BINARY SEARCH TREES

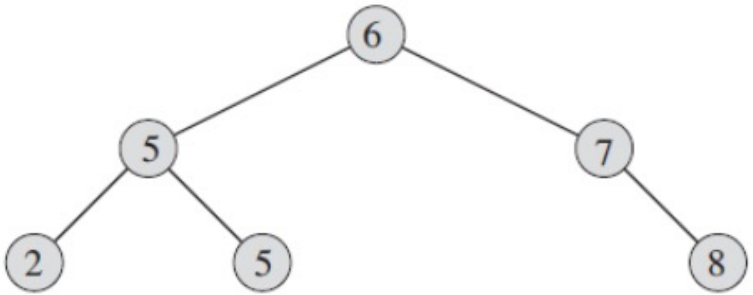
CS340

Binary Search Trees

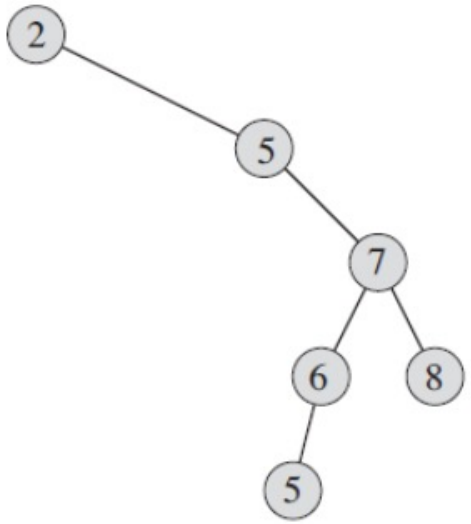
- Linked data structure where each **node** is an object
- Each node contains:
  - **key** (+ other data)
  - pointers to **left child**, **right child**, and **parent**
- Satisfies the **binary search tree property**:
  - If y is a node in the left subtree of x, then y.key ≤ x.key.
  - If y is a node in the right subtree of x, then y.key ≥ x.key.
- Binary search property is **recursively true** for every node

Two binary search trees

- representing the same data



(a)



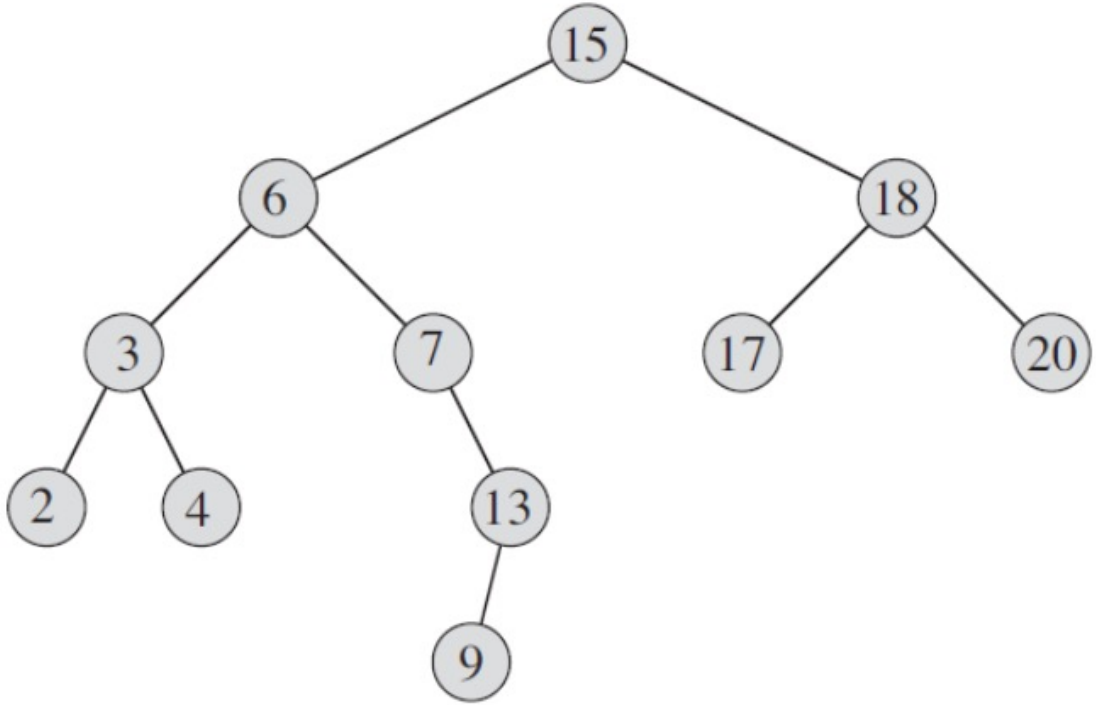
(b)

