Trees & More

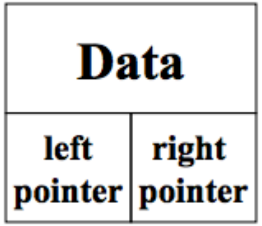
CSE 373 Help Section

Binary Tree

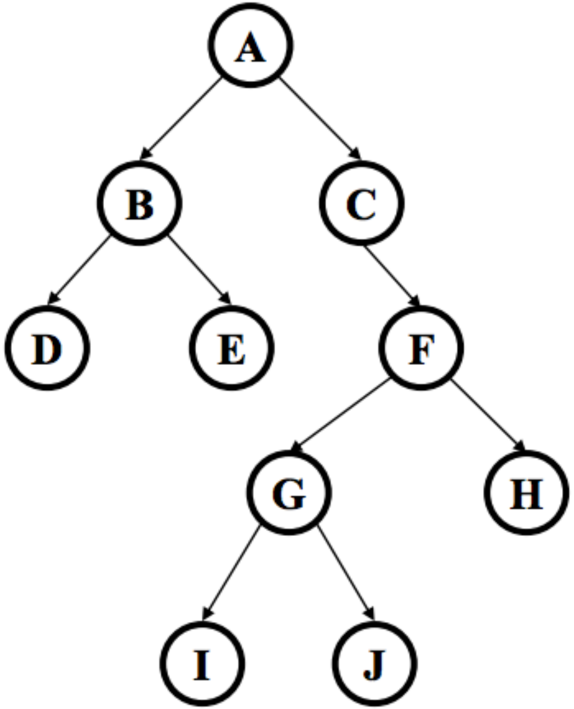
- Binary tree: Each node has at most 2 children (branching factor 2)

- Binary tree is
  - A root *(with data)*
  - A left subtree *(may be empty)*
  - A right subtree *(may be empty)*

- Representation:



- For a dictionary, data will include a key and a value



Binary Tree

For binary tree of height h:

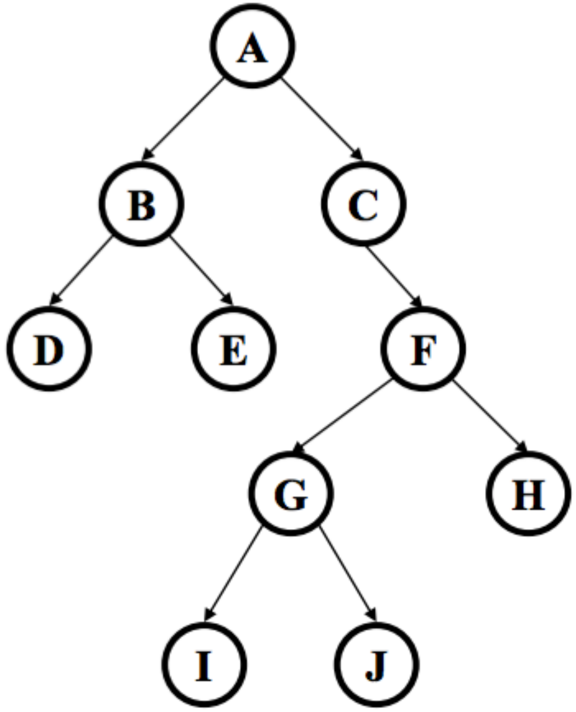
- max # of leaves: 2^h
- max # of nodes: $2^{(h+1)} - 1$
- min # of leaves: 1
- min # of nodes: h + 1

Binary Tree

Preorder: root - left - right

Inorder: left - root - right

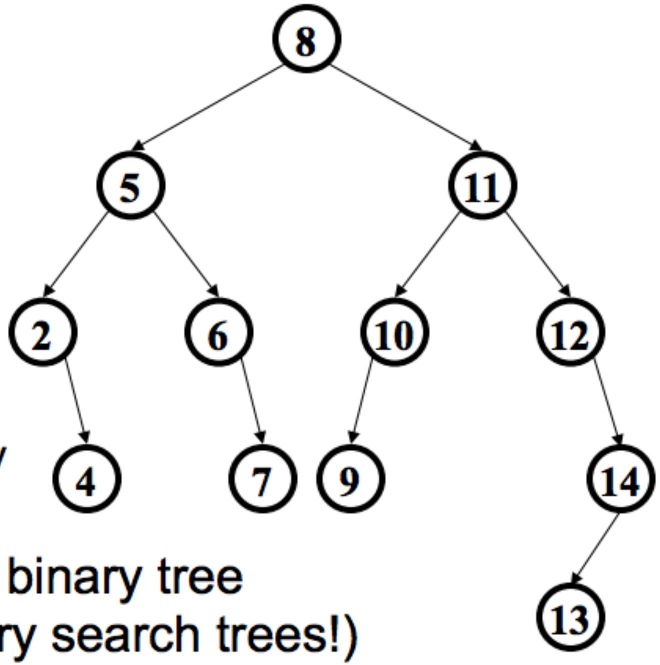
Postorder: left - right - root



Binary Search Tree

- Structure property (binary tree)
  - Each node has ≤ 2 children
  - Result: keeps operations simple

