(07) Trees Part 2: Binary Search Trees

Video (16mins): https://youtu.be/F0pqFMHr_JQ

Searching a Binary Tree

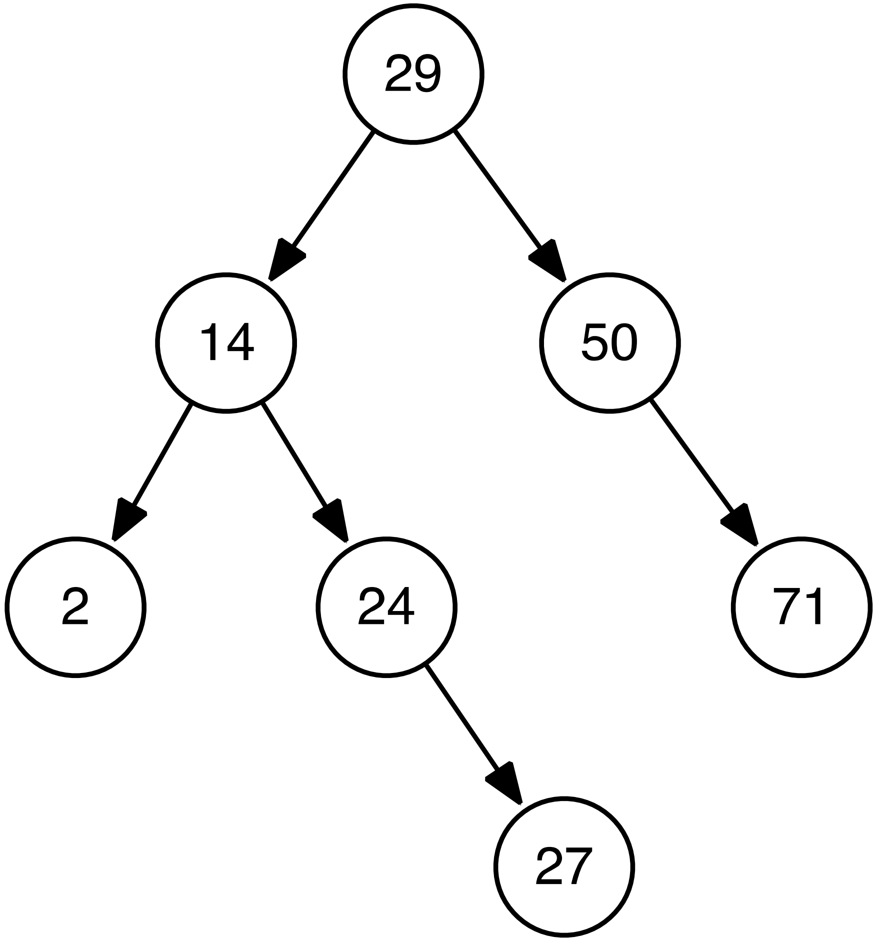
- Fundamental operation on a binary tree
  - Like linear search, we have to iterate through each node in the tree
  - Using one of the traversal functions, searching for a node takes O(`n`) for a tree with `n` nodes
- Remember linear search vs. binary search?
- Can we "sort" a binary tree so we can do "binary search" on it?