Printing keys in sorted order

- Use an inorder walk
  - Recursively walks left node, then root, then right node
  - Visits each node once. Θ(n)
  - (preorder walk prints root node first, postorder prints root node last)

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2      INORDER-TREE-WALK(x.left)
3      print x.key
4      INORDER-TREE-WALK(x.right)
```

Inorder walk

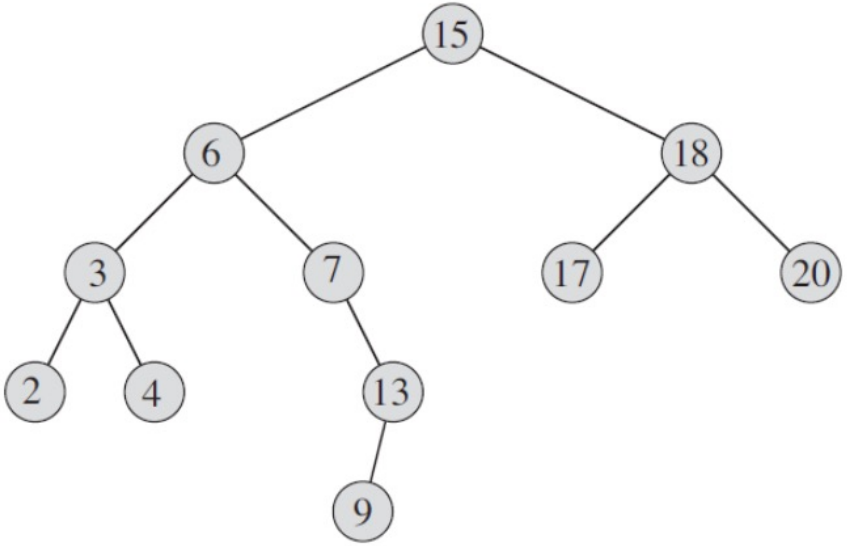


Binary search tree methods

- SEARCH
- MINIMUM
- MAXIMUM
- SUCCESSOR
- PREDECESSOR
- all O(h) on a tree of height h

SEARCH

- Follow left or right based on less than or greater than




