School of Computer Science

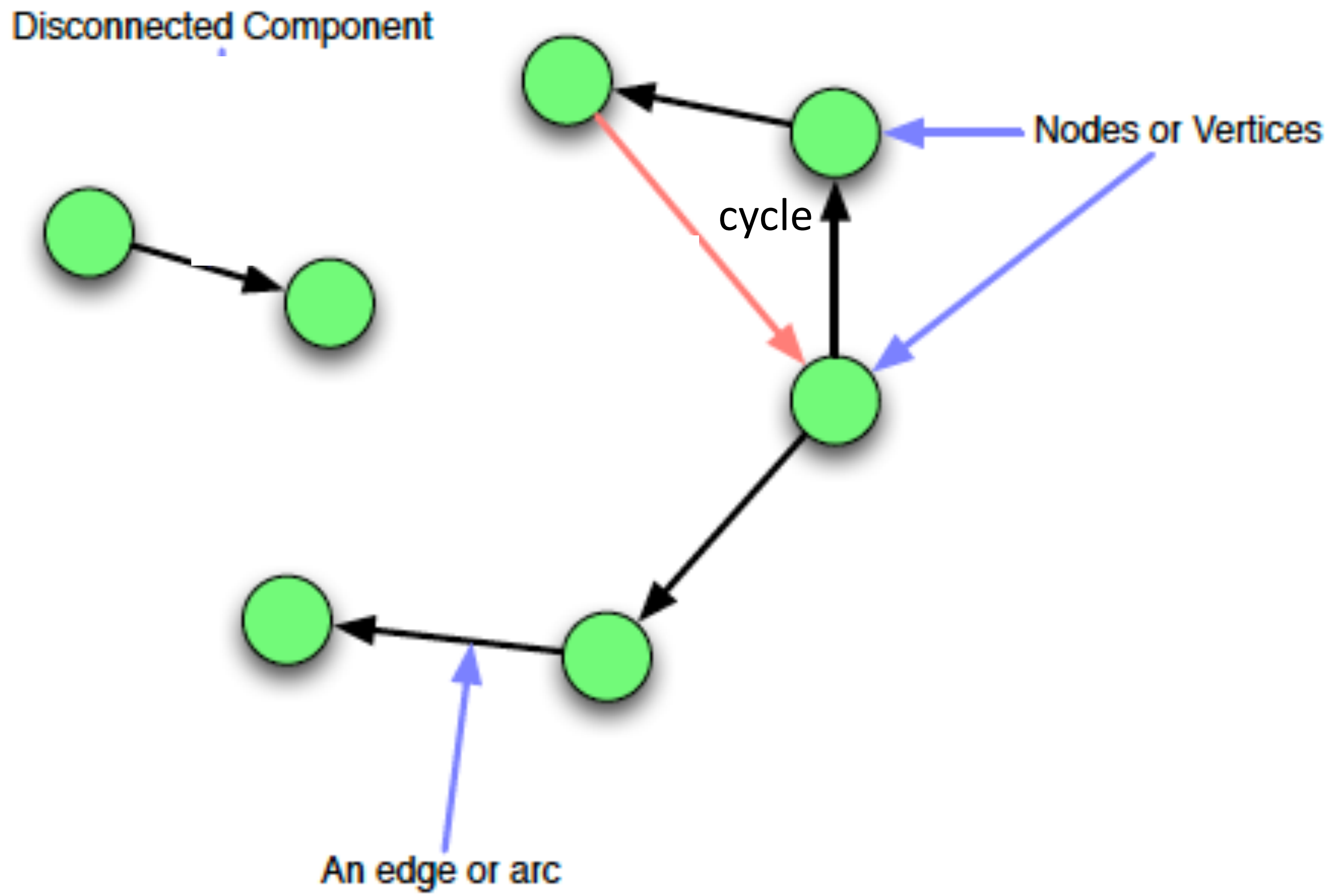
COMP SCI 1103/2103 Algorithm Design & Data Structure

Binary Trees

Review - Graph

- A graph is a collection of points (vertices or nodes) where some of the points are connected by line segments (edges or arcs).
- Connected or not
- Can have cycles
- G= (V,E),
  - V={v1,v2,...,vn},
  - E={e1,e2,...,e_n},
  - ei= (vj,vk)
- Directed - Undirected

Graph Example

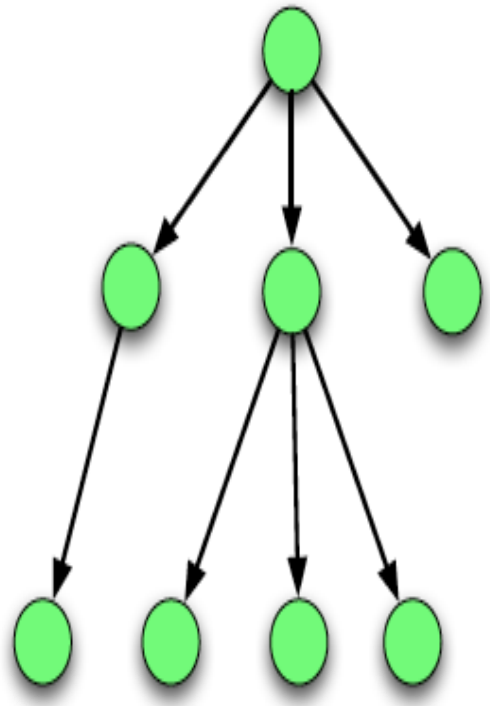


Review - Trees

- Graphs with certain properties are called trees.
- Trees are a subset of Graphs.
  - Trees must have all of their nodes connected.
  - Trees cannot contain cycles.
  - In other words, trees are connected, acyclic graphs.
- A tree can be defined in several ways. One natural way to define a tree is using recursion.
- N nodes, n-1 edges

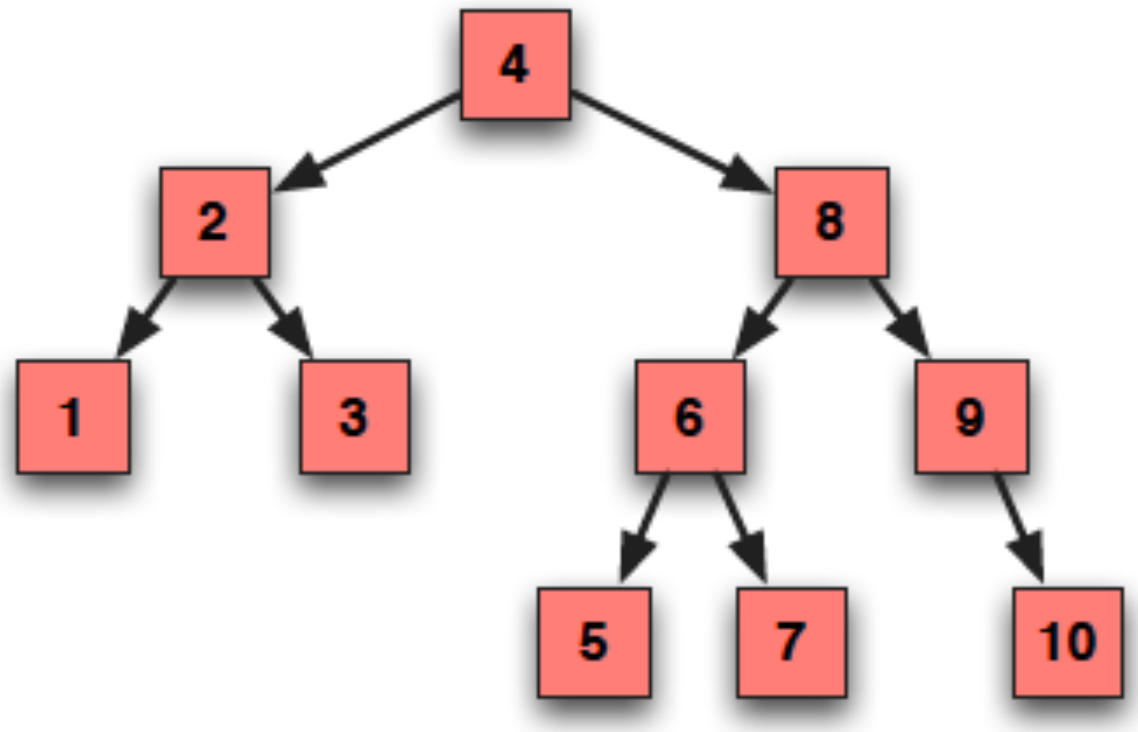
Directed Rooted Tree Terminology

- Root
- Parent- child.
- leaf.
- Depth of a node (size of the path from root).
- Height of a node (size of the longest simple path to a leaf)
- Height of the tree height of the root.



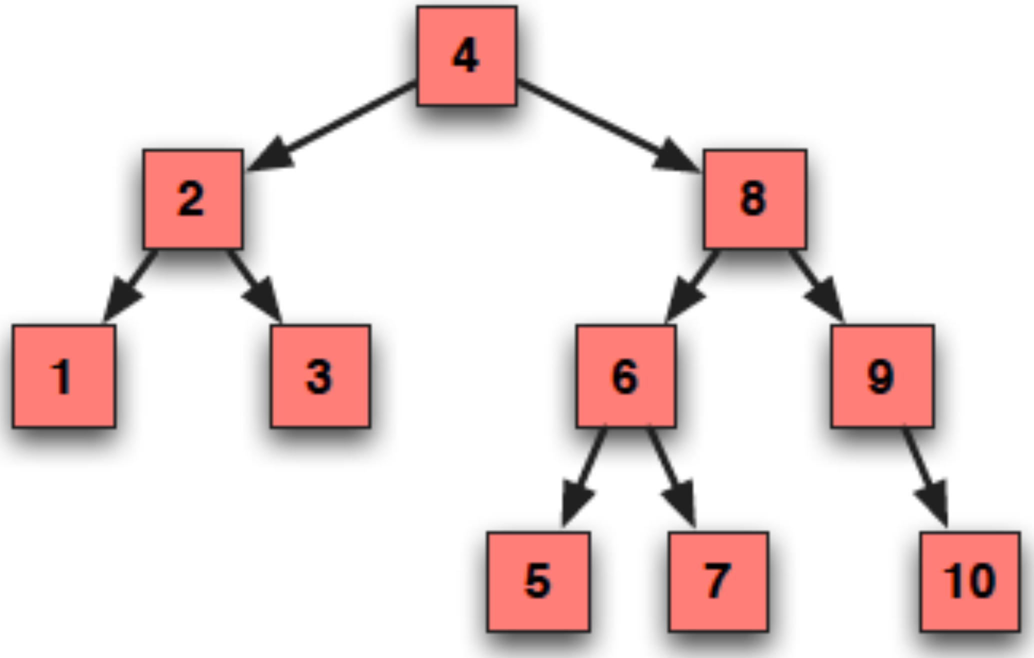
Binary Trees

- Binary Trees are trees that have 0, 1 or 2 children.



