TREES

CS210 – Data Structures and Algorithms

Dr. Basit Qureshi

جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

https://www.drbasit.org/

CS210: THE JOURNEY SO FAR

	Runtime		
Data Structure / Algorithm	Bestcase	Average Case	Worst Case
Singly Linked Lists	O(n)	O(n)	O(n)
Doubly Linked Lists	O(n)	O(n)	O(n)
Circular Linked Lists	O(n)	O(n)	O(n)
Stacks*	O(1)	O(1)	O(1)
Queues*	O(1)	O(1)	O(1)
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2/2)$	$O(n^2/2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2/2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Tim Sort	O(n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(1.39 n log n)	$O(n^2/2)$

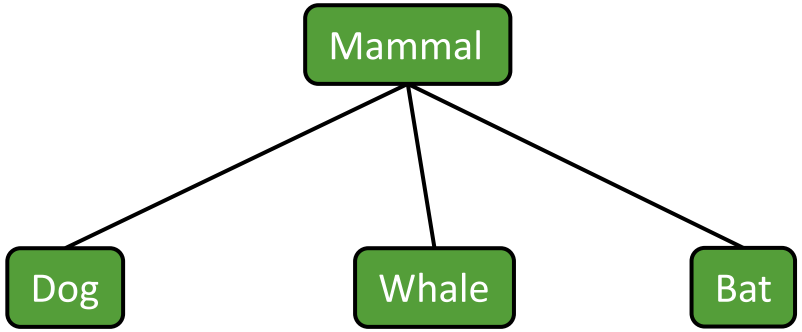
* Limited operations

TREES

- Trees: Concepts
- Tree API
- Caveats in Making Trees
- Binary Trees

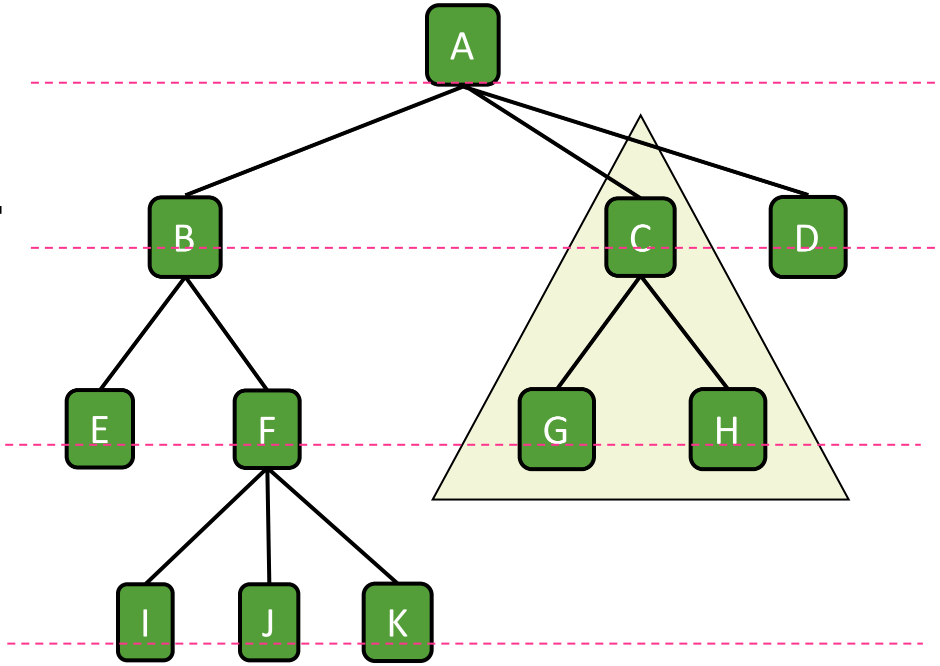
TREES

- A tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a *parent-child* relation
- Examples:
  - Organization Chart
  - File Structures
  - Programming Environments
  - Expression Trees



TREE TERMINOLOGY

- Root
- Internal node: node with at least one child
- Leaf: node without children
- Ancestors: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors of a node
- Height of a tree: maximum depth of any node
- Descendant: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants

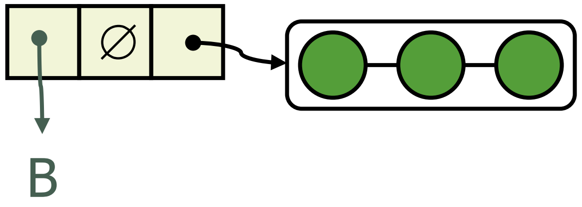


Trivia:
What is the Height of this tree?
What is the Depth at F?
What are the ancestors of G?
What are the decendants of B?

TREE API

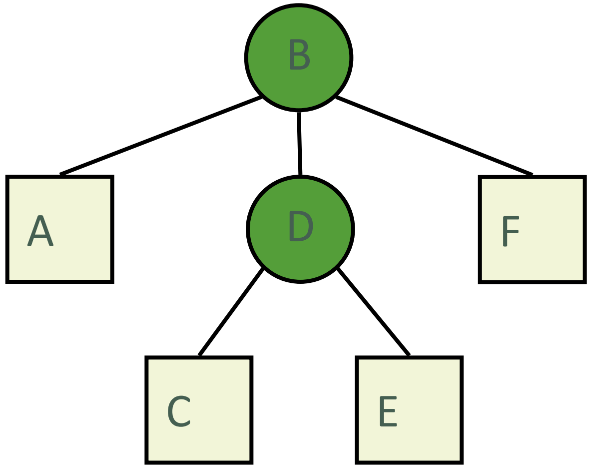
Tree
Node Root; int size;
void **insert**(int x); Node **remove**(int x); Node **search**(int x); boolean **isEmpty**(); String **toString()**; Node **getRoot()**; Node **getParent(Node)**; List **getChildren(Node)**; Node **getNumChildren()**; boolean **isLeaf(Node)**; boolean **isInternal()**;