```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2        return x
3  if k < x.key
4        return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2        if k < x.key
3              x = x.left
4        else x = x.right
5  return x
```

Finding minimum and maximum

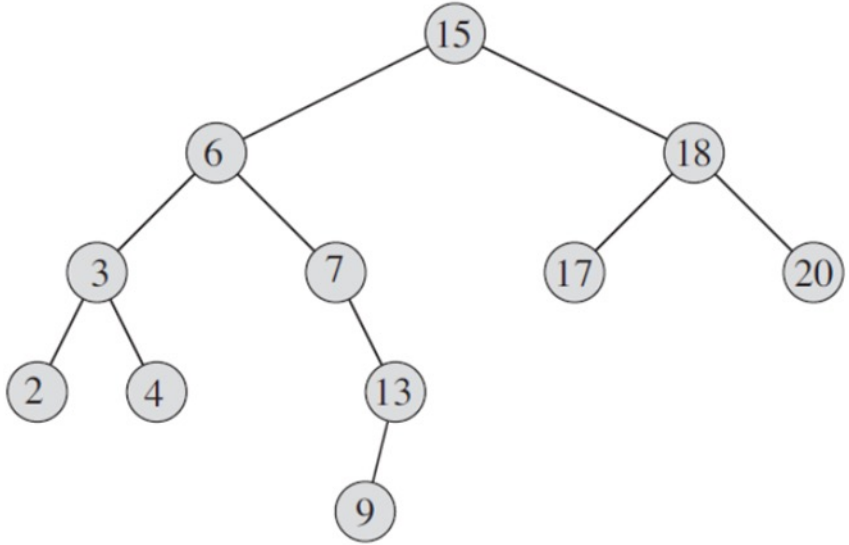
Where are min and max?

min(node x) {

}

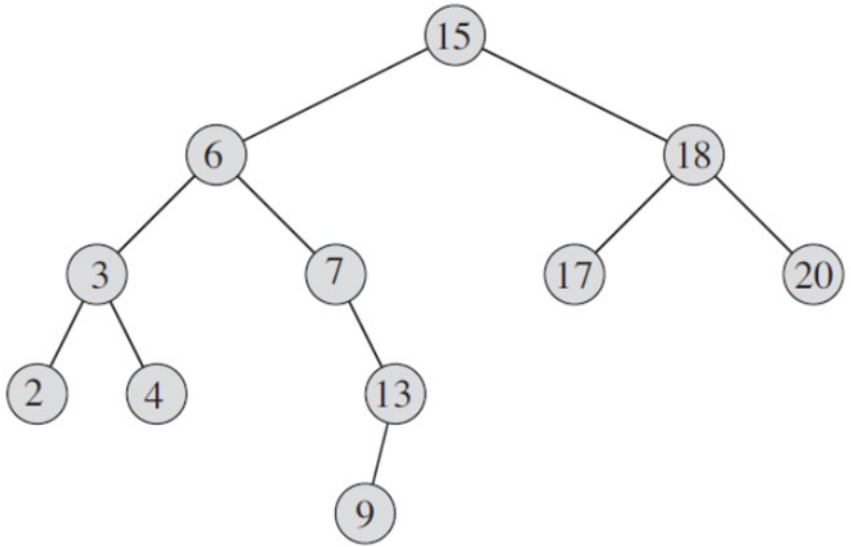
max(node x) {

}



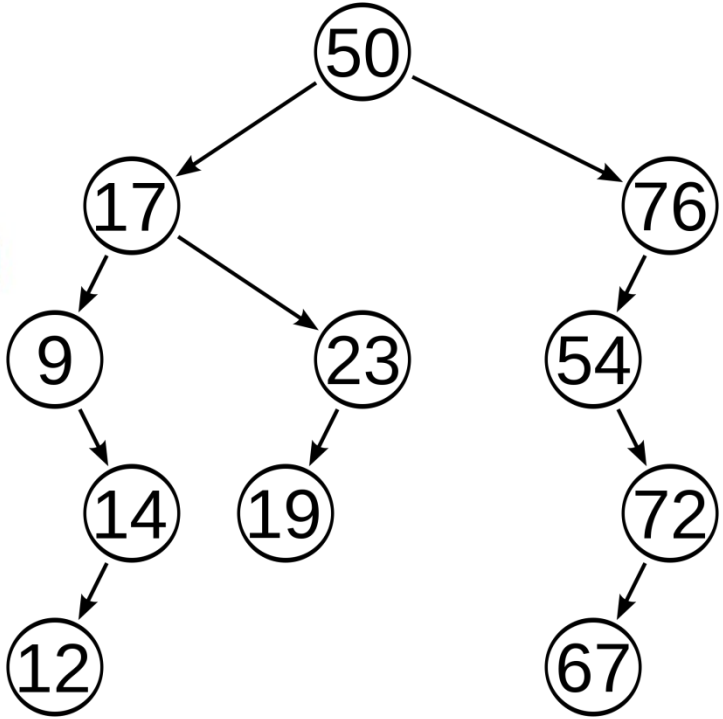
Finding the successor

- If right subtree exists, choose min of right subtree
- else successor is the lowest ancestor of x whose left child is also an ancestor of x.



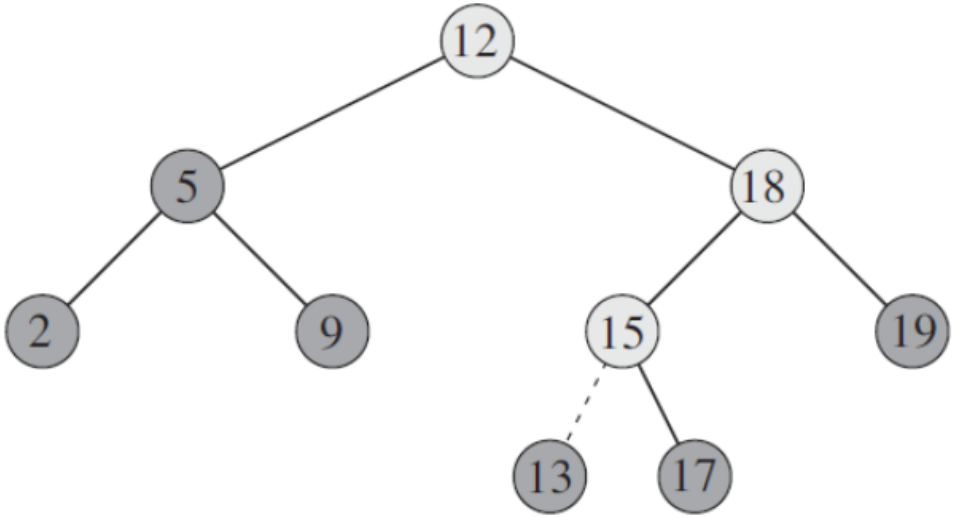
Finding the successor

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2        return TREE-MINIMUM(x.right)
3  y = x.p
4  while y ≠ NIL and x == y.right
5        x = y
6        y = y.p
7  return y
```