Binary Search Tree

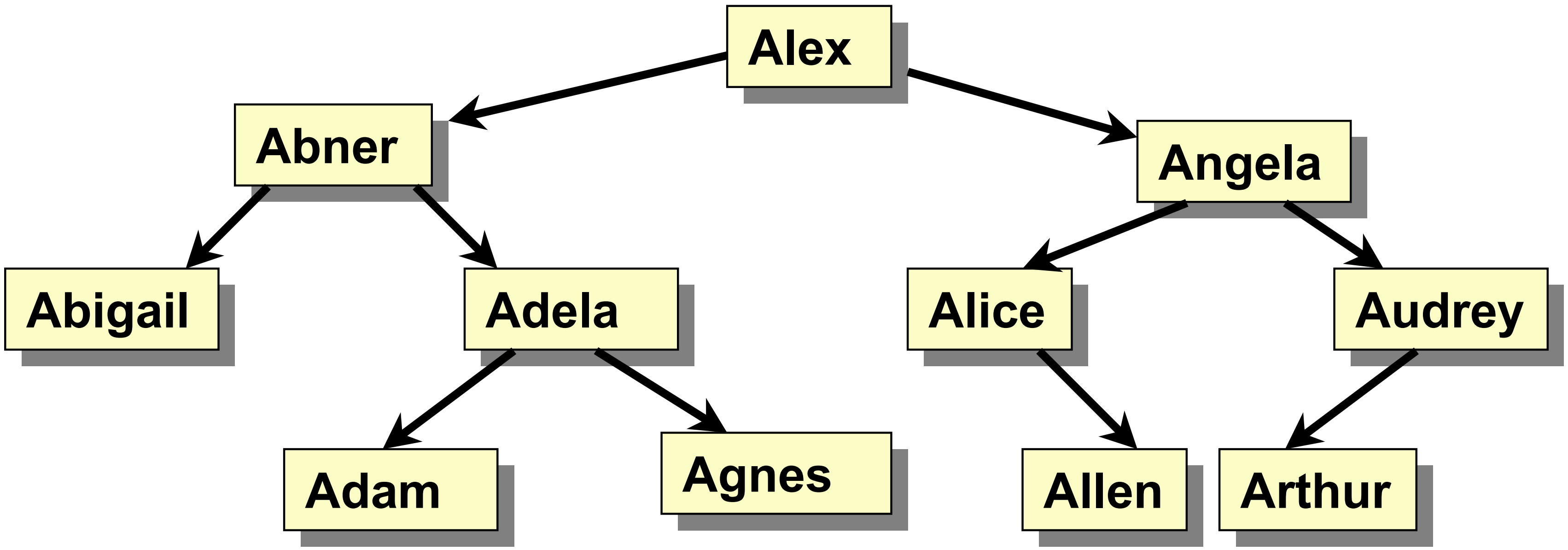
- A binary search tree is a like a binary tree, except that one of the node's attributes is specified to be the search key.
  - For simplicity, we assume the node value is the search key.

- Recursive definition of a binary search tree:
  - root node `r` has a value or key larger than all the nodes in its left subtree
  - root node `r` has a value smaller than all the nodes in its right subtree
  - its left and right subtrees are also binary search trees

Example: Binary Search Tree



Example: Binary Search Tree

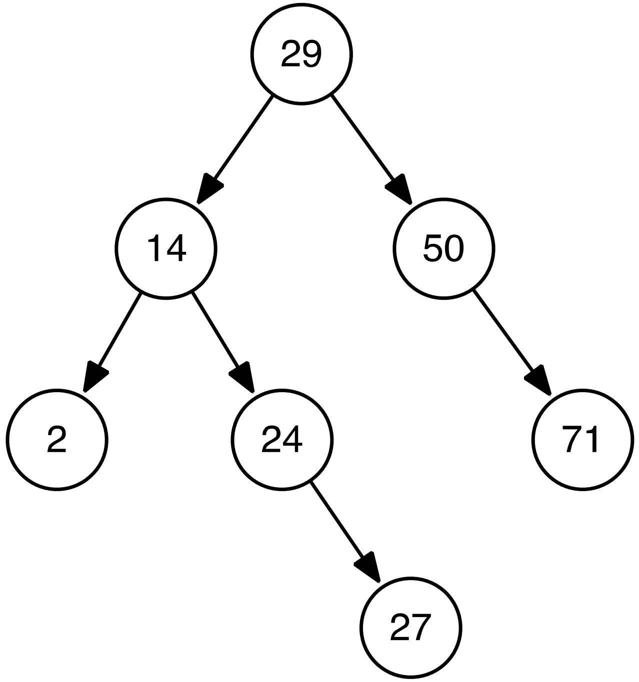


`BinarySearchTree` Object: Operations

- Similar to `BinaryTree` object, with additional operations
- `BinarySearchTree()`
- `search(key)`
  - searches for a node with `value == key`, and returns the node if found
- `insert(key)`
  - inserts a new node with value set to the given `key`
- `remove(key)`
  - removes an existing node with `value == key`