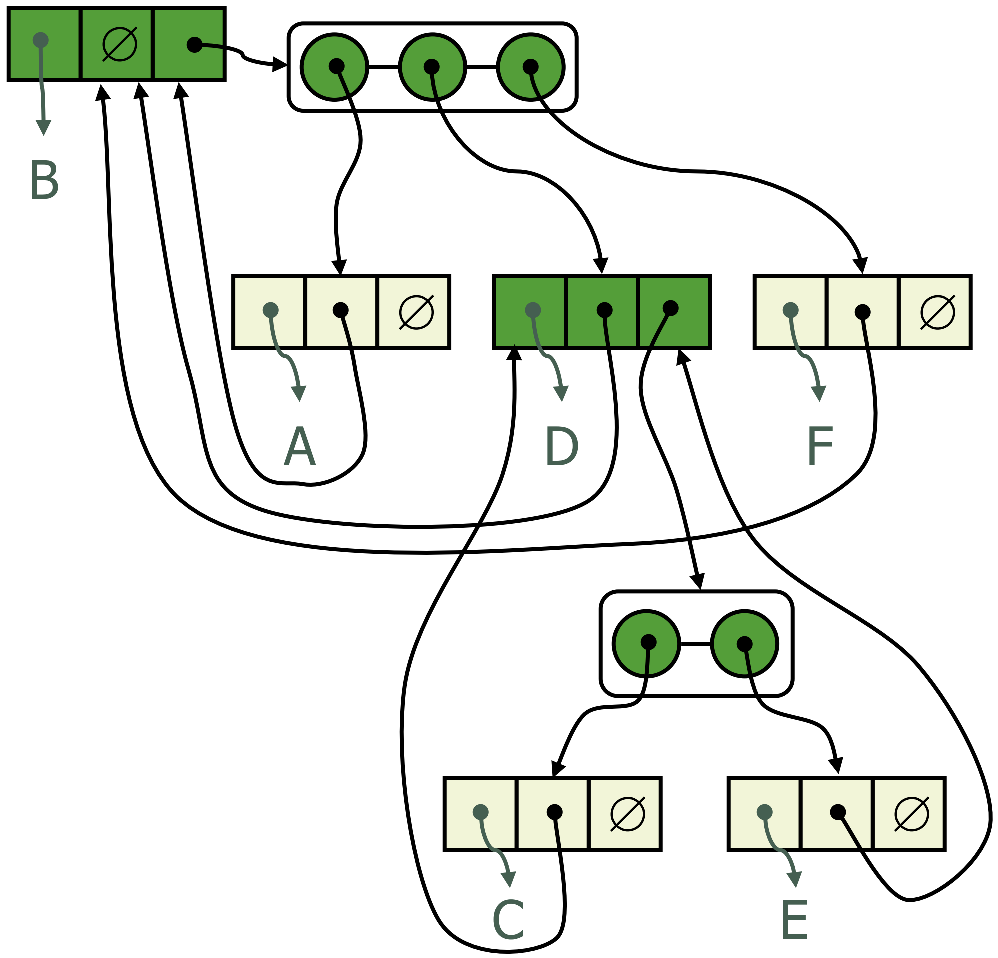
Node
int val; Node parent; List Children;
Node(); **Node**(,);



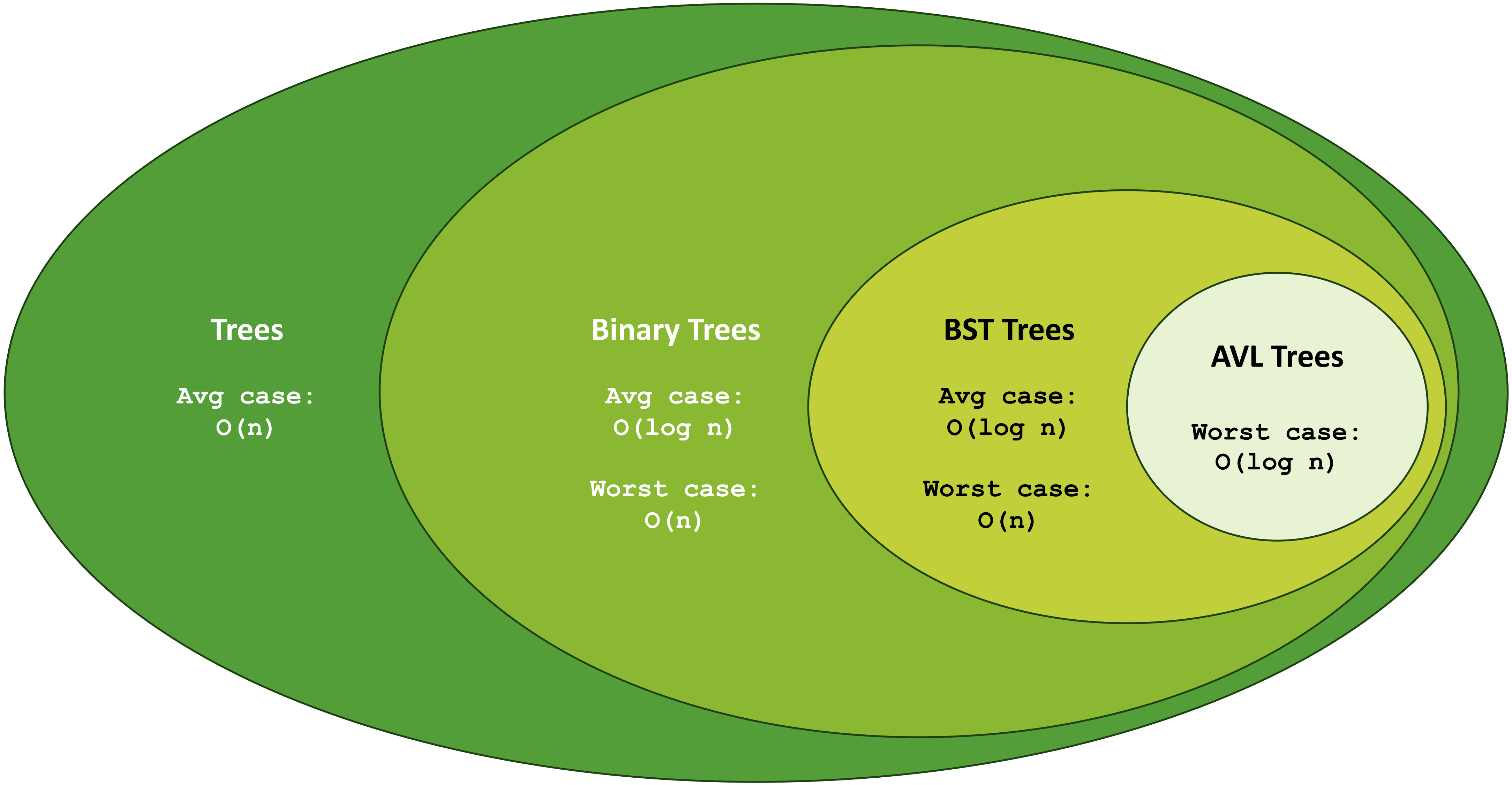
Additional methods can be defined as necessary

IMPLEMENTING A TREE



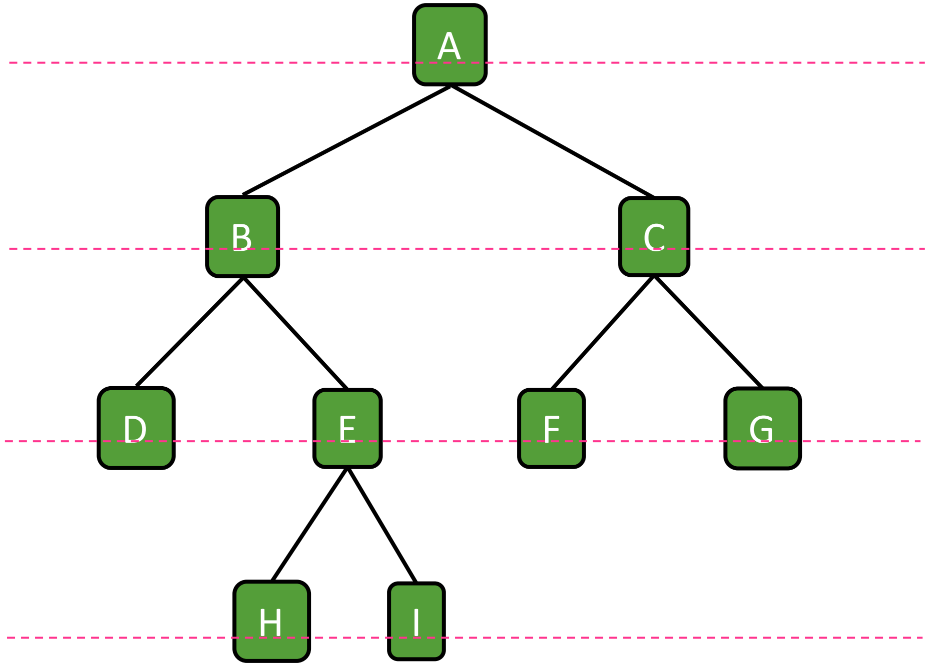


Complicated!
Is the runtime Linear or better?