Traverse the tree

- Pre-order (Node, Left, Right)
- Post-order (Left, Right, Node)
- In-order (Left, Node, Right)
- Level-order



TreeItem

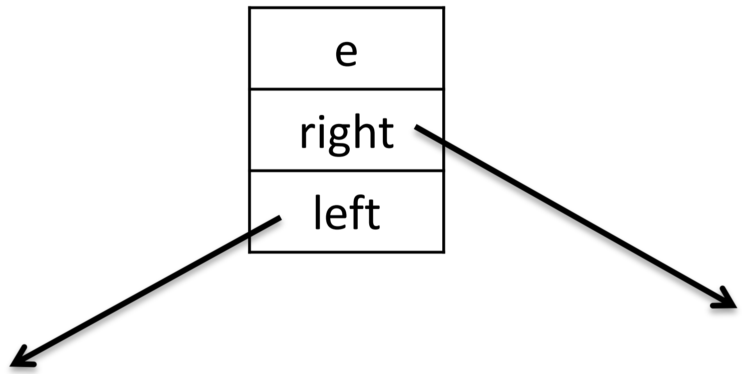
Class Handle = **Pointer to** TreeItem

Class TreeItem **of** Element

e: Element

right: Handle

left: Handle



Tree Traversal

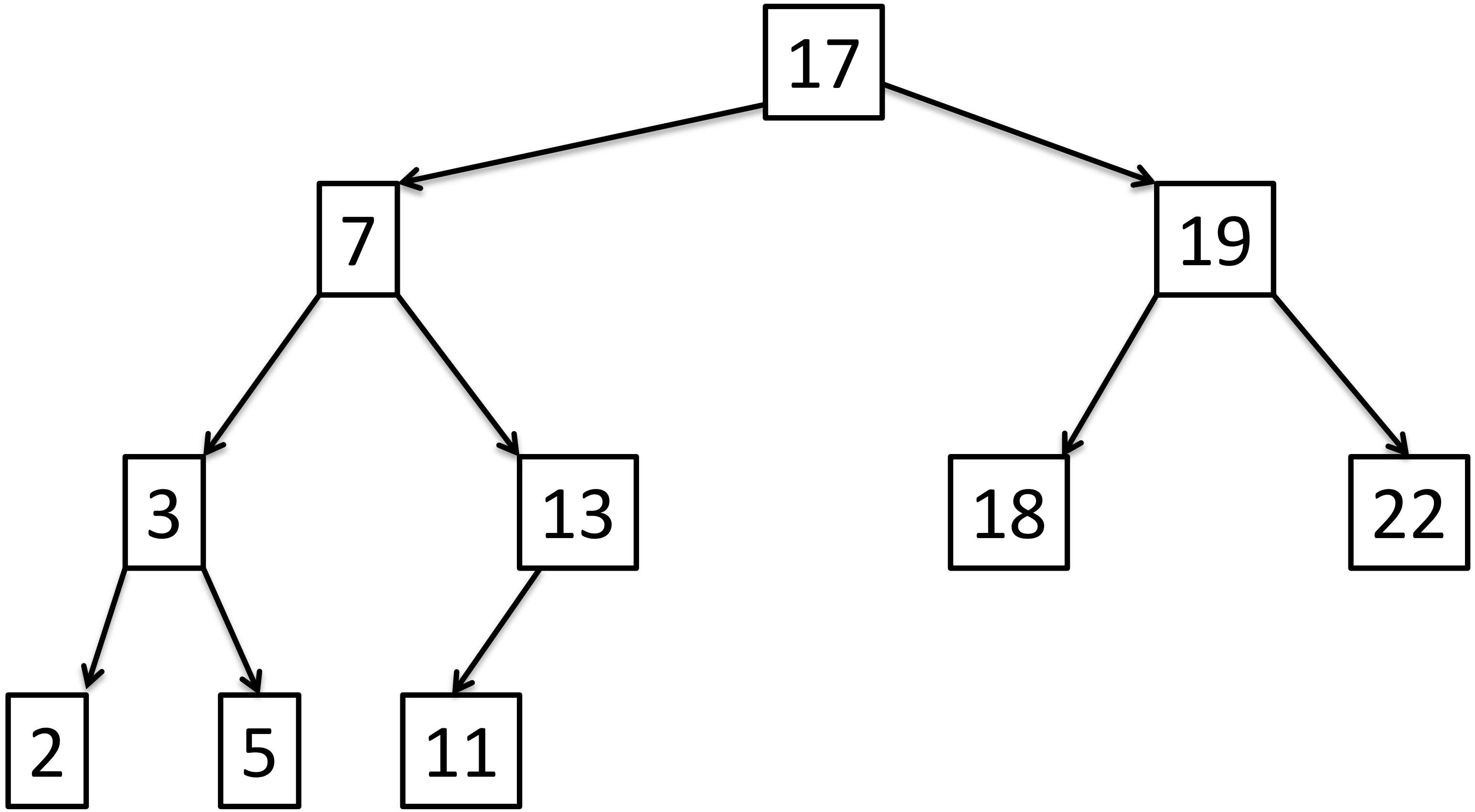
- Want to visit every node in the tree (and print out the elements).
- Recursive formulation for tree traversal

Preorder Traversal

Preorder(Tree T)

1. Visit the root (and print out the element)
2. If (T->left !=null) Preorder(T->left)
3. If (T->right !=null) Preorder(T->right)

Preorder Traversal



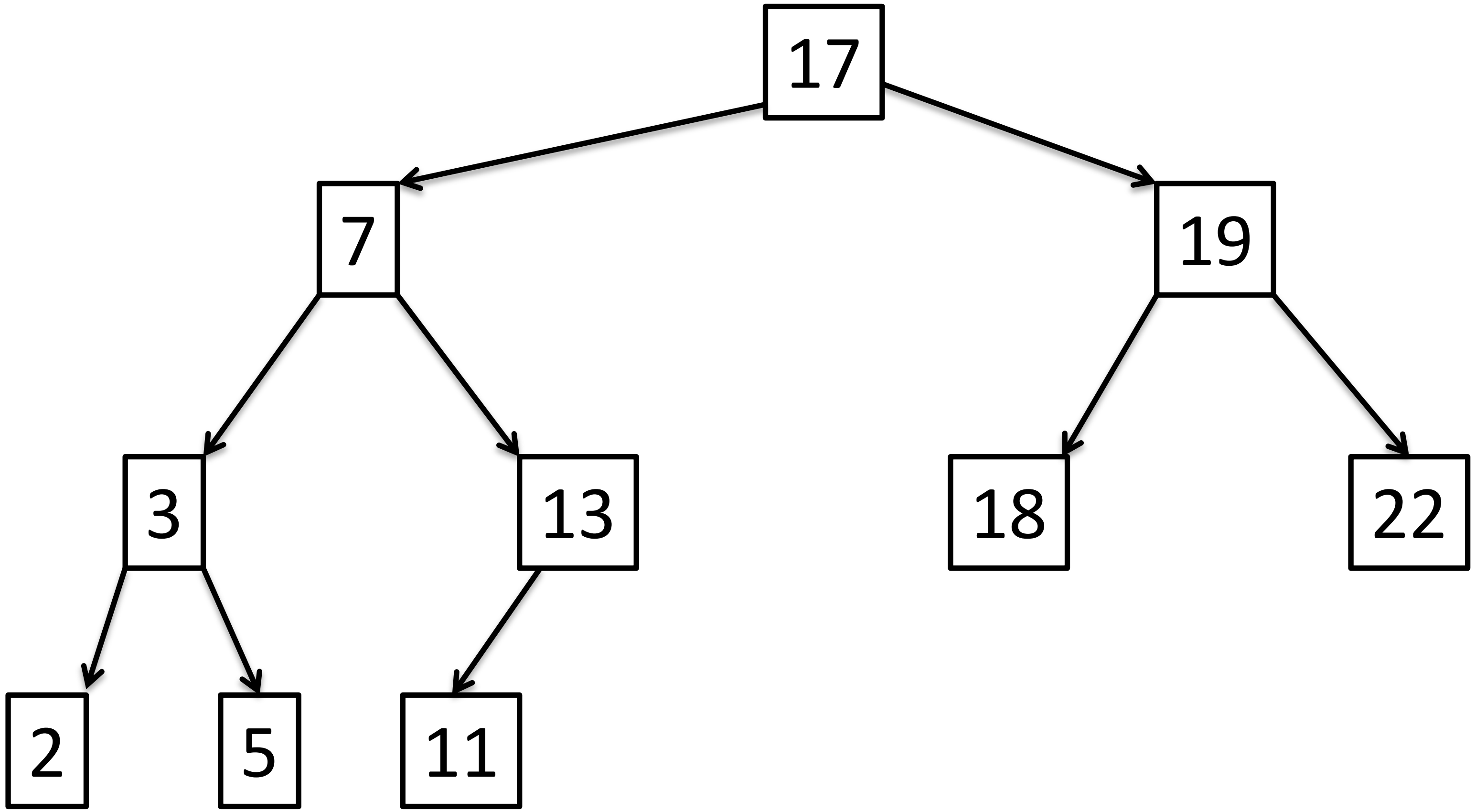
Order nodes are visited: 17, 7, 3, 2, 5, 13, 11, 19, 18, 22

Postorder Traversal

Postorder(Tree T)