Searching a BST

- Start from the root node
  - if the root node has the `value == key`, return the root node
  - if the root node has `value > key`, recursively search the left subtree
  - if the root node has `value < key`, recursively search the right subtree
  - else, return `None`



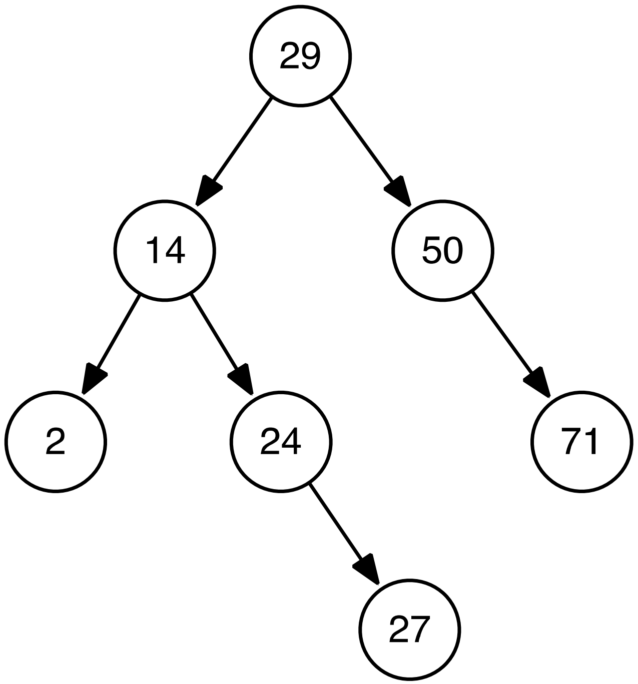
Helper Function `searchbst`

```
def searchbst(root, key):
  if root.value == key:   # found!
    return root
  elif root.left != None and root.value > key:
    return searchbst(root.left, key)
  elif root.right != None and root.value < key:
    return searchbst(root.right, key)
  else:                   # not found
    return None
```

Complexity?

Inserting a Node into a BST

- Insert the new node where we would have expected to find it
  - the exact position where searching the current tree would fail



Animation: http://btv.melezinek.cz/binary-search-tree.html

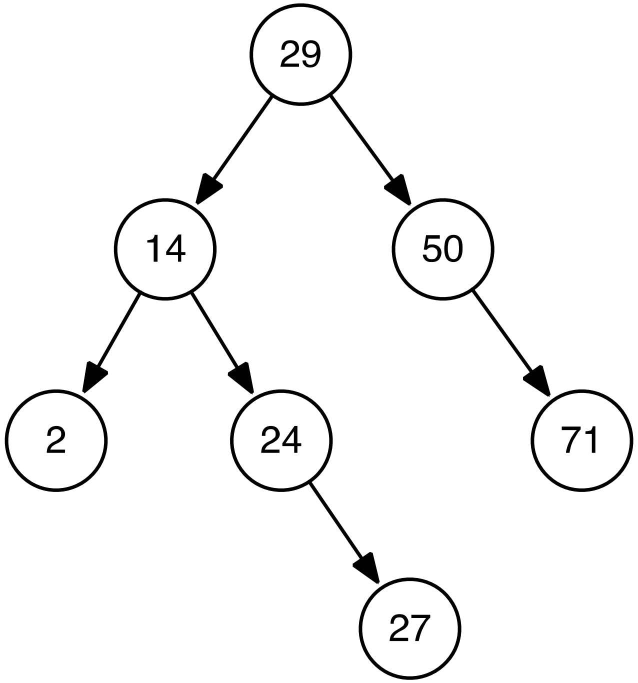
```python
def insertbst(root, key):
  if root.value == key:
    return root
  elif root.value > key:
    if root.left != None:
      return insertbst(root.left, key)
    else:
      node = Node(key)
      root.setLeft(node)
      return node
  else:
    if root.right != None:
      return insertbst(root.right, key)
    else:
      node = Node(key)
      root.setRight(node)
      return node
```