Trees
Avg case:
O(n)
Binary Trees
Avg case:
O(log n)
Worst case:
O(n)
BST Trees
Avg case:
O(log n)
Worst case:
O(n)
AVL Trees
Worst case:
O(log n)

BINARY TREES

- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree



BINARY TREES

- Notation

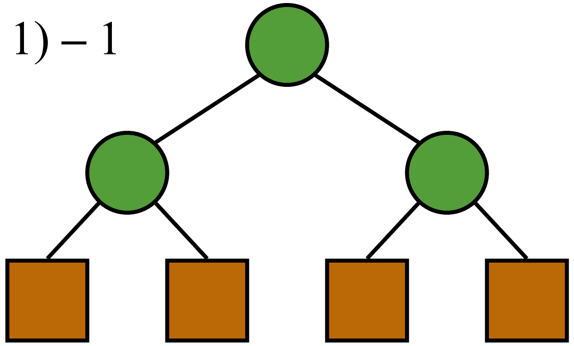
  n number of nodes

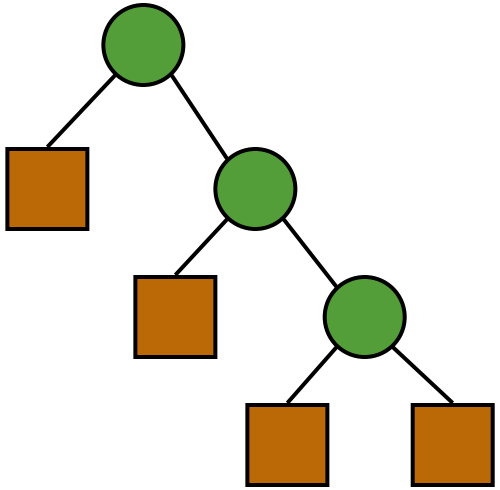
  e number of external nodes

  i number of internal nodes

  h height

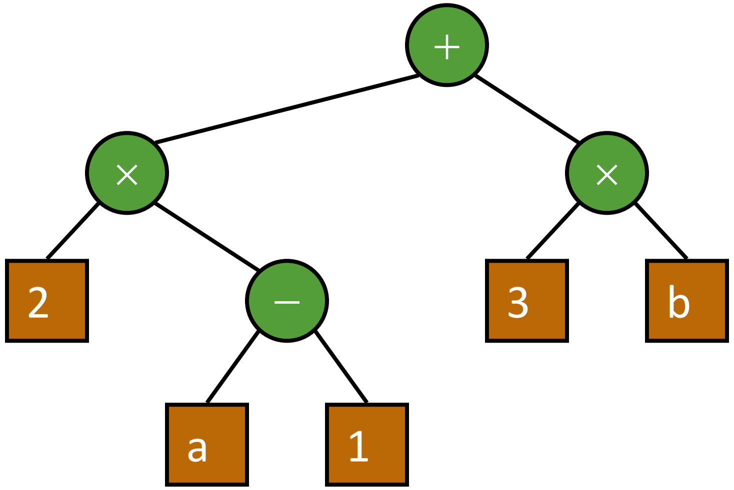
- Properties:
  - $n = 2^h - 1$
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$





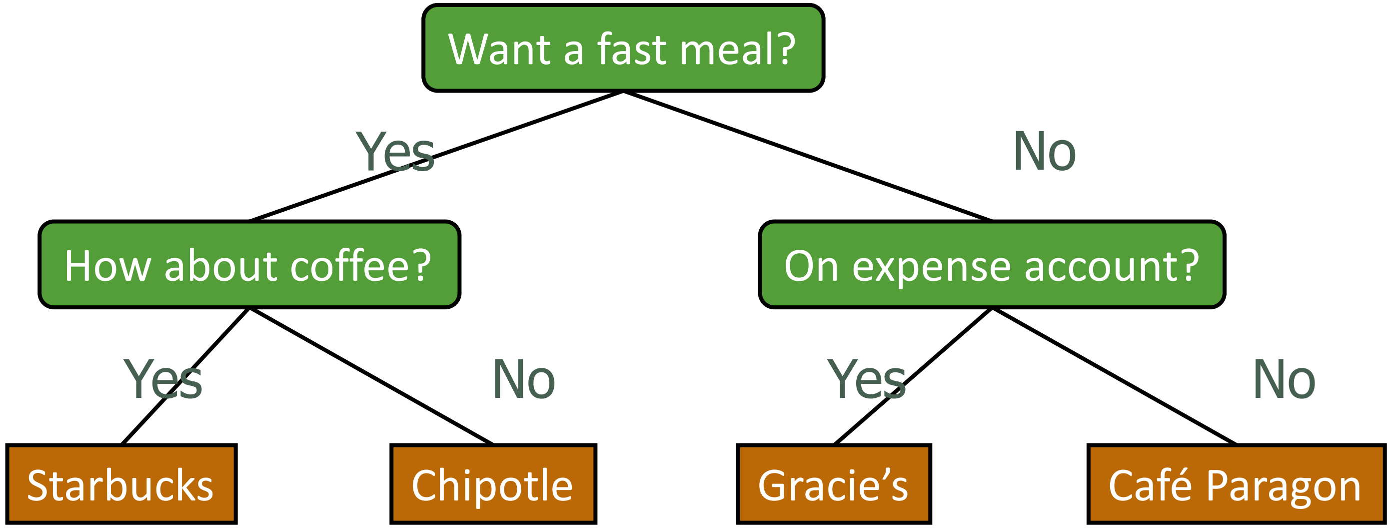
BINARY TREES

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Examples: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



BINARY TREES

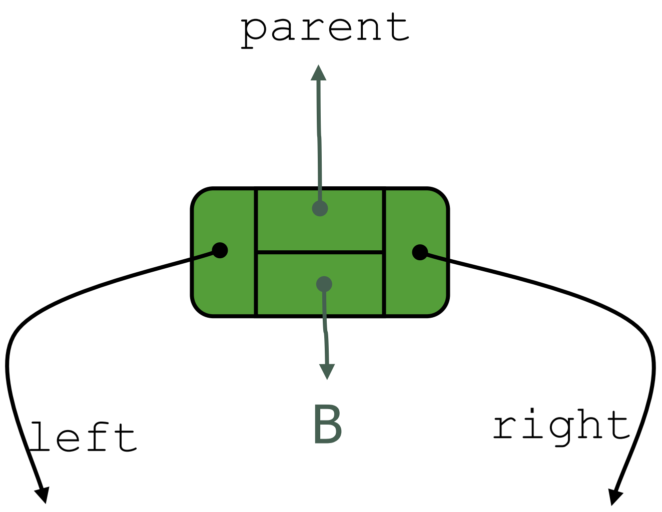
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