1. If (T->left !=null) Postorder(T->left)
2. If (T->right !=null) Postorder(T->right)
3. Visit the root (and print out the element)

Postorder Traversal



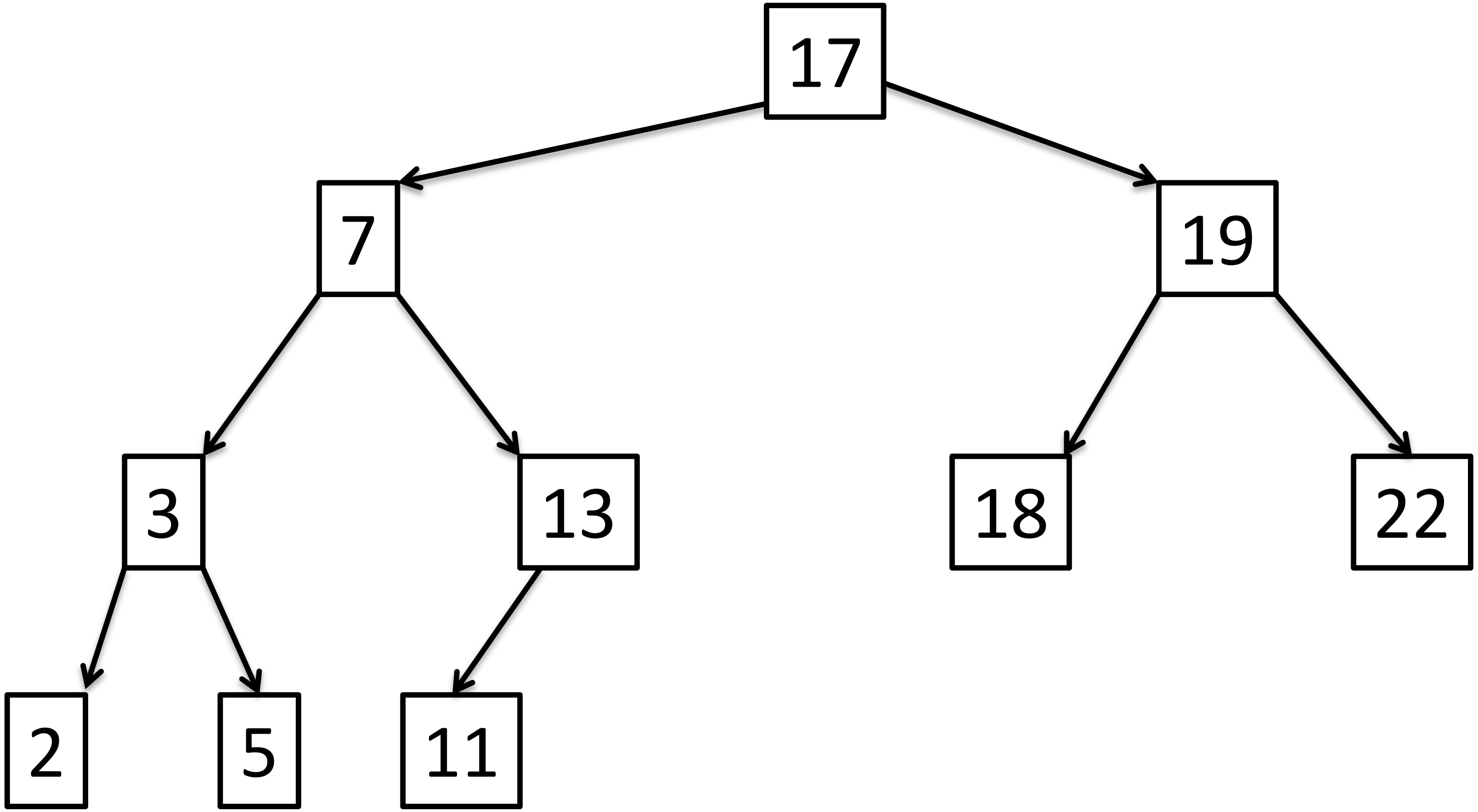
Order nodes are visited: 2, 5, 3, 11, 13, 7, 18, 22, 19, 17

Inorder Traversal

Inorder(Tree T)

1. If (T->left !=null) Inorder(T->left)
2. Visit the root (and print out the element)
3. If (T->right !=null) Inorder(T->right)

Inorder Traversal



Order nodes are visited: 2, 3, 5, 7, 11, 13, 17, 18, 19, 22

Observation: This sequence is sorted

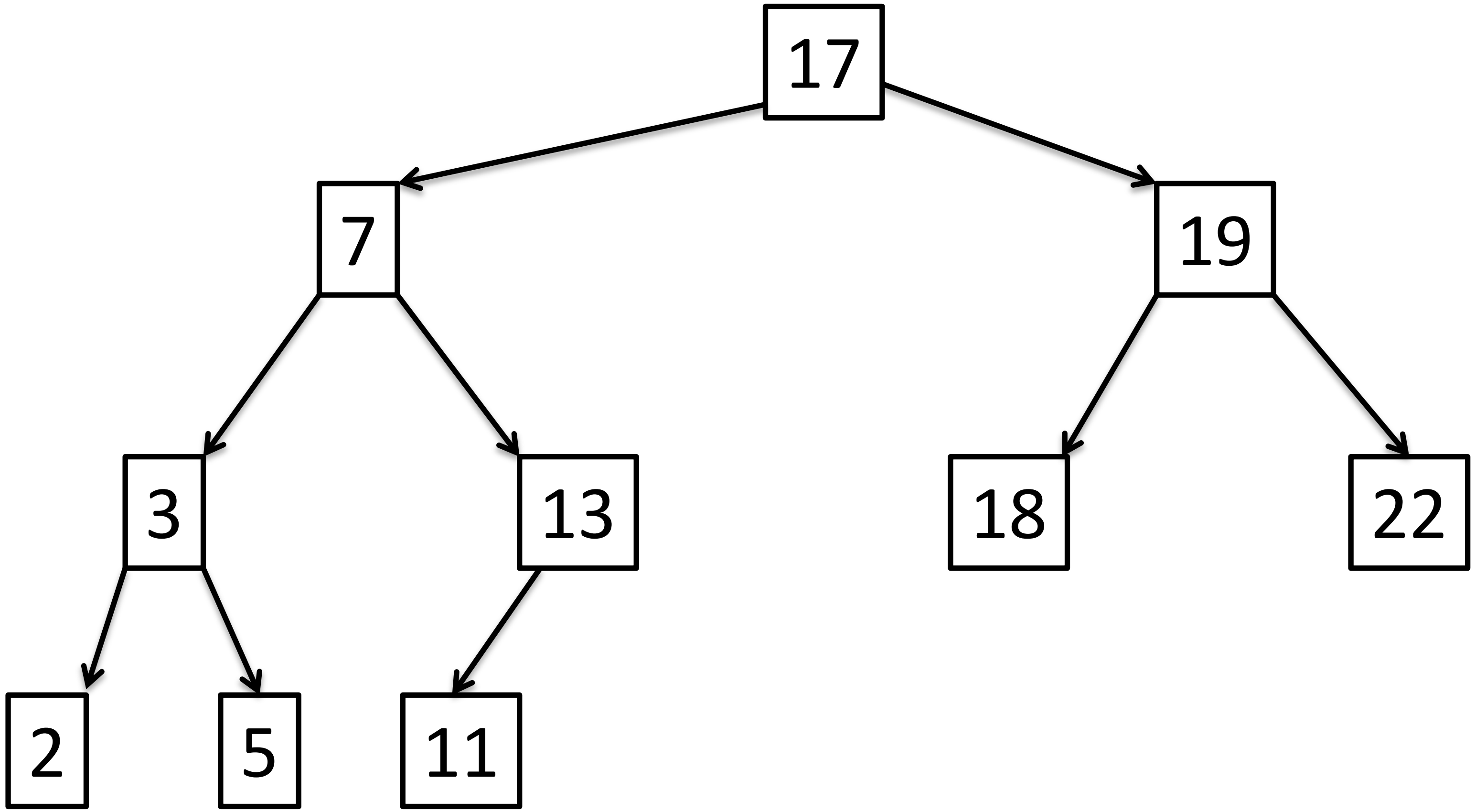
Sorted Sequences

Operations for Sorted Sequences

- Find an element e in the sorted sequence
- Insert an element e into the sorted sequence
- Delete an element e from the sorted sequence.

Want to have all these operations implemented in time $O(\log n)$.

Binary Search Tree



Sorted sequence by Inorder Traversal: 2, 3, 5, 7, 11, 13, 17, 18, 19, 22

Properties of Binary Search Trees

- All elements in the left subtree of a node k have value smaller than k.
- All elements in the right subtree of a node k have value larger than k.

Perfectly Balanced Binary Search Trees

- A binary search tree is perfectly balanced if it has height $\lfloor \log n \rfloor$ (height is the length of the longest path from the root to a leaf)

Trees, pointers and recursion

- As we saw last lecture, it can be difficult to implement our tree using a single array
- Implementing trees using nodes and recursion match our mental model better.
- How would we implement the different traversal orderings using recursion?