searching has failed; perform insertion here

searching has failed; perform insertion here

Removing a Node from BST

- ✦ Take care to keep the resulting BST in proper order
- ✦ Traverse to find the node
- ✦ Need to consider three scenarios:
  - a) the node to be removed is a leaf node
  - b) the node to be removed has one child
  - c) the node to be removed has two children



Removing a Node from BST

- How to delete node (n):
  a) node to be removed (n) is a leaf node
     - Set n's parent's pointer to `None`
  b) node to be removed has 1 child
     - Let n's parent adopt n's child
  c) node to be removed has 2 children
     - need to choose a replacement node for n:
     - either choose the left-most node of the right-subtree of n, OR
     - choose the right-most node of the left-subtree of n

See:
http://en.wikibooks.org/wiki/Data_Structures/All_Chapters#Deleting_an_item_from_a_binary_search_tree

- (c) node to be removed has to 2 children (example 1)

To delete node 30

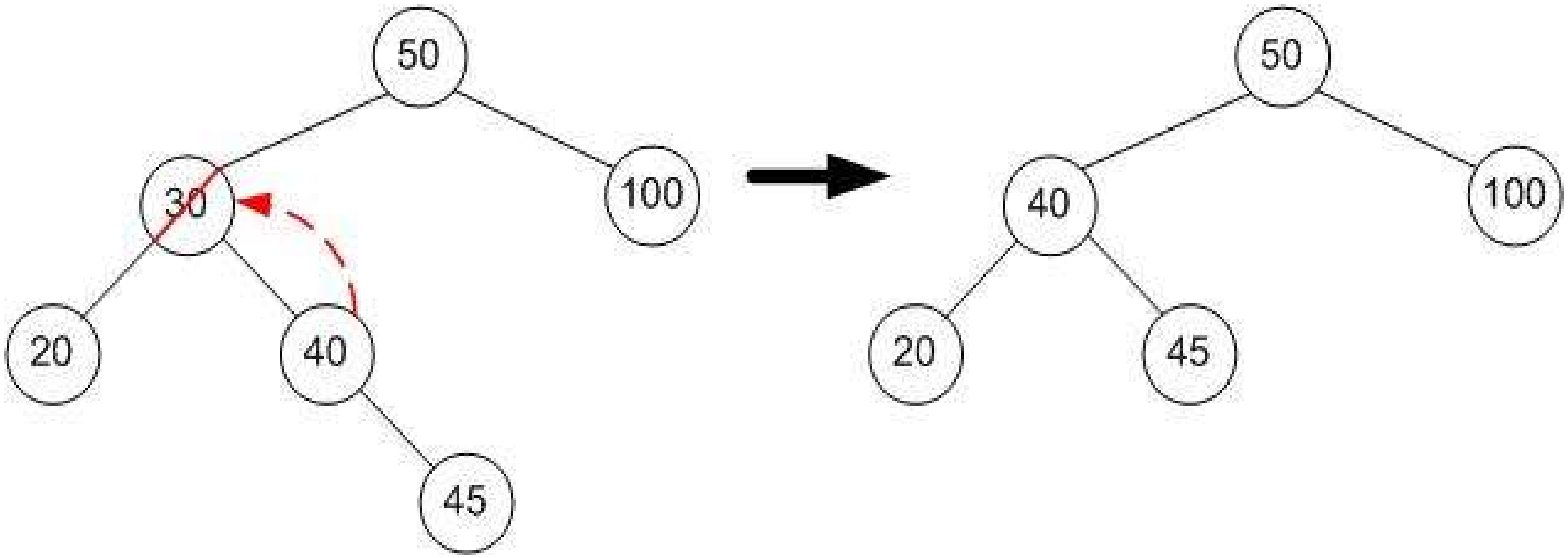


Diagram by Linuxdude at English Wikibooks - Transferred from en.wikibooks to Commons., Public Domain, https://commons.wikimedia.org/w/index.php?curid=61794591

