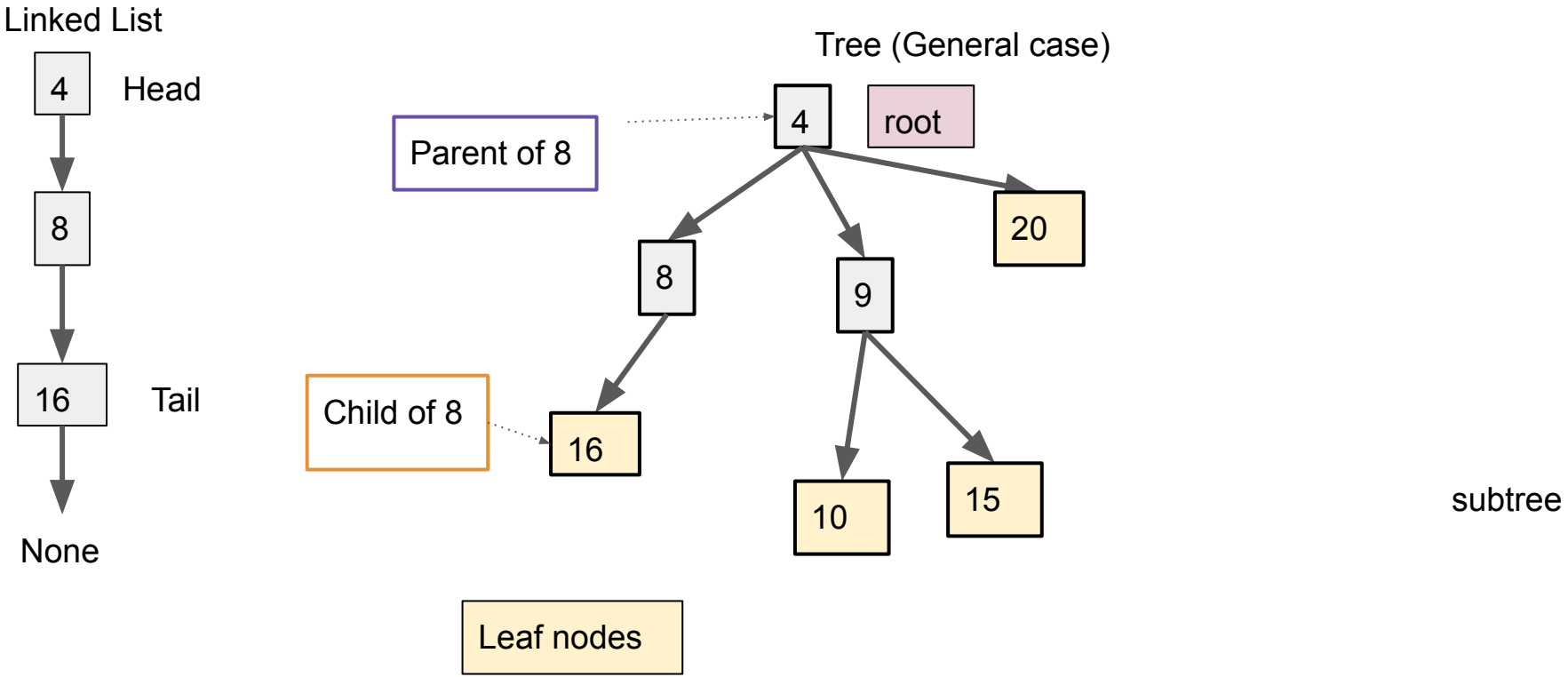
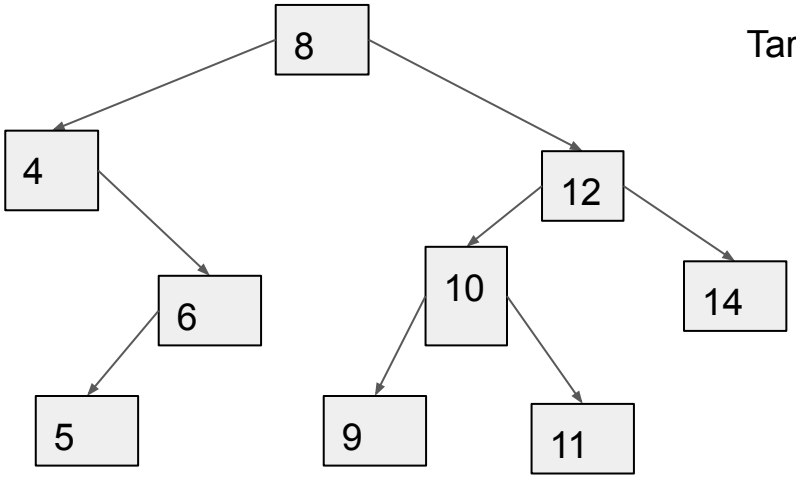

Linked List
4
Head
8
16
Tail
None
Tree (General case)
Parent of 8
4
root
8
9
20
Child of 8
16
10
15
subtree
Leaf nodes

Binary Search Tree



Target = 10

Binary: each node can point to at most 2 nodes

- Left
- Right

Search:

1) All values **less than** the root.value are on the **left** of the root
2) All values **greater than** the root.value are on the **right** of the root
3) All subtrees in the BST are also valid BSTs

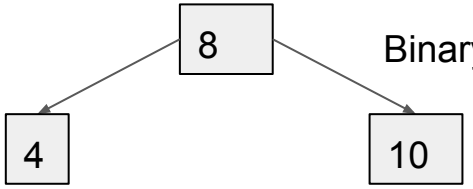
[8, 1, 12, 6, 10, 14, 5, 9, 11]
Searching: O(n)

Searching in a BST: O(log n) / logarithmic:

The number of times we have to divide by 2 / split in half until we get to 1 (or whatever base case)

If we had 1,000,000 ~ 20 comparisons on average

[4, 5, 6, 8 , 9, 10, 11, 12, 14]



Binary Search Tree

Target = 4

Contains:

- Compare target value to node.value
- **If target == node.value:**
  - **return True**
- If target > node.value:
  - Go right
  - If node.right is None:
    - We've traversed the tree and haven't found it
    - **return False**
  - Else:
    - Do the same thing
    - **return node.right.contains(target)**
- Else if target < node.value
  - Go Left
  - If node.left is None:
    - **return False**
  - Else:
    - Do the same thing
    - **return node.left.contains(target)**

Inserting:

- Compare target value to node.value
- If target > node.value:
  - Go right
  - If node.right is None:
    - Create the new node there
  - Else:
    - Do the same thing
    - Insert target into node.right
- Else if target < node.value
  - Go Left
  - If node.left is None:
    - Create node
  - Else:
    - Do the same thing
      - (compare, go left or right)
    - Insert target into node.left