COMP26120 Algorithms and Complexity
Topic 2: Data Structures

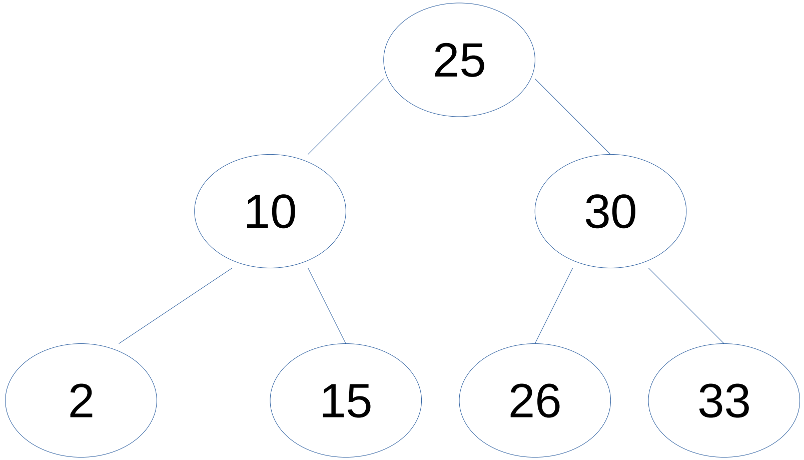
Binary Search Trees

Dr. Thomas Carroll
thomas.carroll@manchester.ac.uk
All information on Blackboard

Learning Outcomes

- Understand the concept of Binary Search Trees
- Understand complexities related with BST
- Be able to insert, search, and remove from BST

Binary Search Tree

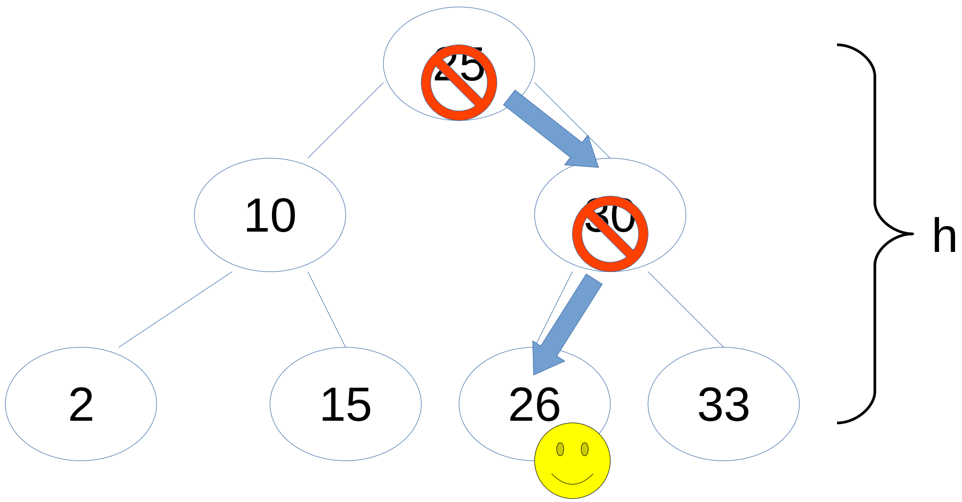


Invariant: leftChild.element() < node.element() < rightChild.element() less than or equal and and greater than or equal

In-Order traversal of BST gives us the elements in **ascending** order

Search in BST

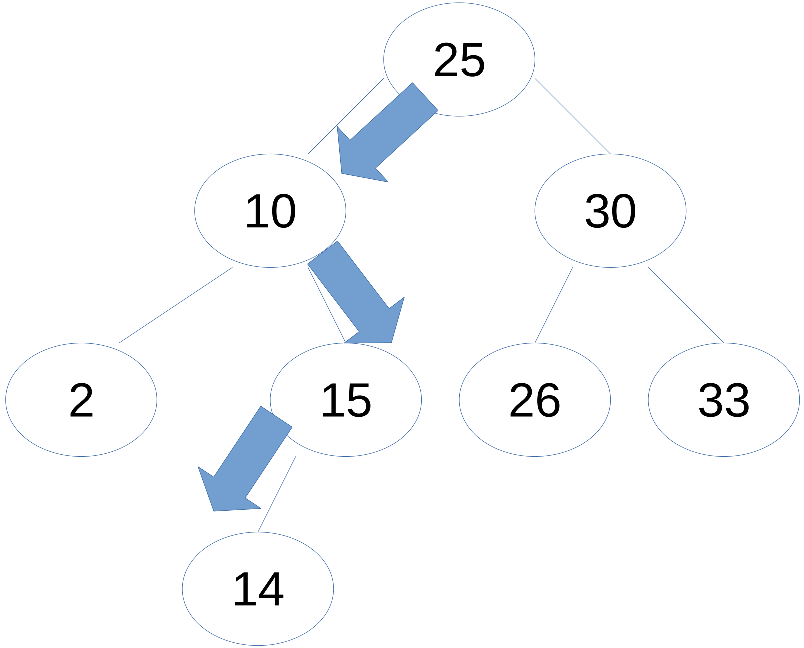
Find: 26



O(h)