Example of Binary Tree

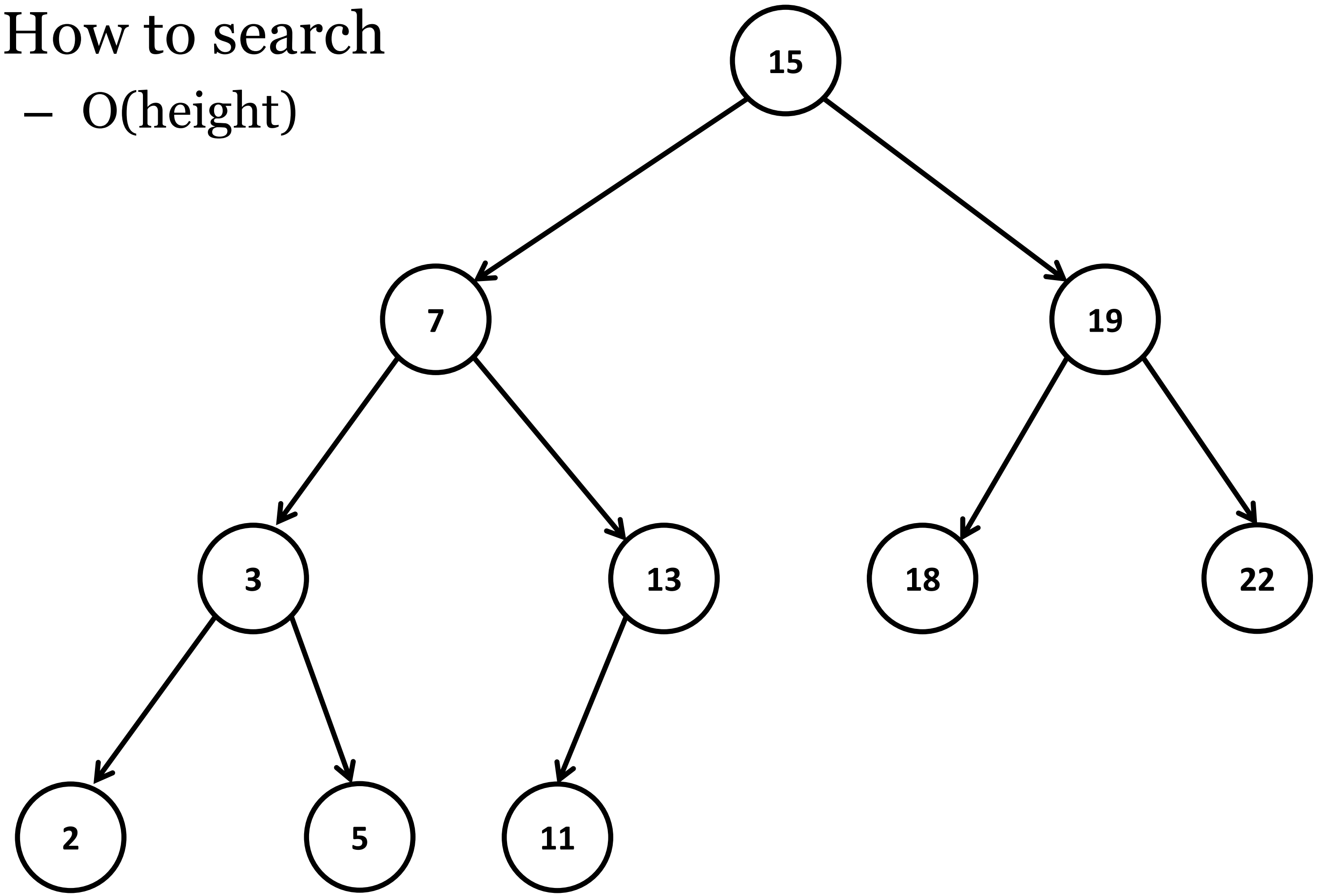
- Expression Trees
  - The leaves of an expression tree are operands and other nodes contain operators.
  - The expression trees can be binary tree since most operators are unary or binary.
- We can evaluate an expression tree T by applying the operator at the root to the values obtained by recursively evaluating the left and right subtrees.
- In-order, pre-order and post-order traverse on this tree gives us in-fix, pre-fix, and post-fix representation of arithmetic expressions
  - Find it confusing? Name the subtrees and find them recursively

Example of Binary Tree

- Expression Trees
  - Given a post-fix expression, build the tree
    - Remember how you could find the result of that expression by means of an stack?
    - Use a stack for pointers to subtrees this time.

Ordered Binary Tree (Binary Search Tree)

- Subtrees are also binary search trees.
- How to search
  - O(height)

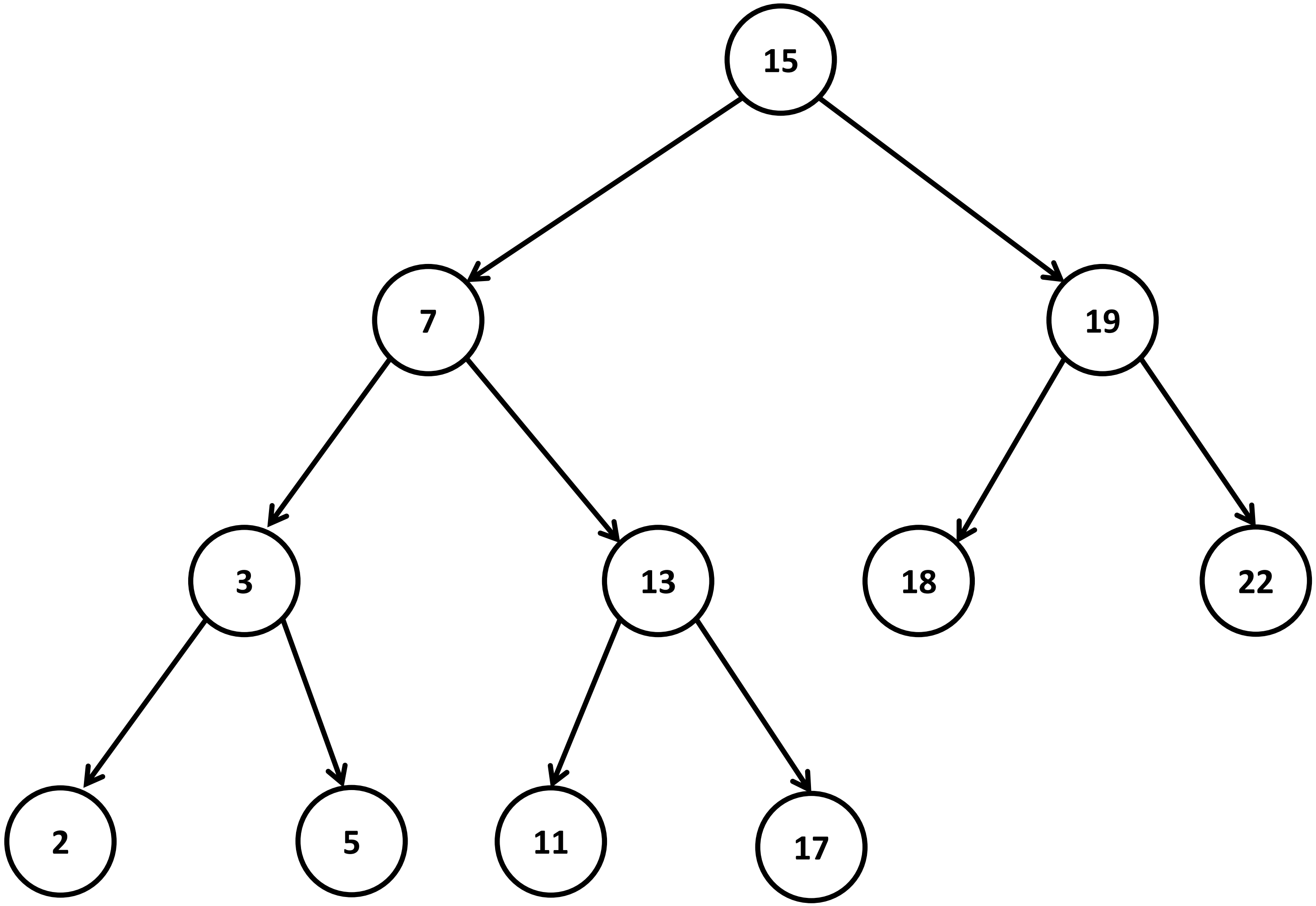


Binary Search Tree

- A binary search tree (BST) is a binary tree with the following properties:
  - Node values are distinct and comparable
  - The left subtree of a node contains only values that are *less than* the node's own value.
  - The right subtree of a node contains only values that are *greater than* the node's own value.

Binary Search Tree

- Is this tree a BST?



Binary Search Tree

- How to make this tree?
  - First think about adding a new node to it
  - We assume that the values are distinct and comparable