- (c) node to be removed has to 2 children (example 2)

To delete node 30

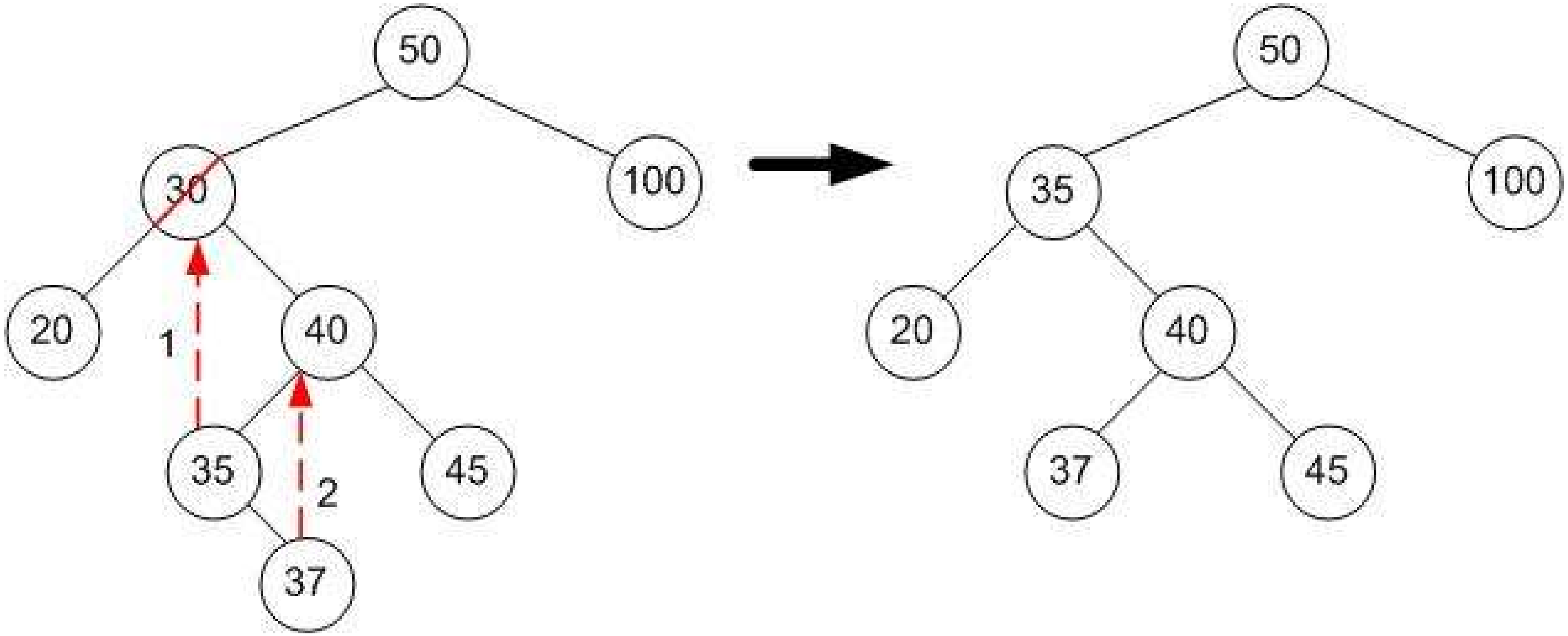
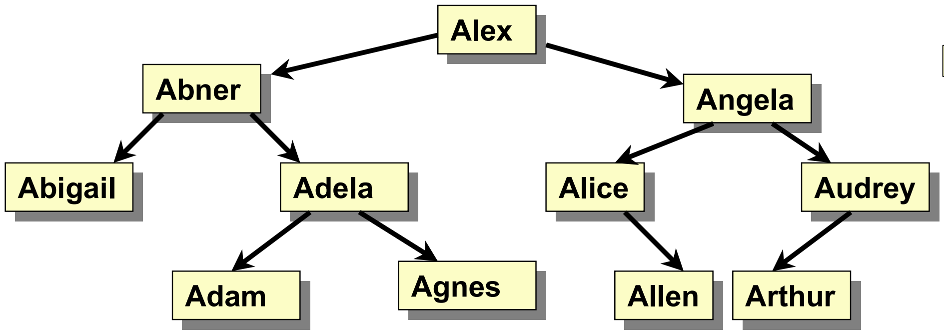
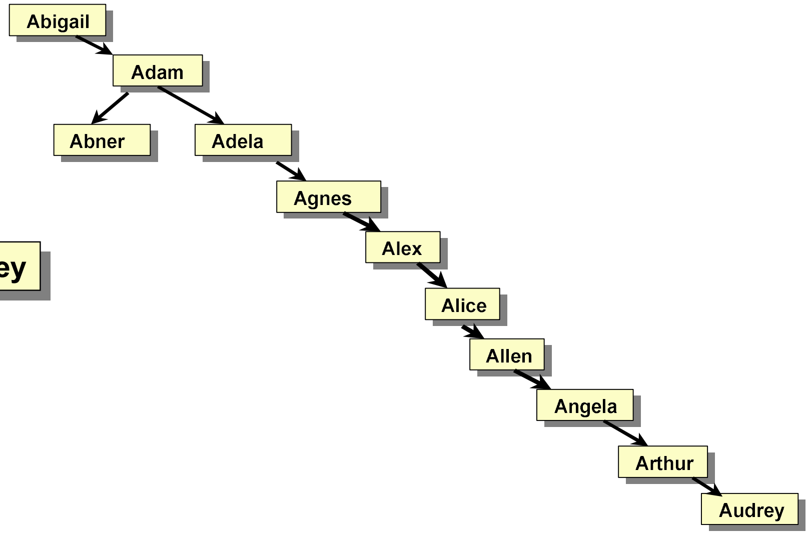