BINARY TREE API

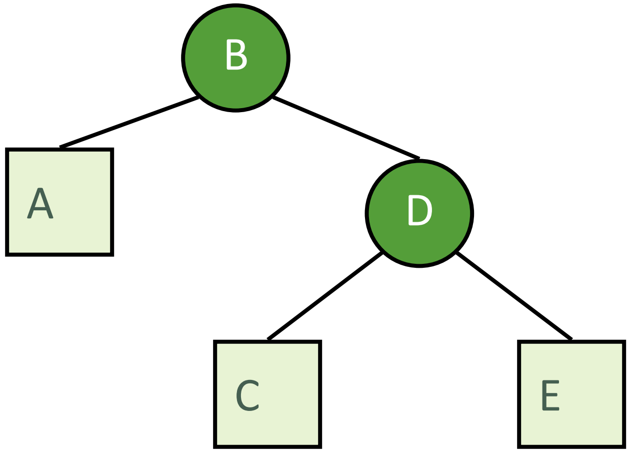
```
BinaryTree
-----------------------------
Node Root;
int size;
-----------------------------
void insert(int x);
Node remove(int x);
Node search(int x);
boolean isEmpty();
String toString();
Node getRoot();
Node getParent(Node);
List getLChild(Node);
List getRChild(Node);
boolean isLeaf(Node);
boolean isInternal();
```

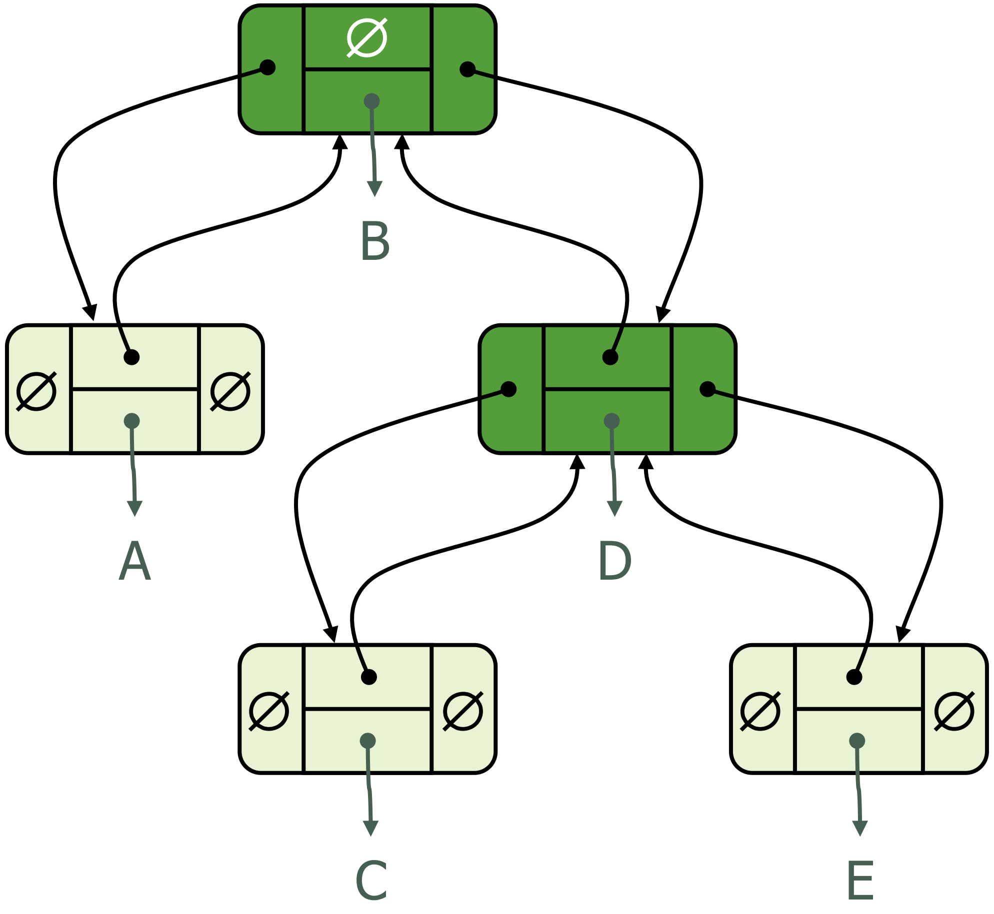
```
Node
-----------------------------
int val;
Node parent;
Node left;
Node right;
-----------------------------
Node();
Node(,);
```



Additional methods can be defined as necessary

BINARY TREE WITH LINKED STRUCTURES



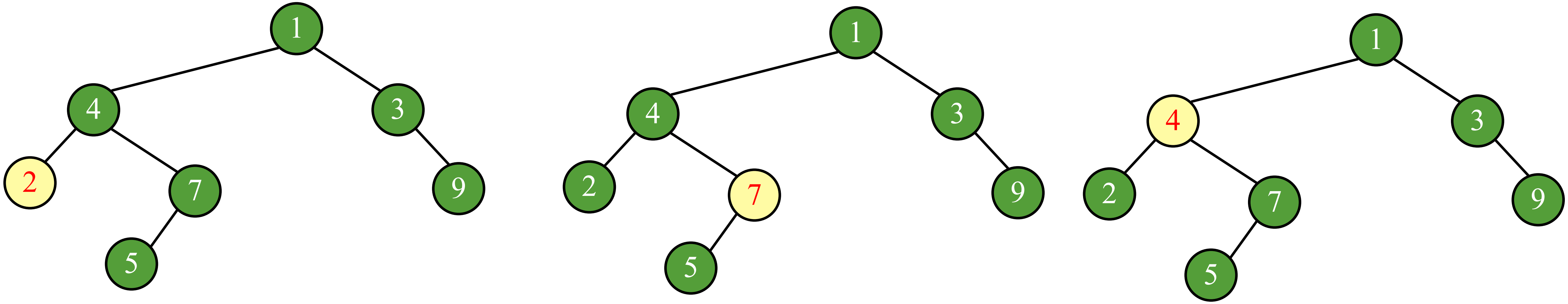


Is the runtime Linear or better?

- Insert
- Search
- Delete

BINARY TREE WITH LINKED STRUCTURES

Deletion is a problem!

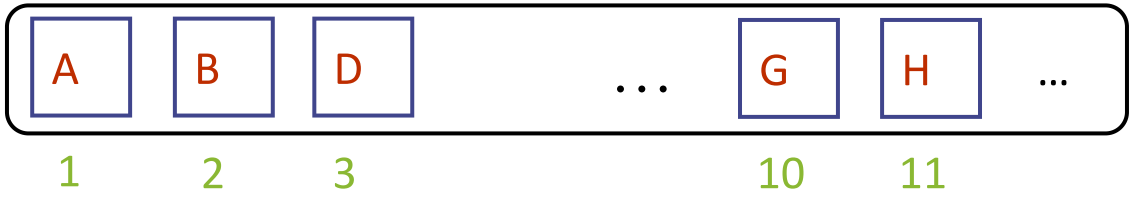


Deletion problem!