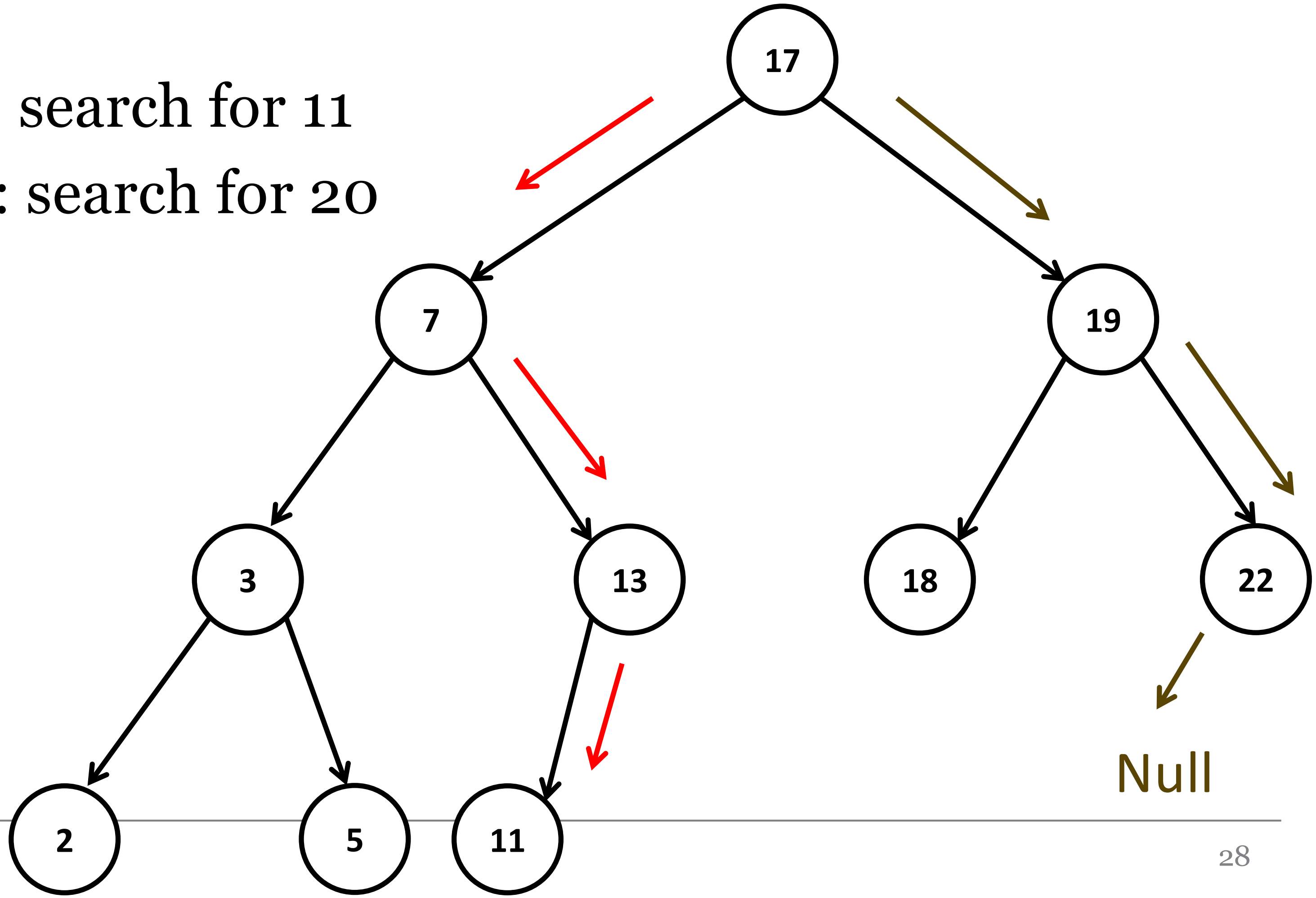
Searching

- Problem: Search whether a value exists in a dataset.
- One suitable data structure for this problem is sorted array (assuming the values are orderable).
  - Searching takes logarithmic time instead of linear time of linked list.
  - However, insertion and deletion are expensive. (Shifting array elements often takes linear time.)
- Ordered tree or Binary search tree is an easy-to-implement data structure, under which searching, insertion, and deletion **all take logarithmic time on average**.
  - All are done in O(height), but height can be $\Omega(n)$ in worst case

BST - Searching

- This operation returns true if there is a node in tree T that has value X, or false if there is no such node.

- Example 1: search for 11
- Example 2: search for 20



BST - Searching

- This operation returns true if there is a node in tree T that has value X, or false if there is no such node.

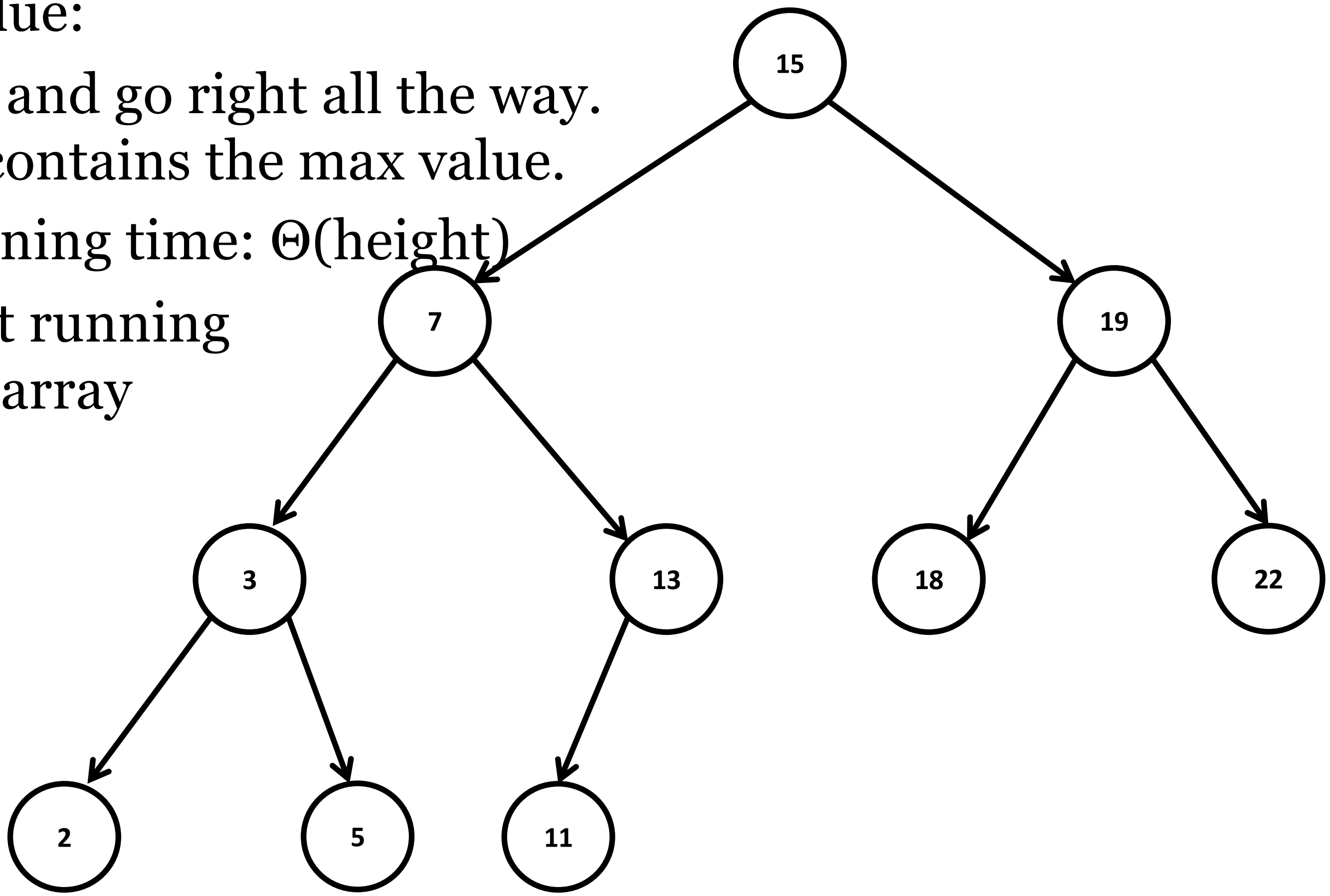
- Start from root
- If current subtree is empty, return not found
- If target value = current value, return found
- If target value < current value, go left
- If target value > current value, go right

BST - Searching

- Which of the following best describes the worst-case running time of searching under a BST with n nodes?
  - $\Theta(n)$
  - $\Theta(\log(n))$
  - $\Theta(\text{height})$
  - $\Theta(1)$

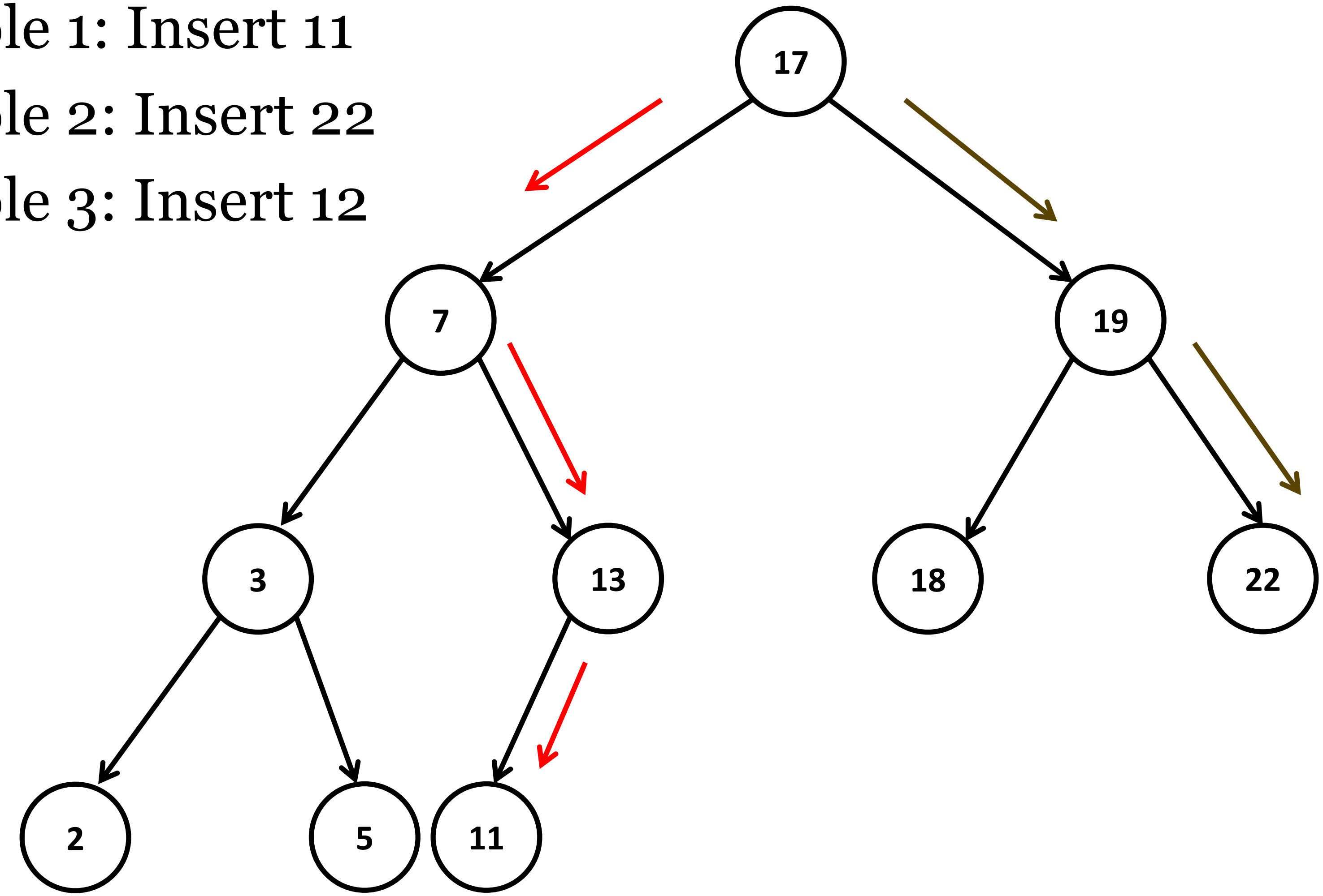
BST – Min and Max

- The operation returns the node containing the smallest or largest elements in the tree.
- To find the max value:
  - Start from root and go right all the way. The last node contains the max value.
  - Worst-case running time: Θ(height)
  - Versus constant running time for sorted array
- Similar for min.