Inserting a node

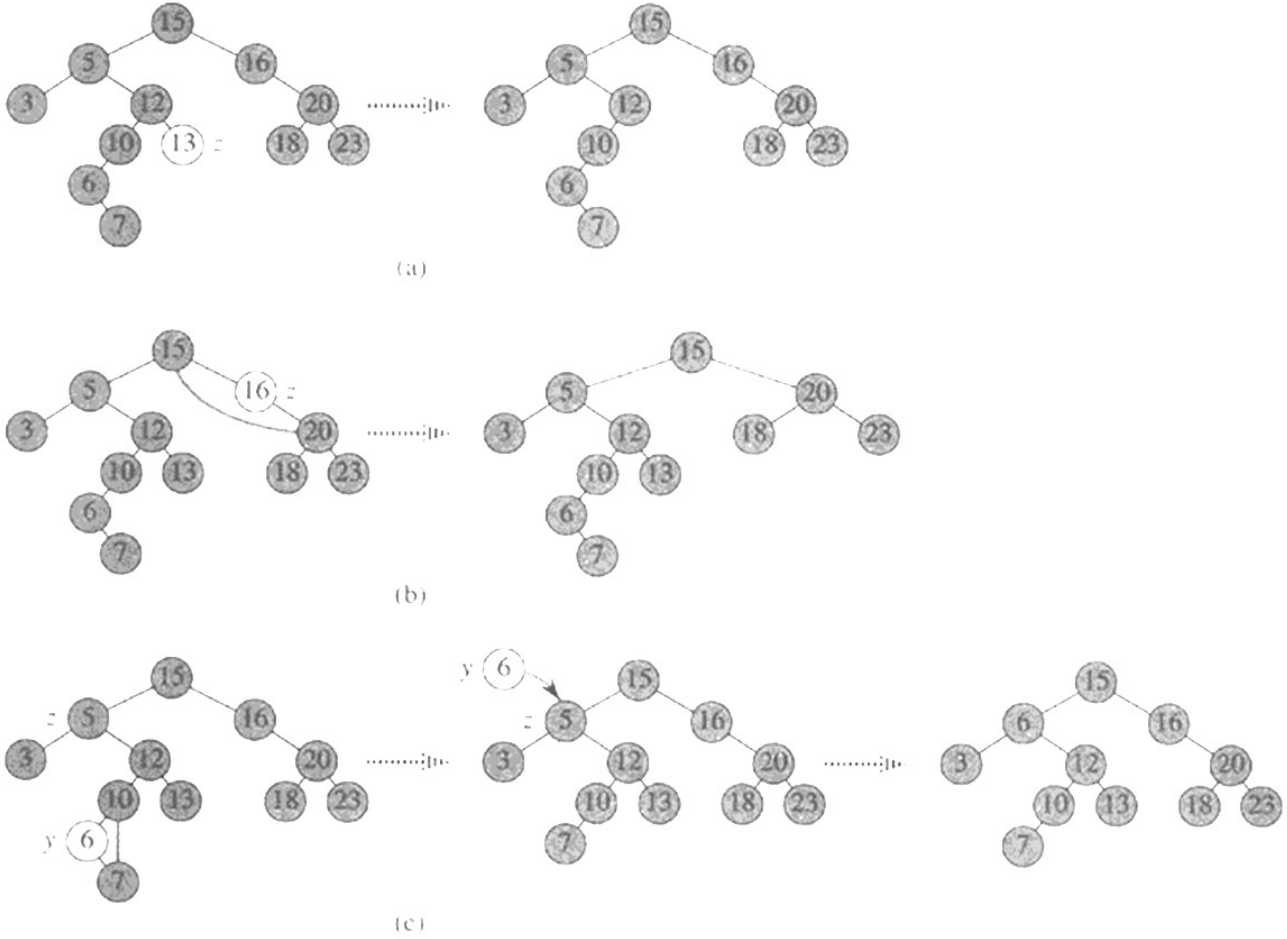
```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4        y = x
 5        if z.key < x.key
 6              x = x.left
 7        else x = x.right
 8  z.p = y
 9  if y == NIL
10        T.root = z          // tree T was empty
11  elseif z.key < y.key
12        y.left = z
13  else y.right = z
```