- Delete node that is a leaf
- Delete node that has one child
- Delete node that has two children

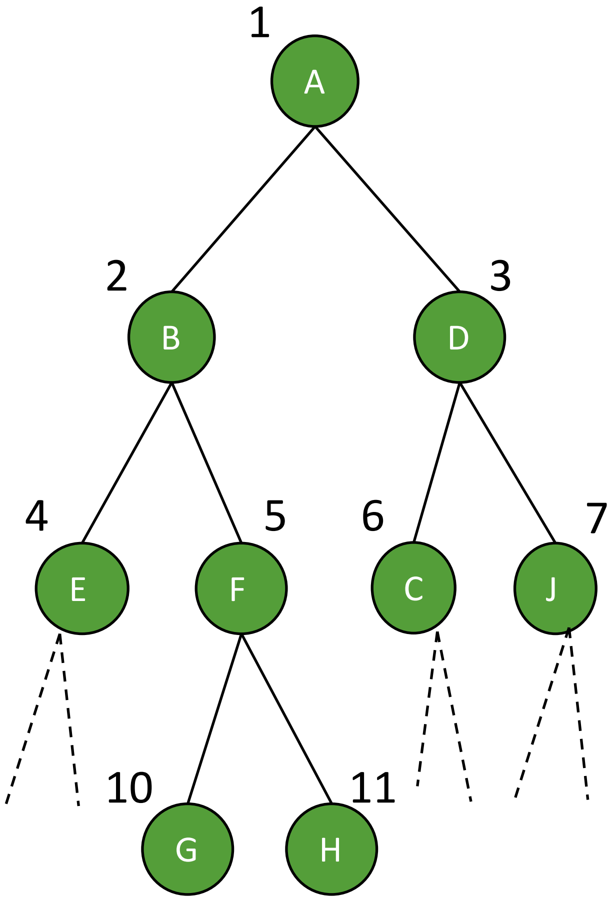
BINARY TREE WITH ARRAYS

- Fixed size, but faster?!!



Node v is stored at A[position(v)]

- **position(root) = 1**
- **Parent(v) = position(v) / 2**
- **LeftChild(v) = position(v) * 2**
- **RightChild(v) = position(v) * 2 + 1**

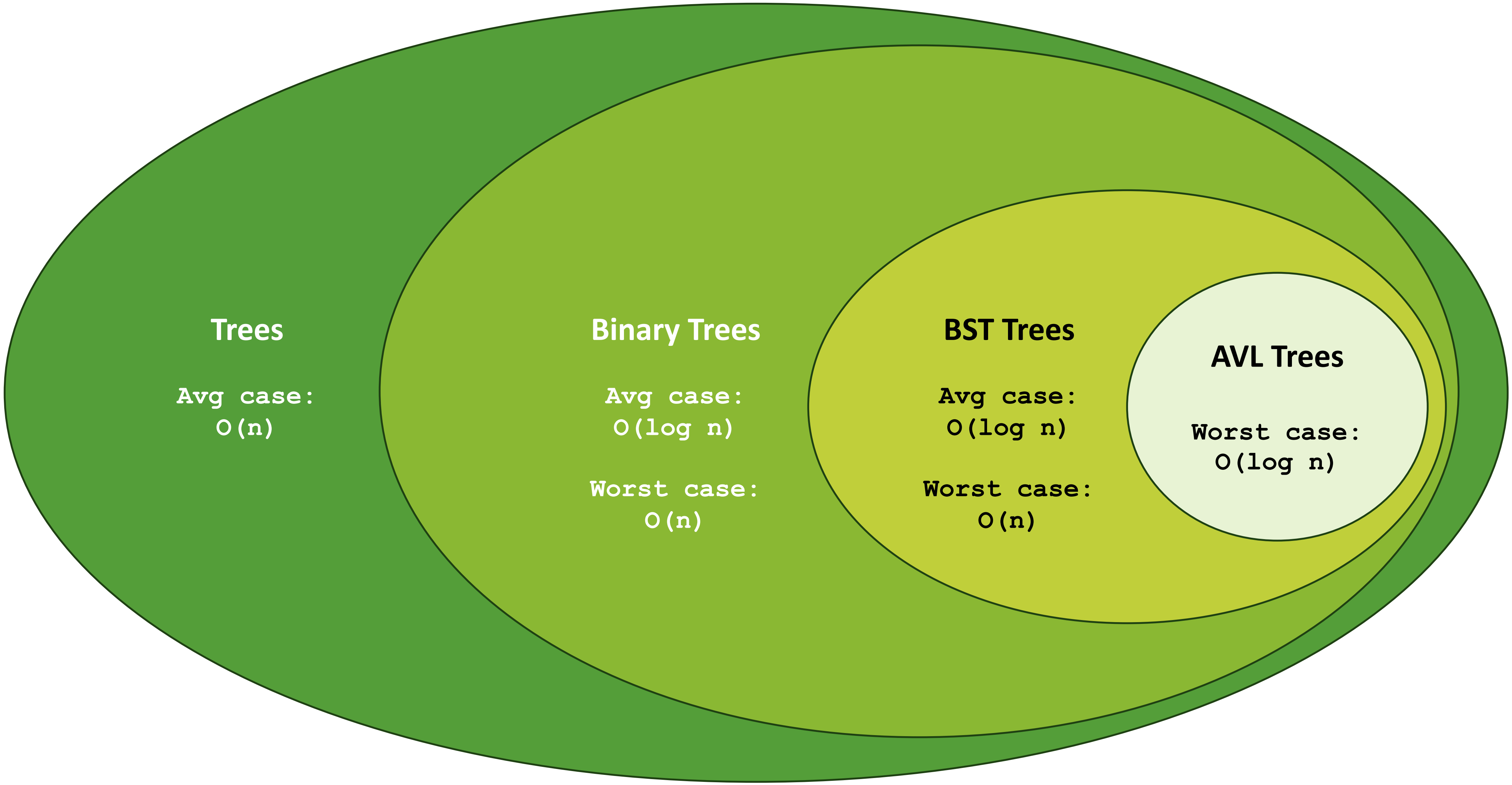


Does it improve the runtime?

- Insert
- Search
- Delete

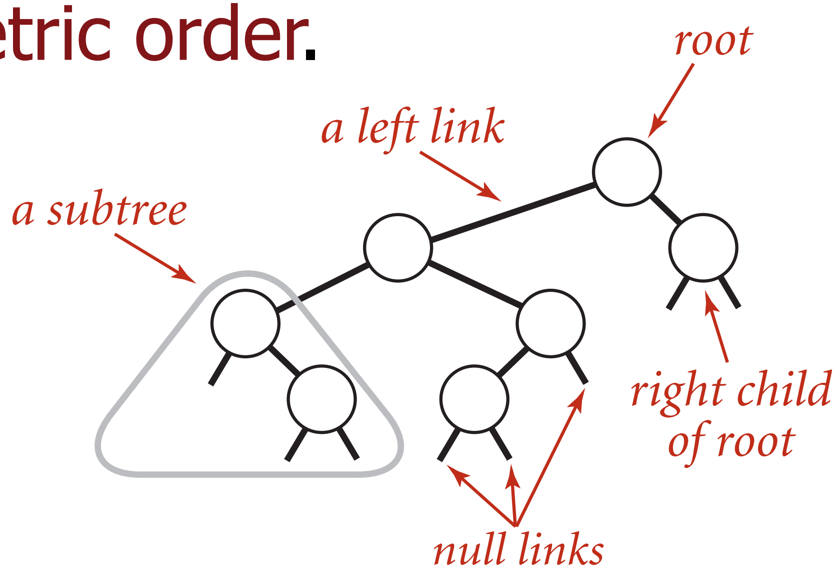
What about the deletion problem!