Insert in BST

Insert: 14

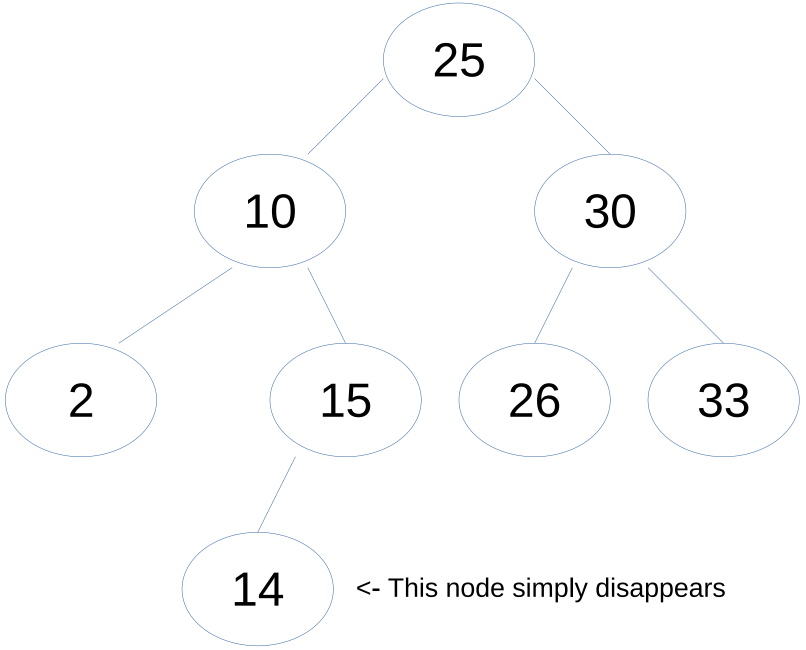


Remove in BST (Case 1)

Remove: 14

Leaf node:
Just remove it!

Keep the Invariant:
Left < node < right

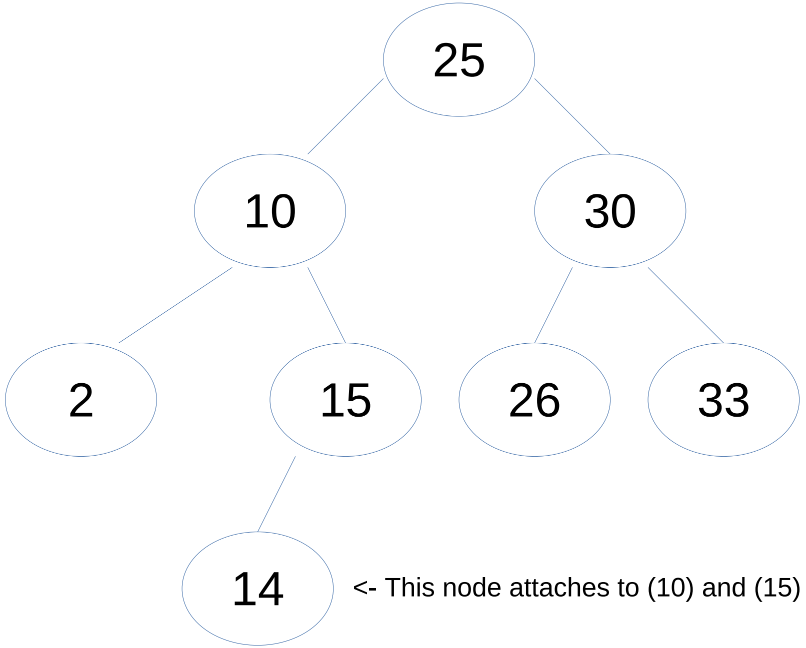


Remove in BST (Case 2)

Remove: 15

Single Child:
Elevate the Child

Keep the Invariant:
Left < node < right



<- This node attaches to (10) and (15) disappears

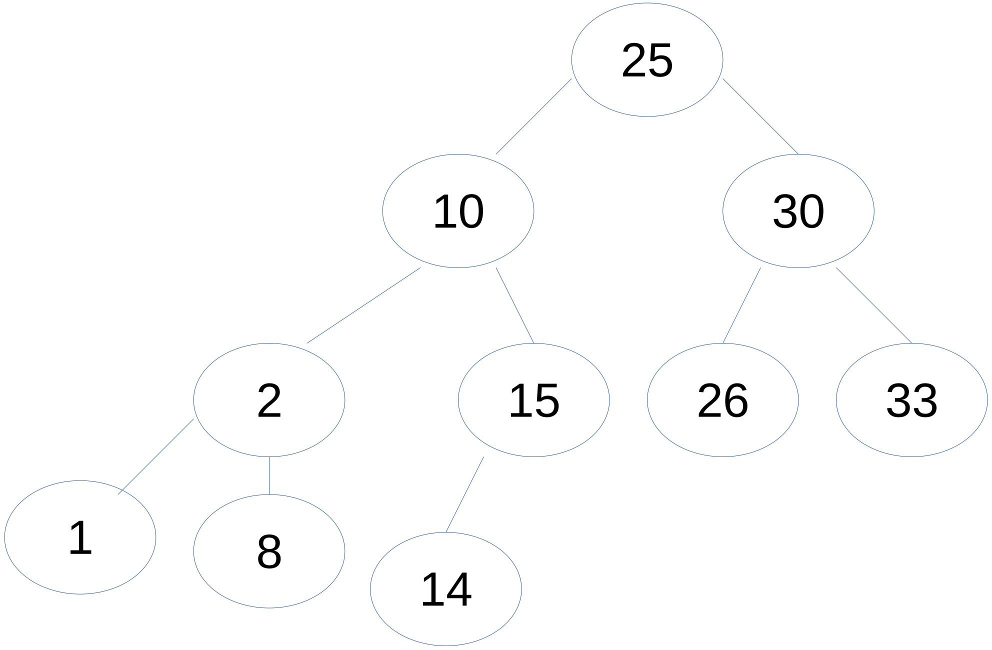
Remove in BST (Case 3)