BST - Insertion

- Example 1: Insert 11
- Example 2: Insert 22
- Example 3: Insert 12



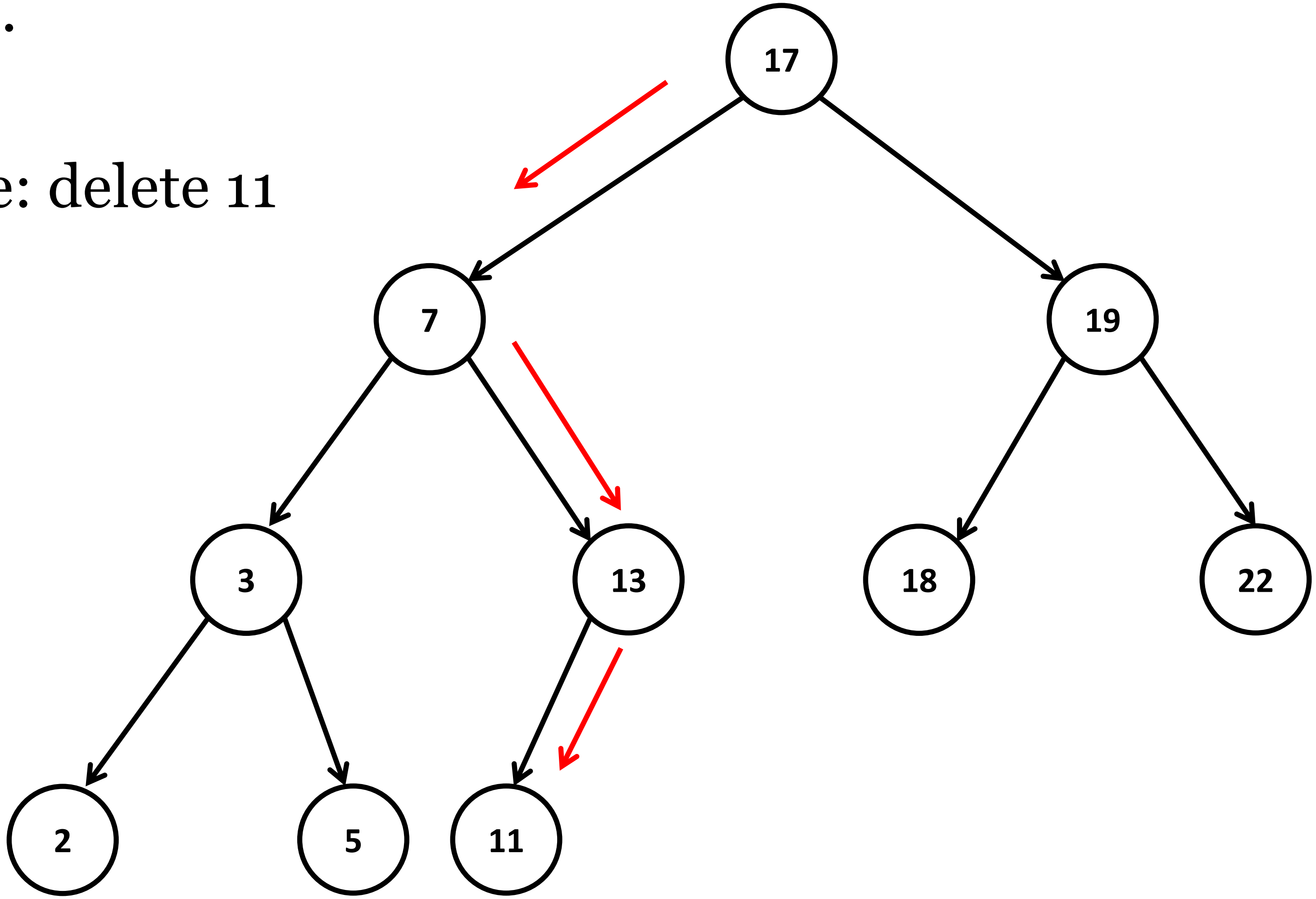
BST - Insertion

- Start from root
- If current subtree is empty, create new node here.
- If target value = current value, terminate.
- If target value < current value, go left.
- If target value > current value, go right.

- What is the worst-case running time of insertion under a BST with n nodes?
  - Θ(height)

BST - Deletion

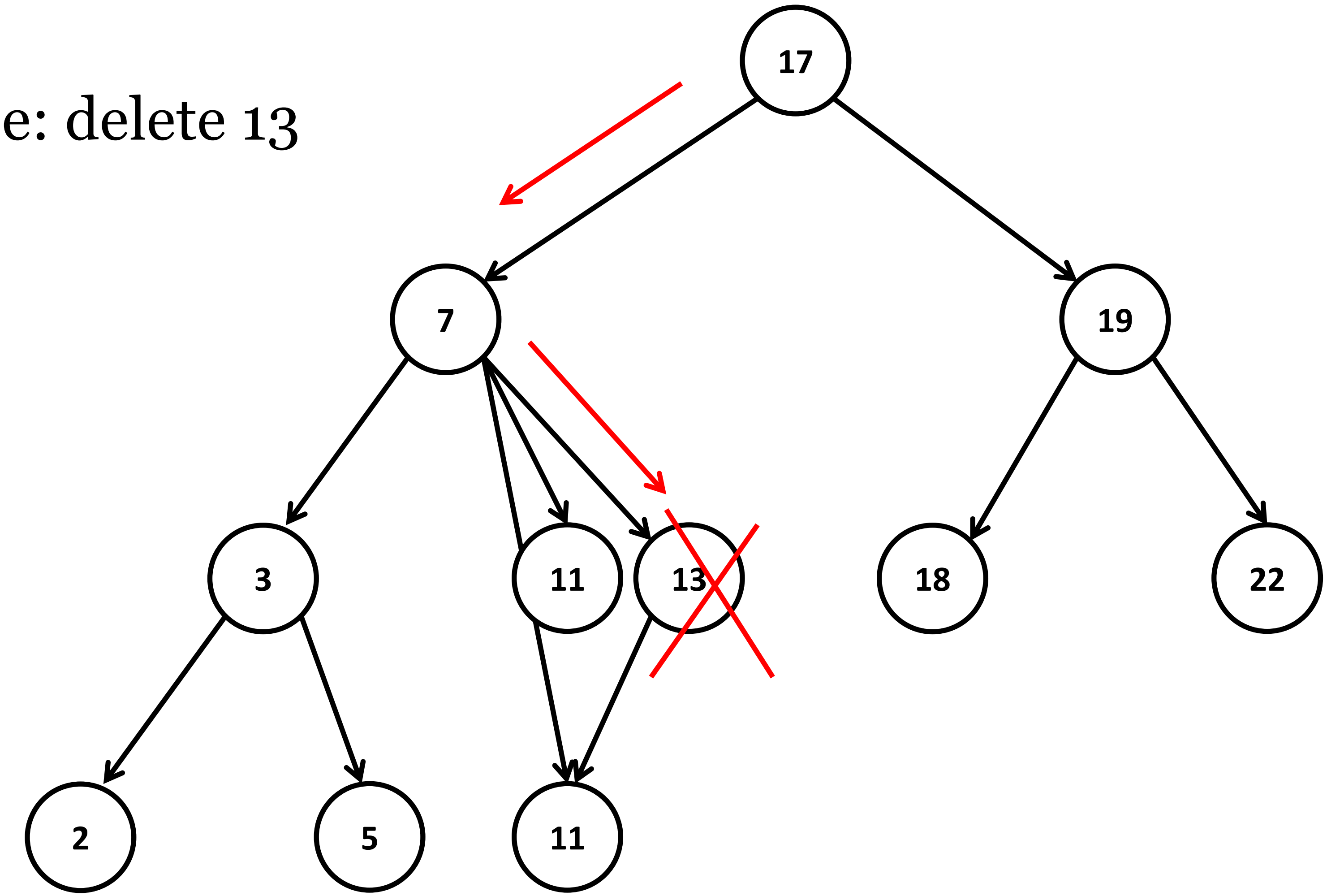
- Case 1: the node to be deleted does not have any children.
- Example: delete 11