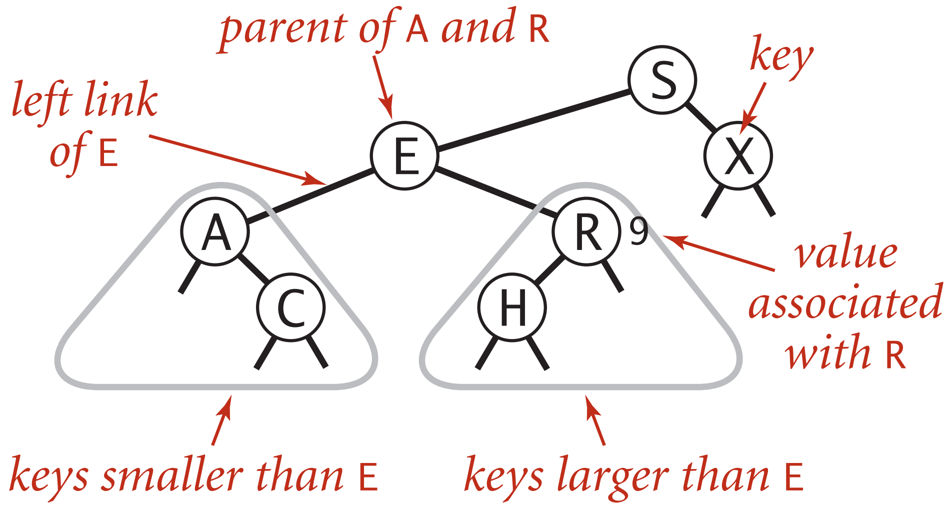
- Delete node that is a leaf
- Delete node that has one child
- Delete node that has two children

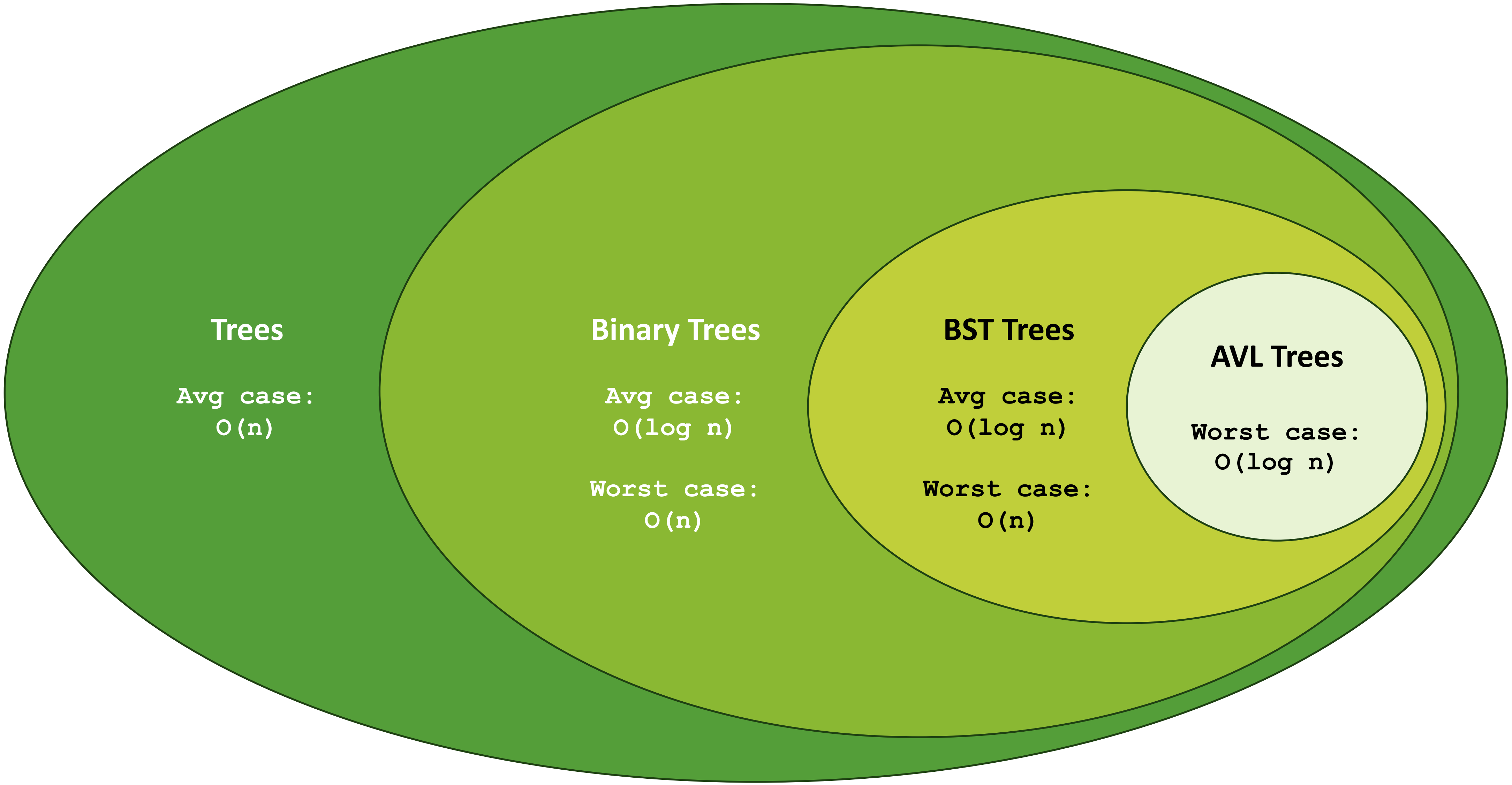

Trees
Avg case:
O(n)
Binary Trees
Avg case:
O(log n)
Worst case:
O(n)
BST Trees
Avg case:
O(log n)
Worst case:
O(n)
AVL Trees
Worst case:
O(log n)

BINARY SEARCH TREES

- **Definition.** A BST is a binary tree in symmetric order.
- A binary tree is either:
  - Empty.
  - Two disjoint binary trees (left and right).
- **Symmetric order.** Each node has a key, and every node's key is:
  - Larger than all keys in its left subtree.
  - Smaller than all keys in its right subtree.