Diagram by Linuxdude at English Wikibooks - Transferred from en.wikibooks to Commons., Public Domain, https://commons.wikimedia.org/w/index.php?curid=61794591

Efficiency of BST Operations

- The maximum number of comparisons for a search, insertion, or deletion operation is the height of the tree
- What is the maximum and minimum heights of a binary search tree of n nodes?





- The shape of a binary search tree determines the efficiency of its operations

In-Class Exercises

(a) Build a BST when inserting numbers in the following sequences respectively:

- BST1: 55, 4, 23, 21, 12, 7, 66, 1, 91, 44
- BST2: 23, 7, 66, 55, 12, 91, 21, 1, 4, 44
- BST3: 1, 91, 7, 55, 4, 44, 12, 23, 21, 66

(b) How many comparisons are needed to search for the number 1 in each BST above?

(c) Provide an algorithm that takes a binary tree as input, and outputs **True** if the input is a BST and **False** otherwise

Summary

- Trees are data structures that capture hierarchical relationships
- Binary tree:
  - every parent can have at most two children
- Traversal: walking through every node in a tree
  - preorder: visiting each parent *before* visiting its left and right children
  - inorder: visiting each parent in *between* visiting left and right children
  - postorder: visiting each parent *after* visiting left and right children
- Binary search tree:
  - more efficient search by keeping the nodes ordered
  - searching only requires visiting one node at every level
  - insertion and removal have to take care to preserve the order

Road Map

Algorithm Design and Analysis

(Weeks 1 - 5)

Fundamental Data Structures

- Week 6: Linear data structures (stack, queue)
- Week 7: Hierarchical data structure (binary tree)
- Next week → Week 9: Networked data structure (graph)

Computational Intractability and Heuristic Reasoning

(Weeks 10 - 13)