Remove: 10

Both left and right child

1) Take 10’s *in-order successor*
2) Pace in 10’s original position

Keep the Invariant:
Left < node < right



In this case (14) takes the place of (10). This is because if we do in order traversal of the tree, the next node visited after (10) would be 14

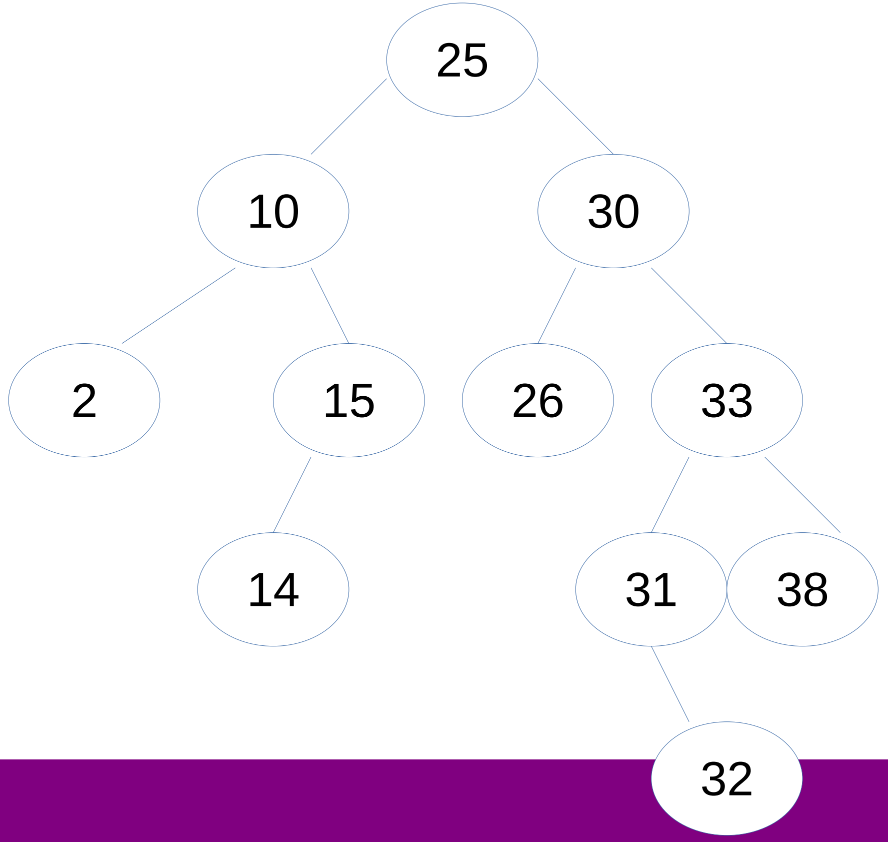
Recall in order traversal of a BST gives you the values in ascending order

Remove in BST (Case 4)

Remove: 30

In-order successor is within a subtree

1) Take 30’s *in-order successor (31)*
2) Replace 31 with 31’s Left Child
3) Replace 30 with its in-order successor, 31



Average Complexities of BST

- Assuming uniform distribution of elements:

Find	O(log n)
Insert	O(log n)
Delete	O(log n)

logn because each time
we make a left or right
decision we halve the number
of elements we're searching

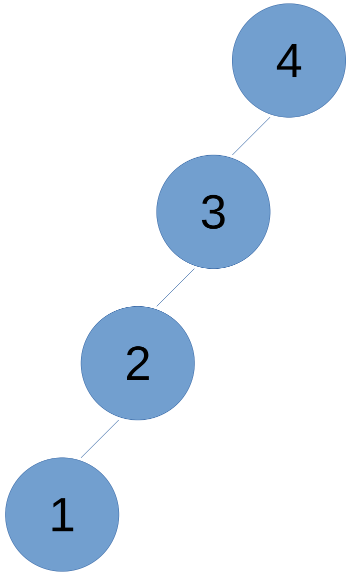
insert and delete are also
logn because you must
use find.

Worst Case Complexities of BST

Find	~~O(h)~~
Insert	~~O(h)~~
Delete	~~O(h)~~

O(n)

Insert:
- 4
- 3
- 2
- 1



Height of the binary search tree is O(n)!