O(h)

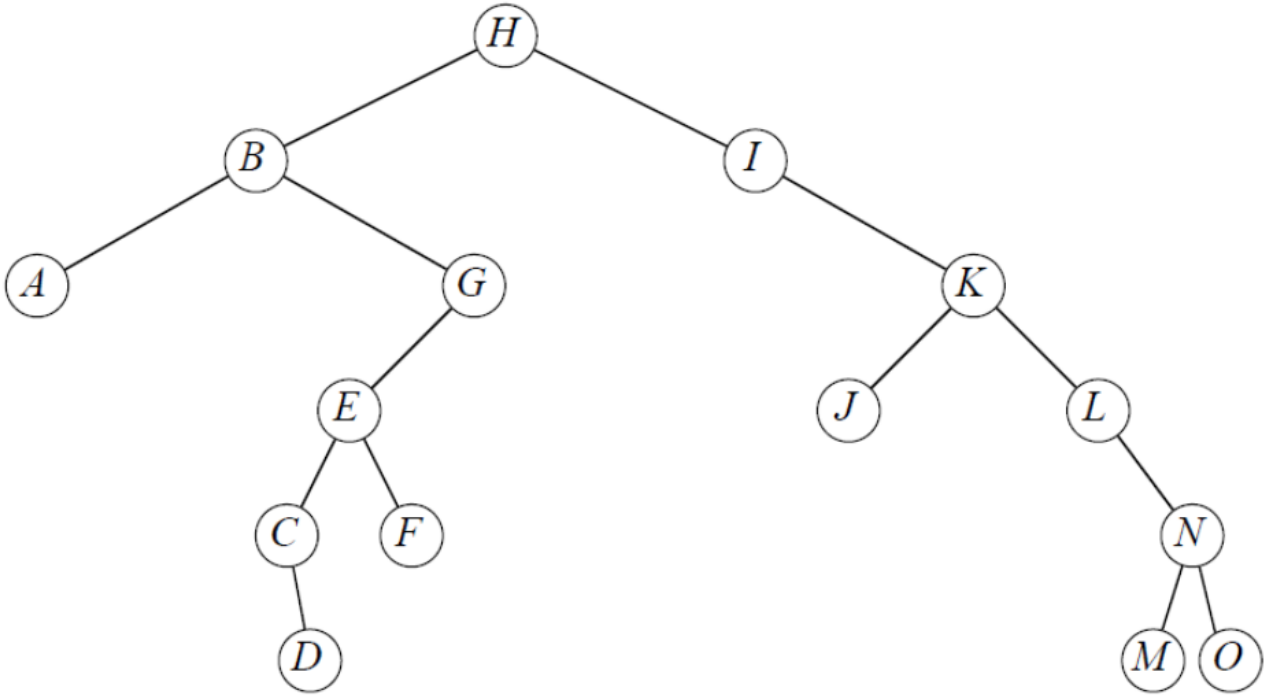
Delete z

- No children, remove z
- 1 child, splice it out
- 2 children, splice out (y) successor, then replace z key with successor key



Practice Delete

Tree-Delete(T, I)
Tree-Delete(T, G)
Tree-Delete(T, K)
Tree-Delete(T, B)



How big is h?

- The expected height of a randomly built binary search tree on n distinct keys is O(lg n)
- How to avoid an n-height tree?