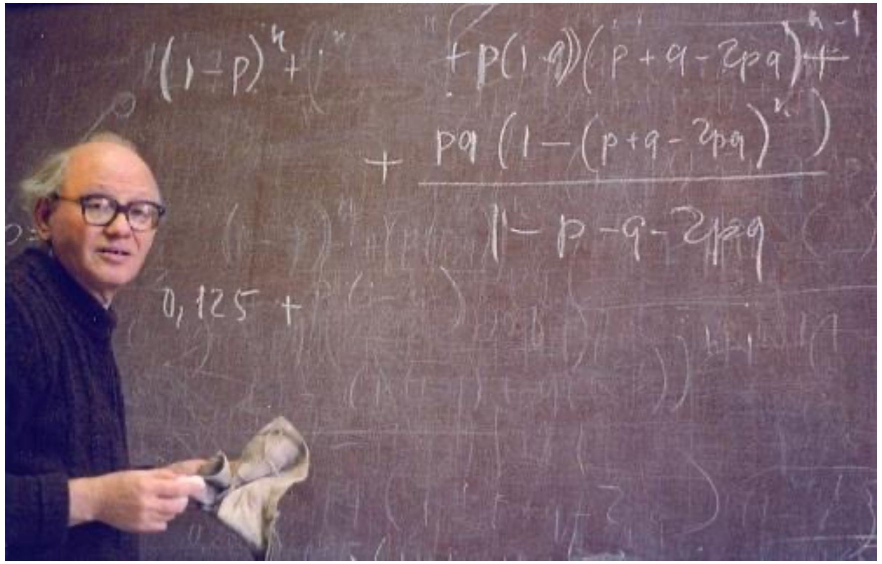





Trees
Avg case:
O(n)
Binary Trees
Avg case:
O(log n)
Worst case:
O(n)
BST Trees
Avg case:
O(log n)
Worst case:
O(n)
AVL Trees
Worst case:
O(log n)

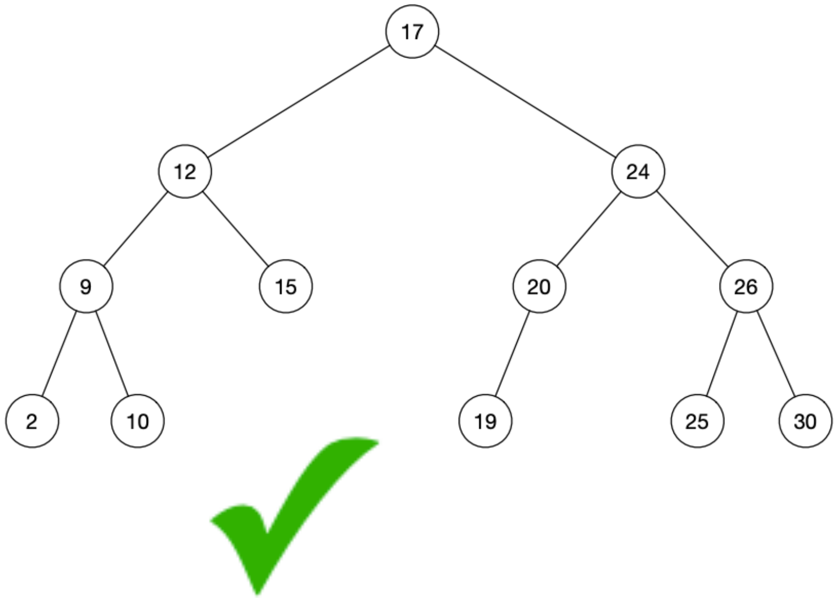
AVL TREES

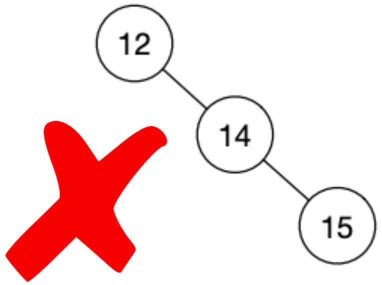
- **A**delson-**V**elsky and **L**andis (AVL)
- A Self Balancing Binary Search Tree
- Cost of Insert, Remove, Search is O(log n)

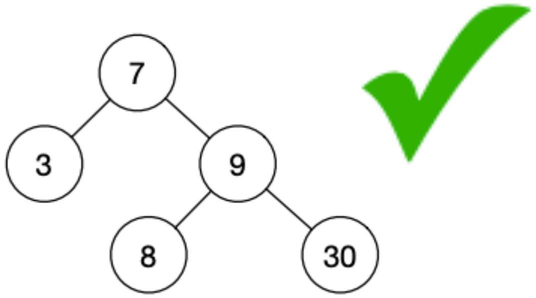


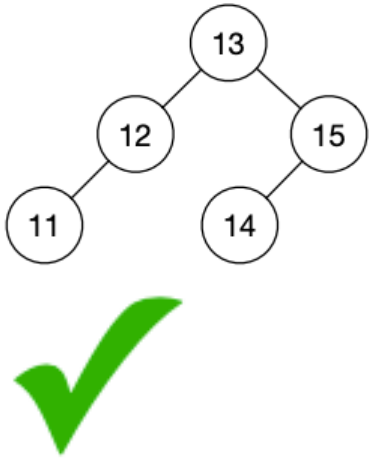
AVL TREES

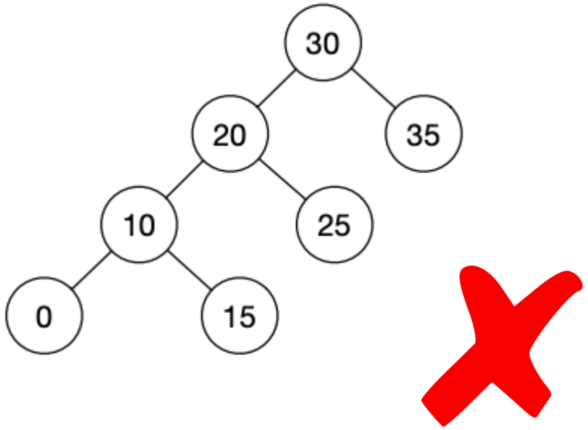
- Algorithm:
    1. Check Balanced Tree .i.e the **height difference should not exceed ONE**