BST - Deletion

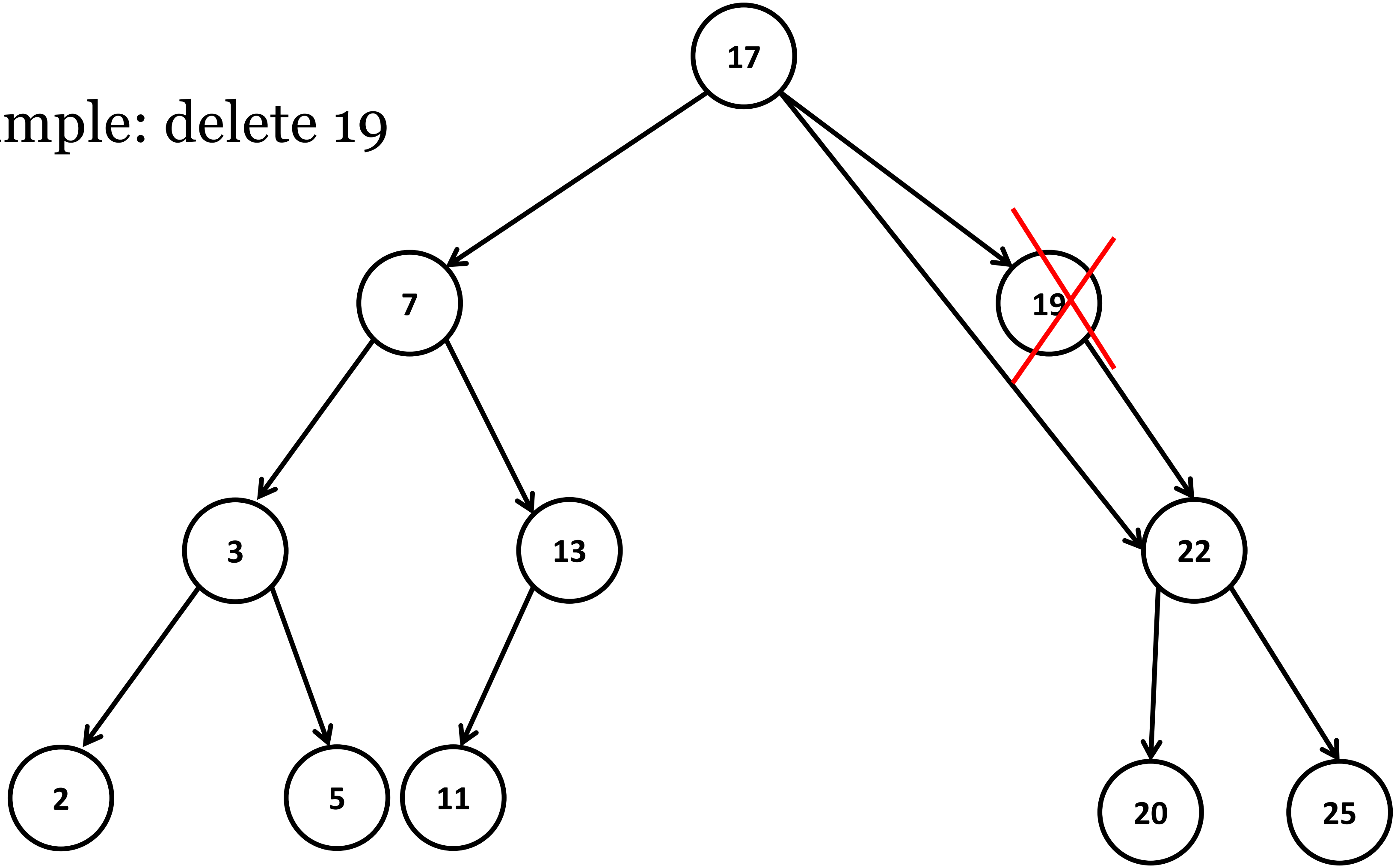
- Case 2: the node to be deleted has one child.

- Example: delete 13



BST - Deletion

- Case 2: the node to be deleted has one child.
- Example: delete 19

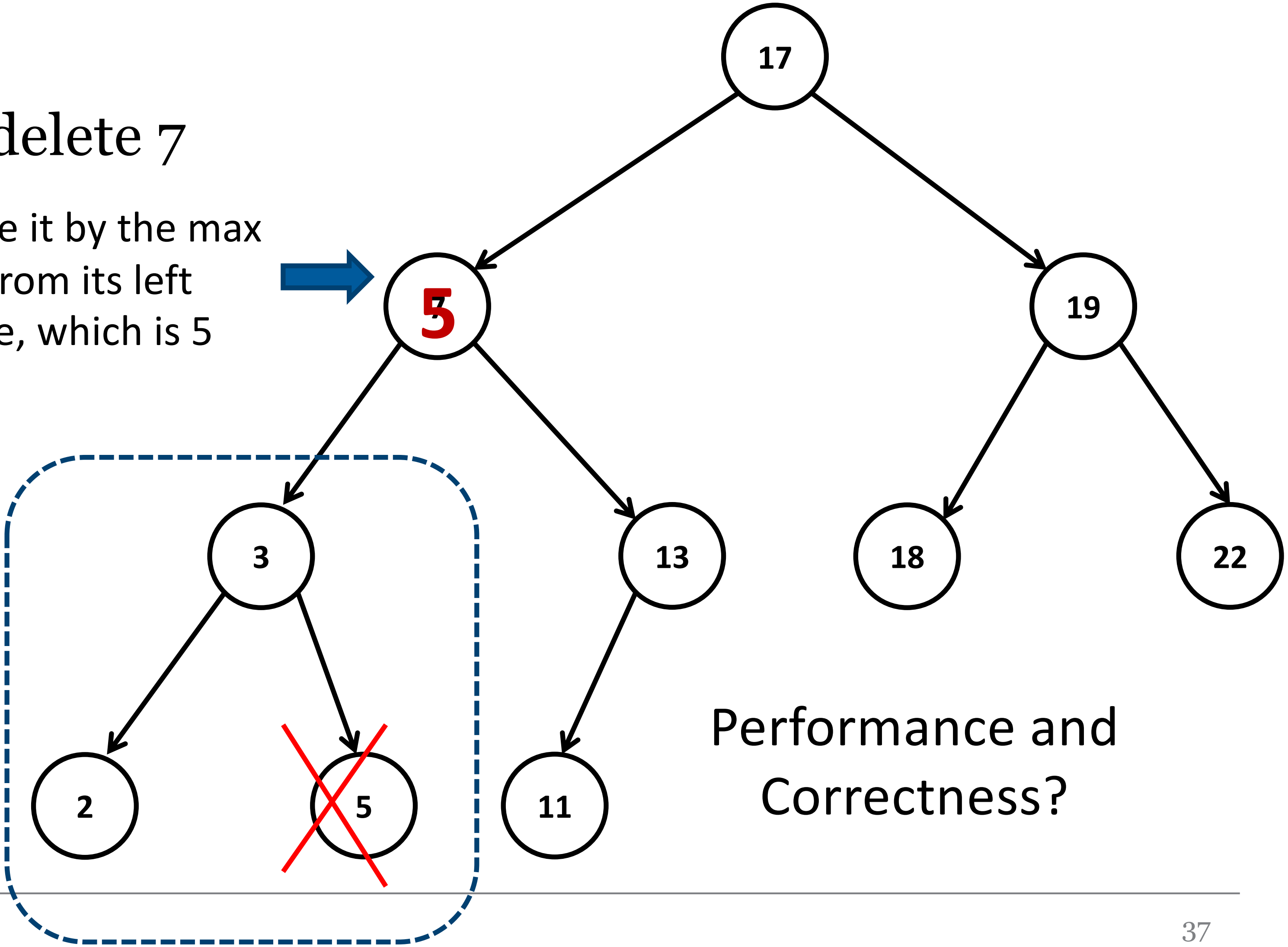


BST - Deletion

- Case 3: the node to be deleted has both children.

- Example: delete 7

Replace it by the max value from its left subtree, which is 5



Then delete 5 from its left subtree (case 1 or 2)

Performance and Correctness?

BST - Performance

- Searching, insertion, and deletion all take Θ(height) time in the worst case.
- Height is at most n-1.
- If height is k, then n is at most 1+2+...+2^k = 2^(k+1)-1.
  - n <= 2^(k+1)-1
  - k >= log(n+1)-1
  - Height is at least logarithmic in n.
- **[Fekete et al. 10]**: If the insertion order is random, then experimentally, BST's average height is less than 2.989 log(n).
- Therefore, in some sense, we can claim that for BST, searching, insertion, and deletion all take logarithmic time **on average**. (All three operations take linear time in the worst case).

Average Case for random insertion

- Assume that the items to be inserted are in random order.
- We may be lucky and the tree has small depth (does not degenerate to a list)

Question:

- What is the average time to find an element in such a tree?

Permutations of n elements

Assume that we have a set of n elements

Consider all permutations of these elements

There are n! permutations.

Example: Set {1, 2, 3}

Permutations:

(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)

Analysis

In our analysis:

- we average over the different permutations for building the binary search tree.
- all queries for the elements.

Formally, we consider “double expected value” with respect to:

- the order of elements inserted
- the element we query

Cost of a search tree

c(v): number of nodes on the path from the root to v.

Cost of a tree T:

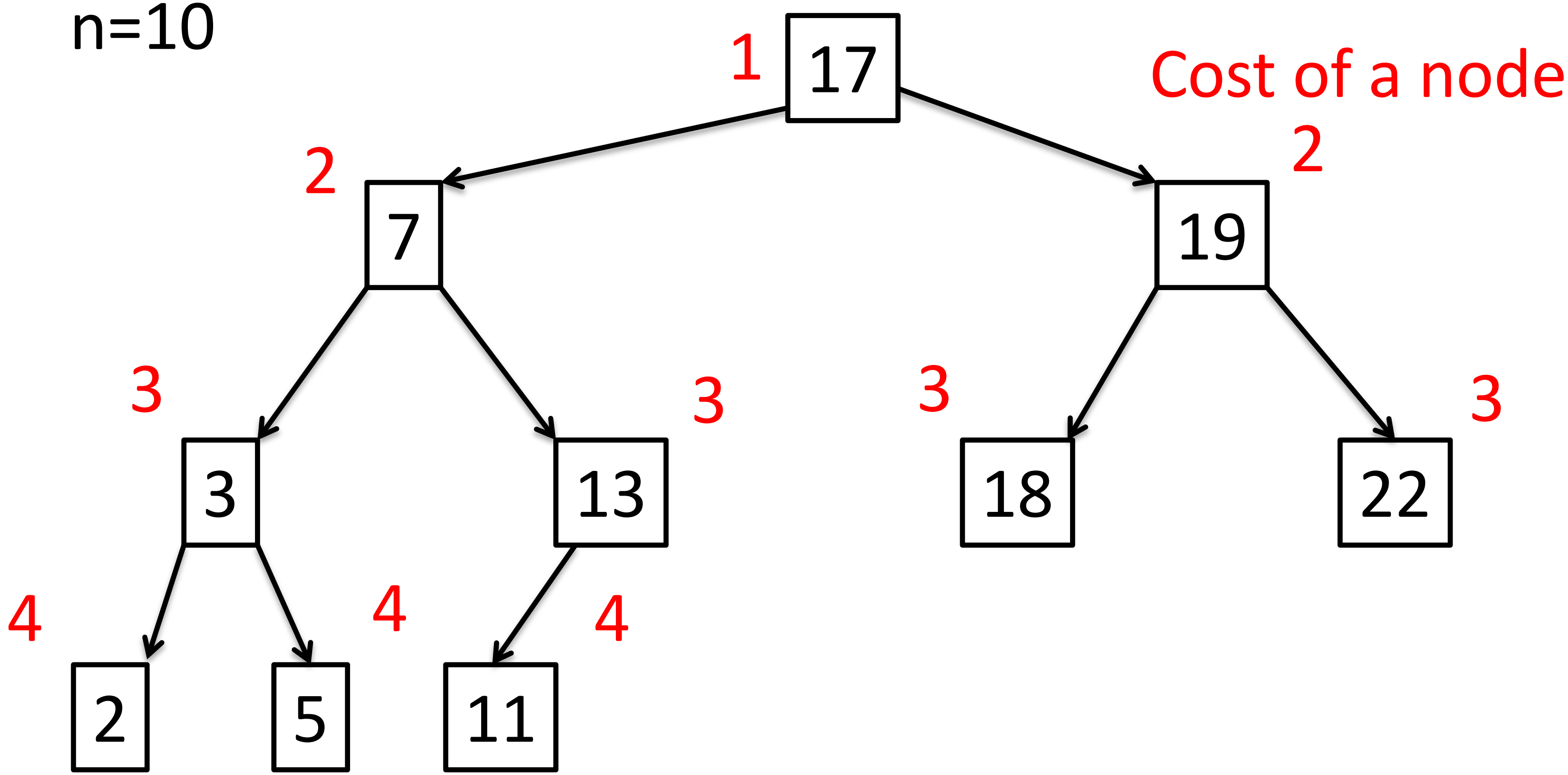
$$C(T) = \sum_{v \in T} c(v)$$

Average search cost of a tree T:

