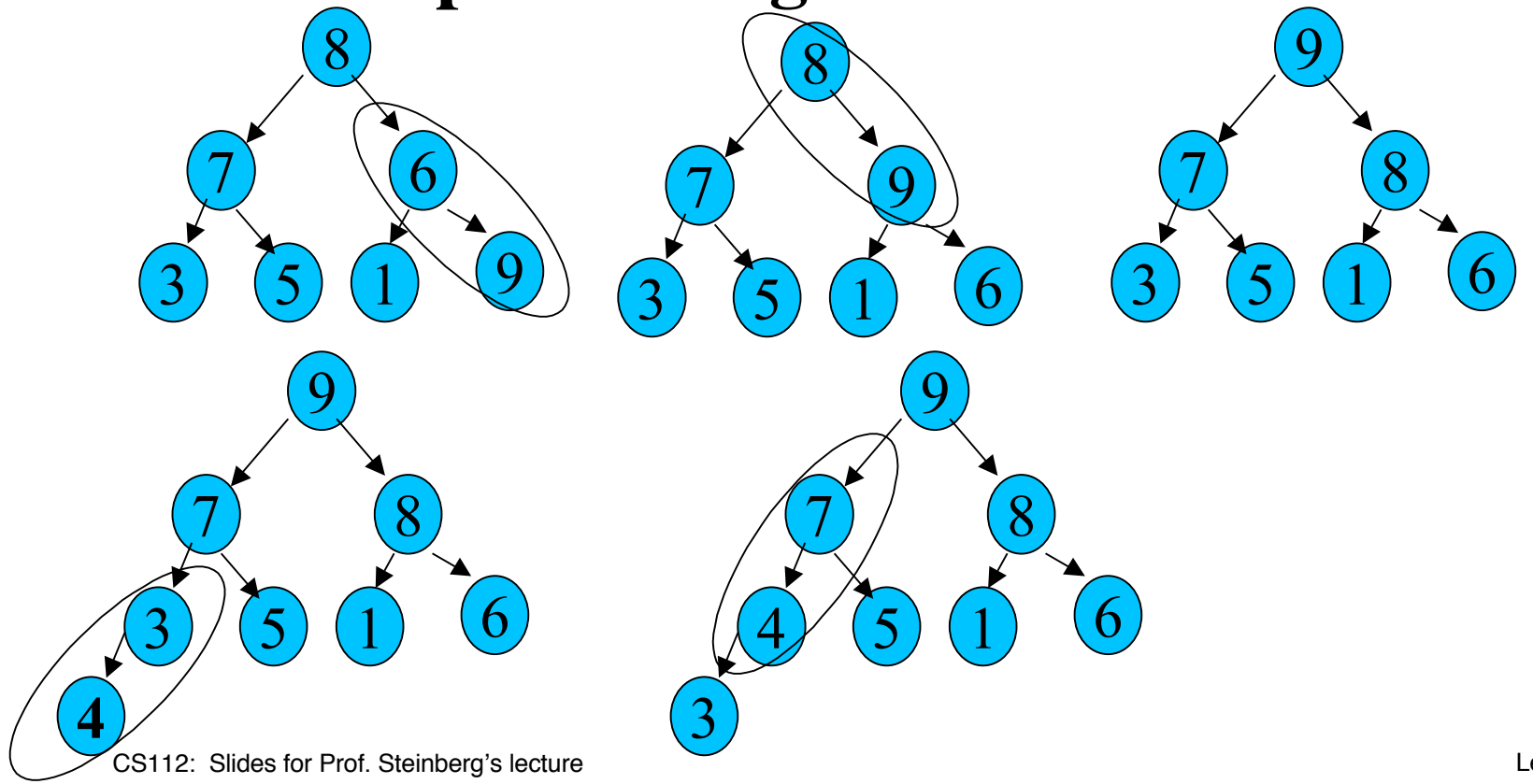
good



bad

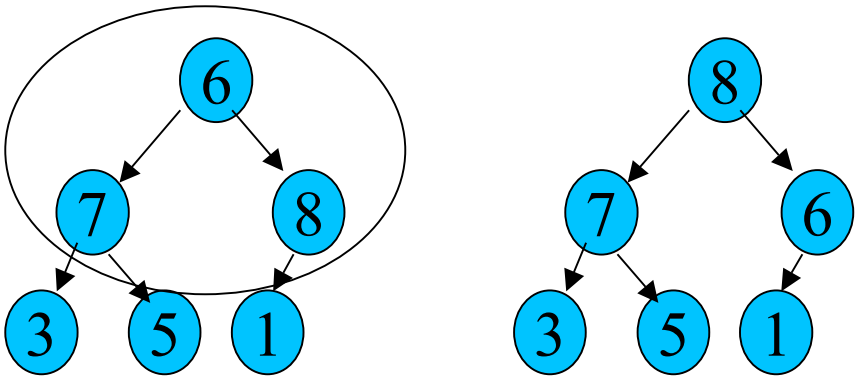
Heap Insert

- **Add node at end of last level**
- **Move up restoring order**



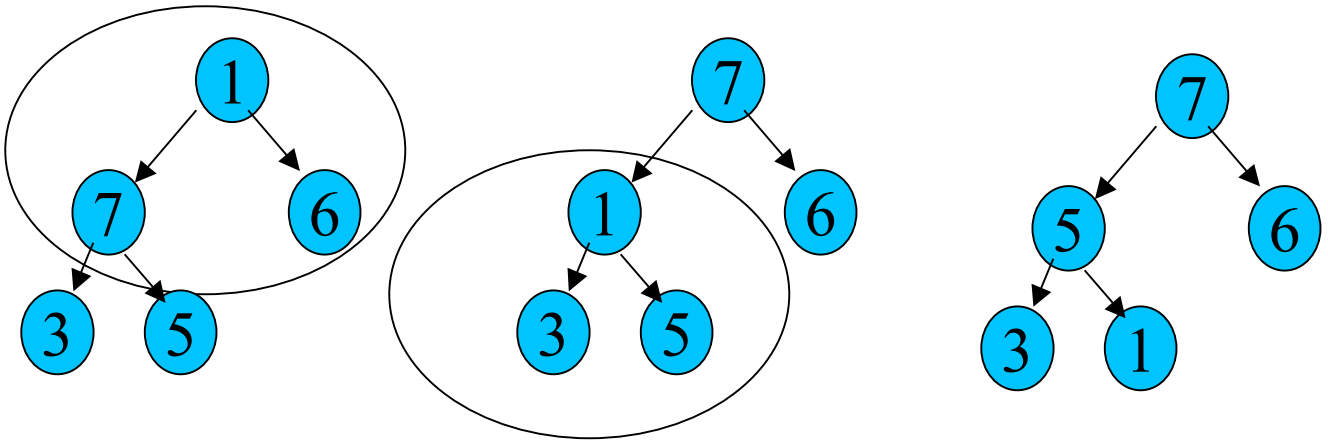
Heap Deletion

- **Copy out data at root**
- **Delete last node on last row & put data in root**
- **Move down restoring order**



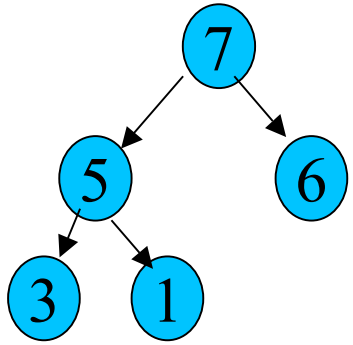
Heap Deletion

- **Compare current node and two children**
  - **if current node largest, stop**
  - **if left node largest swap current and left**
  - **ditto if right largest**



Heap Representation

- **Store heap in an array**
  - **For node at index j, children are at 2j+1 and 2j+2**
  - **Root at index 0**