- Order property
  - All keys in left subtree smaller than node's key
  - All keys in right subtree larger than node's key
  - Result: easy to find any given key



A binary search tree is a type of binary tree
(but not all binary trees are binary search trees!)

AVL Tree

An AVL tree is a self-balancing binary search tree.

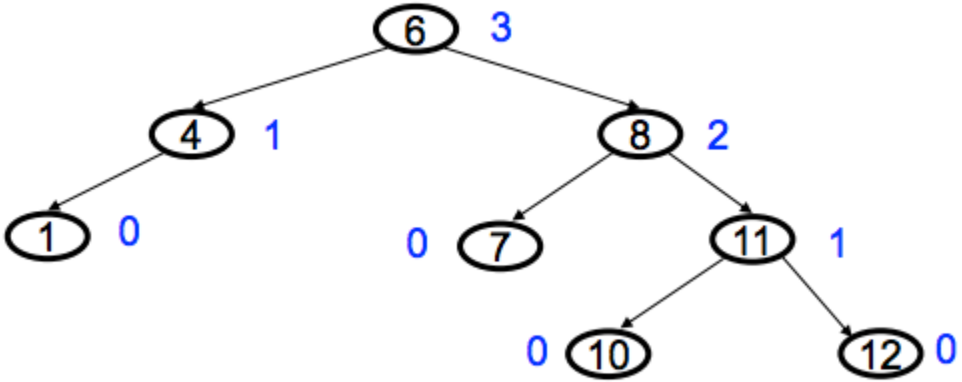
Structural properties

1. Binary tree property (same as BST)
2. Order property (same as for BST)

3. Balance property:
   balance of every node is between -1 and 1

Result: **Worst-case** depth is O(log *n*)

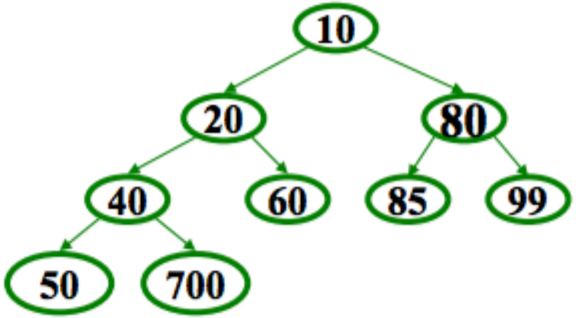
AVL Tree



Balance: Left and right subtrees of every node have heights differing by at most 1

Priority Queue & Heap

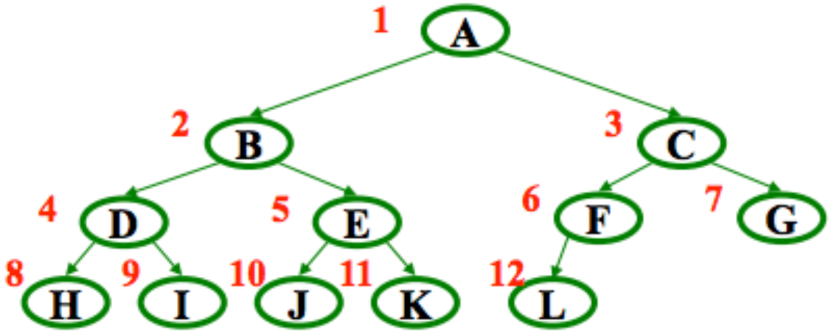
- `findMin`: return `root.data`
- `deleteMin`:
  1. `answer = root.data`
  2. Move right-most node in last row to root to restore structure property
  3. "Percolate down" to restore heap property
- `insert:`
  1. Put new node in next position on bottom row to restore structure property
  2. "Percolate up" to restore heap property



Overall strategy:

- *Preserve structure property*
- *Break and restore heap property*

Priority Queue & Heap



From node `i`:

left child: `i*2`
right child: `i*2+1`
parent: `i/2`

(wasting index 0 is convenient for the index arithmetic)

implicit (array) implementation:

	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13