$$C(T)/n$$

Cost of a tree

n=10



Cost of the tree C(T) = 1+2+2+3+3+3+3+4+4+4=29

Average search time for T: C(T) / n = 29 / 10 = 2.9

Average costs of a tree

Let E(n) be the average cost of tree with n elements.

Recursion:

$$E(0) = 0$$

$$E(1) = 1$$

$$E(n) = n + \frac{1}{n}\sum_{i=1}^{n}(E(i-1) + E(n-i))$$

Recursive Formula

i-1 elements go into the left subtree

n-i elements go into the right subtree

$$E(n) = n + \frac{1}{n} \sum_{i=1}^{n} (E(i-1) + E(n-i))$$

Root lies on every path to a node

Each element i is with equal probability the root

Solve Recursion

- Recursive Formula seems to be complicated.
- Is it worth the effort?

Reasons for doing that:

- Result is interesting
- Math tricks can often be used
- Similar analysis gives average case results for the Quicksort algorithm.

Solving Recursion

$$E(n) = n + \frac{1}{n}\sum_{i=1}^{n}(E(i-1) + E(n-i))$$

contains E(0), E(1), …., E(n-1).

First step:

- Get a recursive formula for E(n) that only depends on E(n-1).

Consider $$n \cdot E(n) - (n-1)E(n-1)$$

This implies that E(n-2), ..., E(1) get the same factor and cancel out, i. e.

$$n \cdot E(n) = n^2 + \sum_{i=1}^{n} (E(i-1) + E(n-i))$$

$$= n^2 + 2 \cdot (E(1) + E(2) + \ldots E(n-1))$$

$$(n-1)\cdot E(n-1) = (n-1)^2 + \sum_{i=2}^{n}(E(i-1)+E(n-i))$$

$$= (n-1)^2 + 2\cdot(E(1)+E(2)+\ldots E(n-2))$$

$$n\cdot E(n) - (n-1)E(n-1)$$

$$= n^2 - (n-1)^2 + 2\cdot E(n-1)$$

$$= 2n - 1 + 2\cdot E(n-1)$$

$$n \cdot E(n) - (n+1) \cdot E(n-1) = 2n - 1$$

Divide by n(n+1)

$$\frac{1}{n+1} \cdot E(n) - \frac{1}{n} \cdot E(n-1) = \frac{2n-1}{n(n+1)}$$

Consider:

$$Z(n) = \frac{1}{n+1} \cdot E(n)$$

$$
\begin{aligned}
Z(n) &= Z(n-1) + \tfrac{2n-1}{n(n+1)} \\
&= Z(n-2) + \tfrac{2(n-1)-1}{(n-1)n} + \tfrac{2n-1}{n(n+1)} \\
&= Z(0) + \textstyle\sum_{i=1}^{n} \tfrac{2i-1}{i(i+1)}
\end{aligned}
$$

Use: $\dfrac{1}{i(i+1)} = \dfrac{1}{i} - \dfrac{1}{i+1}$

Then we get:

$$Z(n) = \quad 2\sum_{i=1}^{n}\frac{i}{i} - 2\sum_{i=1}^{n}\frac{i}{i+1}$$

$$- \quad \sum_{i=1}^{n}\frac{1}{i} + \sum_{i=1}^{n}\frac{1}{i+1}$$

$$= 2n - 2n + 2\sum_{i=1}^{n}\frac{1}{i+1} - 1 + \frac{1}{n+1}$$

$$= 2\sum_{i=1}^{n}\frac{1}{i} - 2 + \frac{2}{n+1} - 1 + \frac{1}{n+1}$$

$$= 2\cdot H(n) - 3 + \frac{3}{n+1}$$

Harmonic sum $H(n) = \sum_{i=1}^{n}\frac{1}{i}$

Remember: $$Z(n) = \frac{1}{n+1} \cdot E(n)$$

$$E(n) = (n+1) \cdot Z(n)$$

$$= 2(n+1) \cdot H(n) - 3(n+1) + 3$$

Average Cost for Find

Average cost for find after random insertion:

$$E(n)/n = 2 \cdot \frac{n+1}{n} \cdot H(n) - 3 \cdot \frac{n+1}{n} + \frac{3}{n}$$

Using: $\ln(n+1) \leq H(n) \leq \ln n + 1$

we get

$$\begin{aligned} E(n)/n &= 2 \cdot \ln n - O(1) = (2 \ln 2) \cdot \log n - O(1) \\ &\approx 1.386 \cdot \log n \end{aligned}$$

Theorem

Theorem: The insertion of n randomly chosen elements leads to a Binary Search Tree whose expected time for a successful find operation is

$$(2 \ln 2) \cdot \log n - O(1) \approx 1.386 \cdot \log n$$

Runtimes for Binary Search Tree

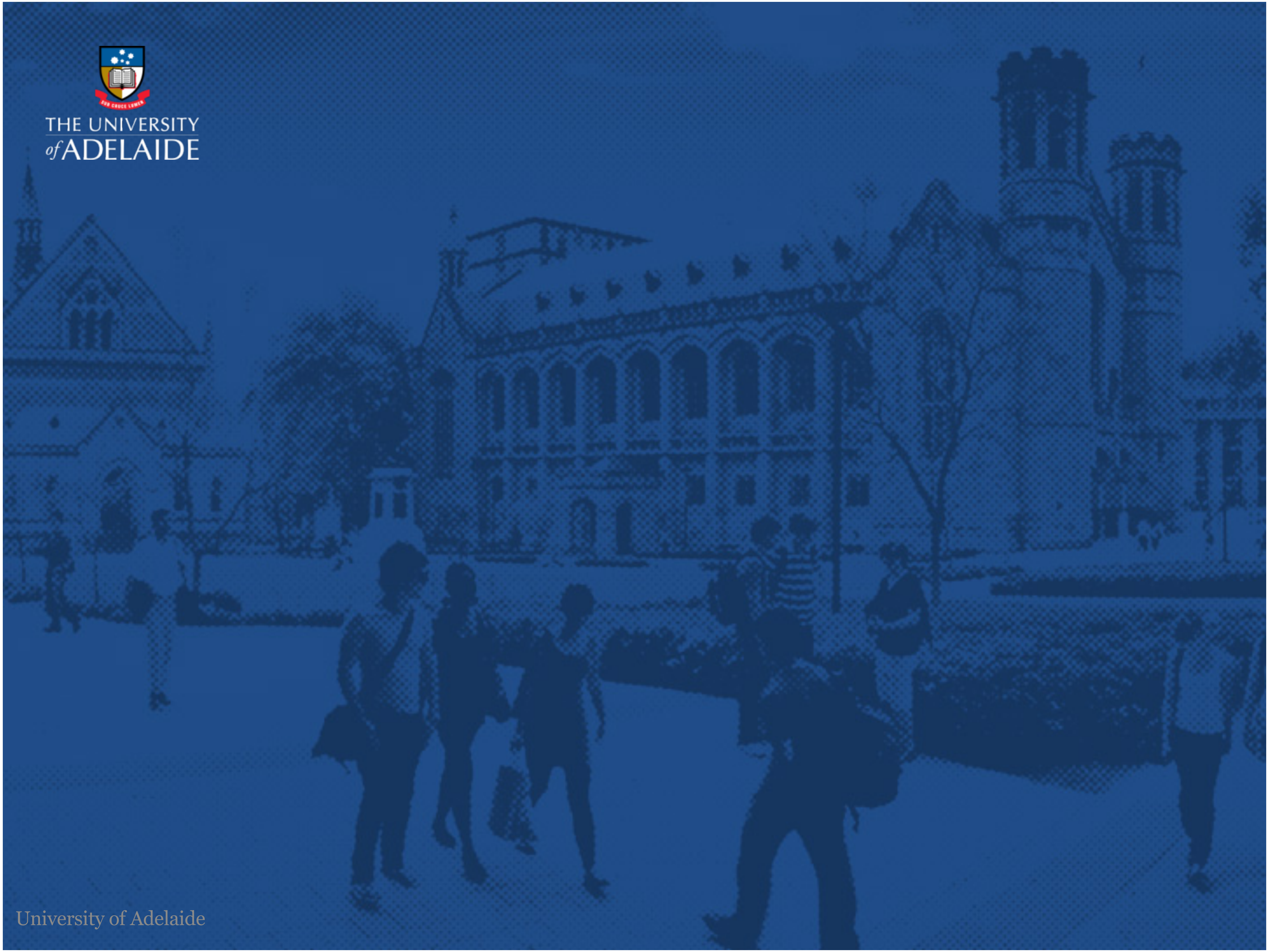
Find, insert, remove:

Worst case: $\Theta(n)$

Best case: $\Theta(\log n)$

Average case: $\Theta(\log n)$

Aim: Time O(log n) in the worst case


THE UNIVERSITY
of ADELAIDE
University of Adelaide