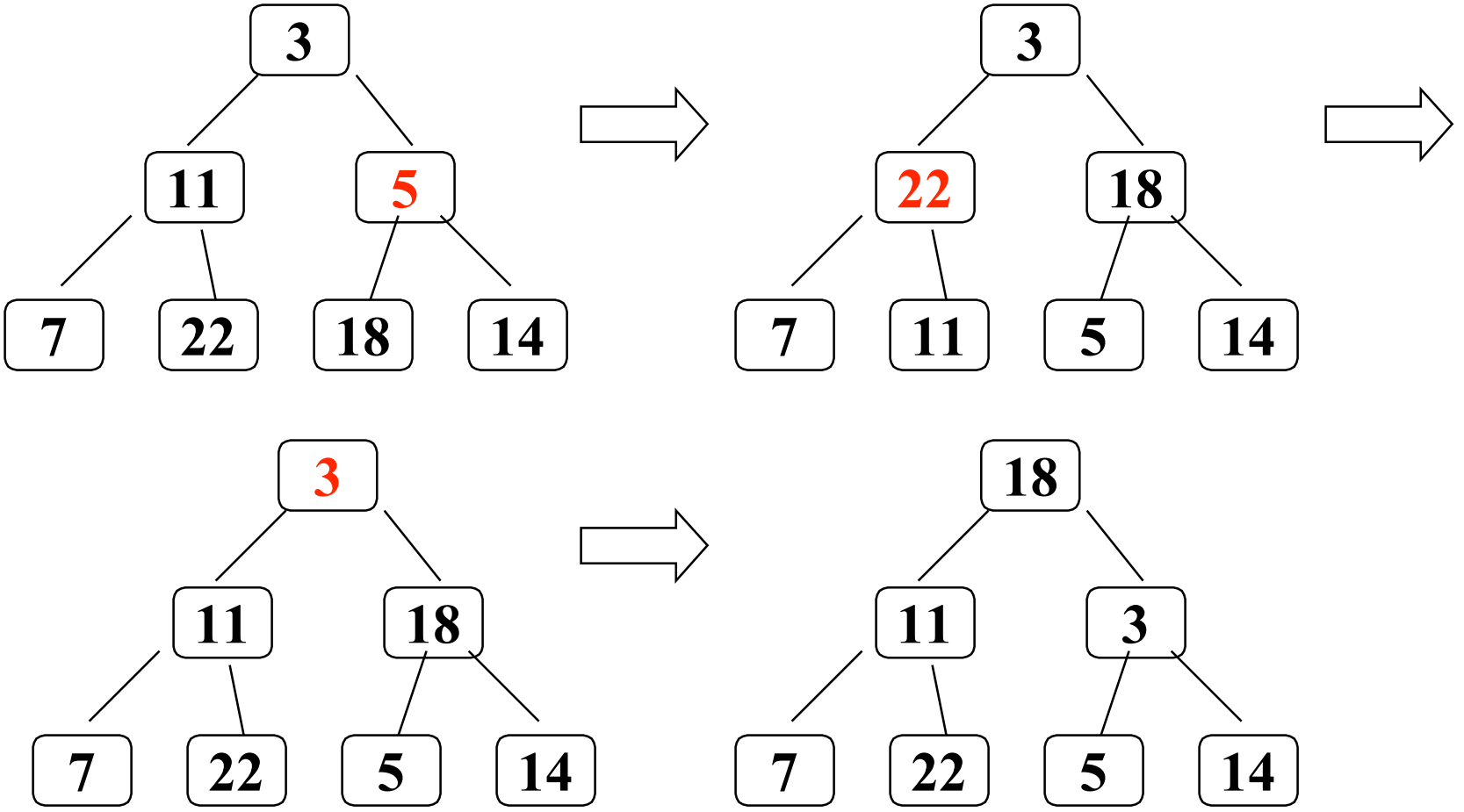


0	1	2	3	4
7	5	6	3	1

Building a Heap from an Array

- Go from last non-leaf to index 0
- At each node, do filter-down
- Work at a node is O(height of node)
- In a complete binary tree, majority of nodes close to bottom, so adds up to O(n)

Building a Heap from an Array



Sorting with trees

- **Sort with heap or AVL tree: O(n logn)**
  - **insert all the data**
  - **read in order**
    - **AVL tree: do inorder traversal**
    - **heap: remove all nodes**