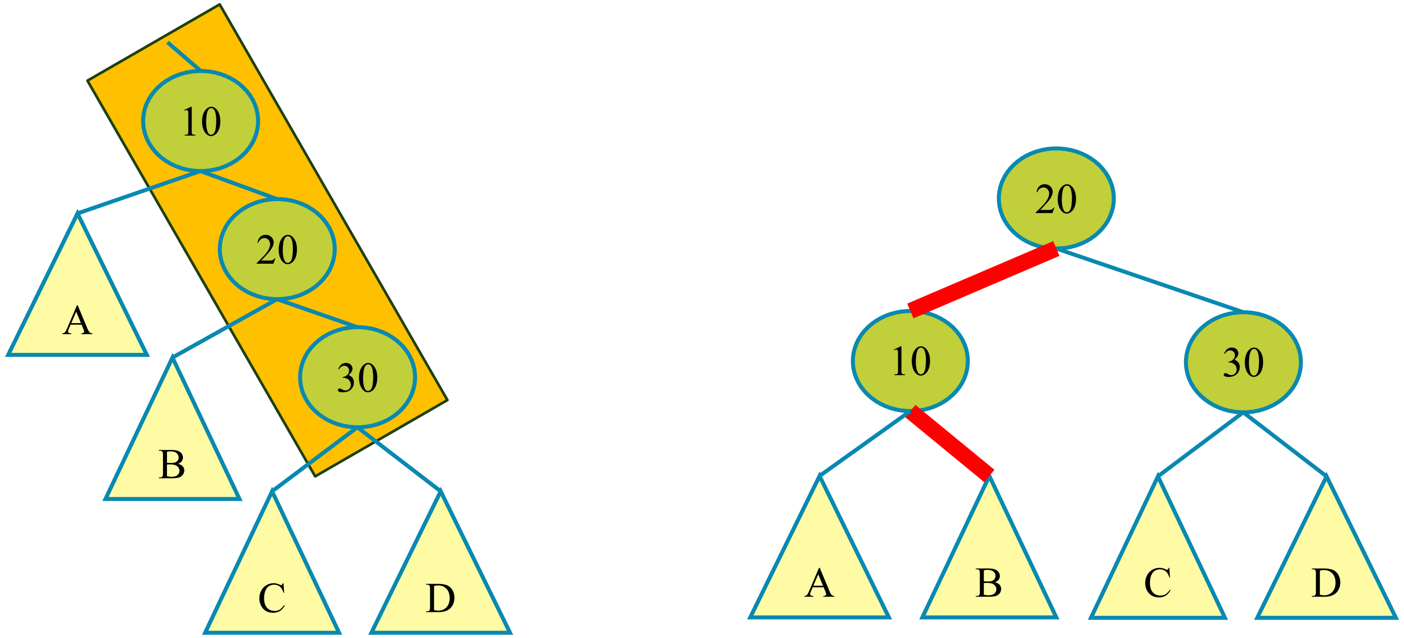




AVL TREES

- Algorithm:
   2. If not Balanced then **ROTATE**

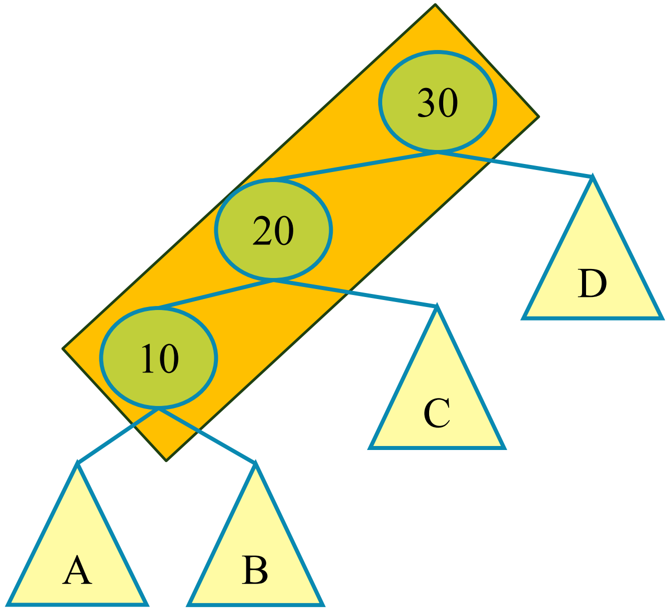
RR

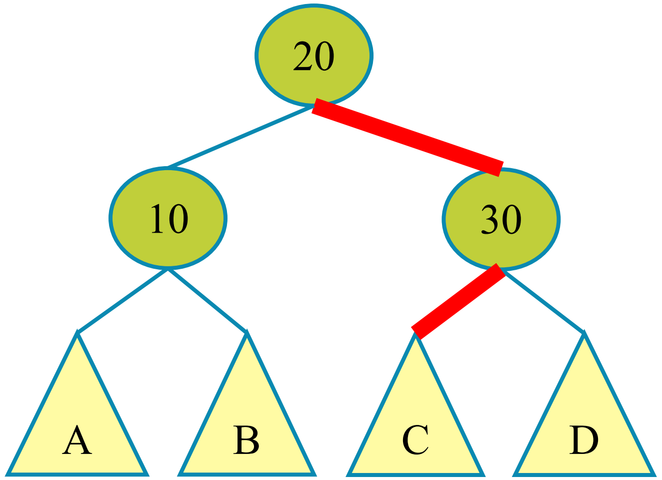


AVL TREES

- Algorithm:
  2. If not Balanced then **ROTATE**

LL

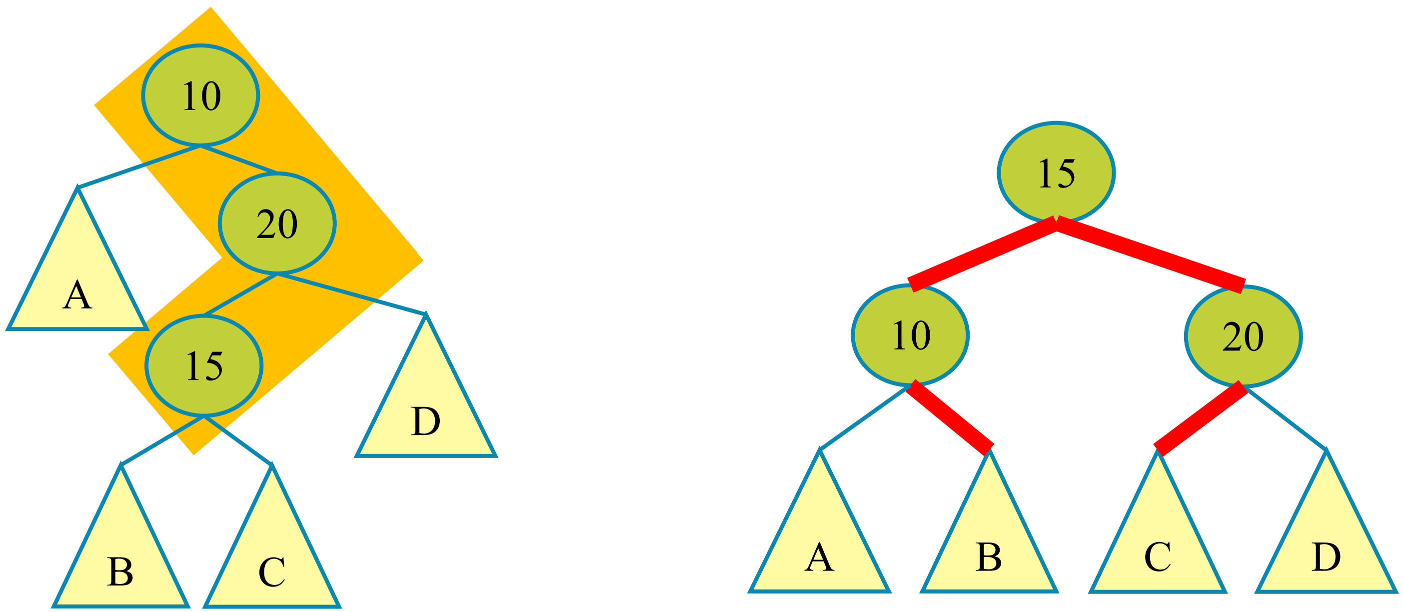




AVL TREES

- Algorithm:
  2. If not Balanced then **ROTATE**

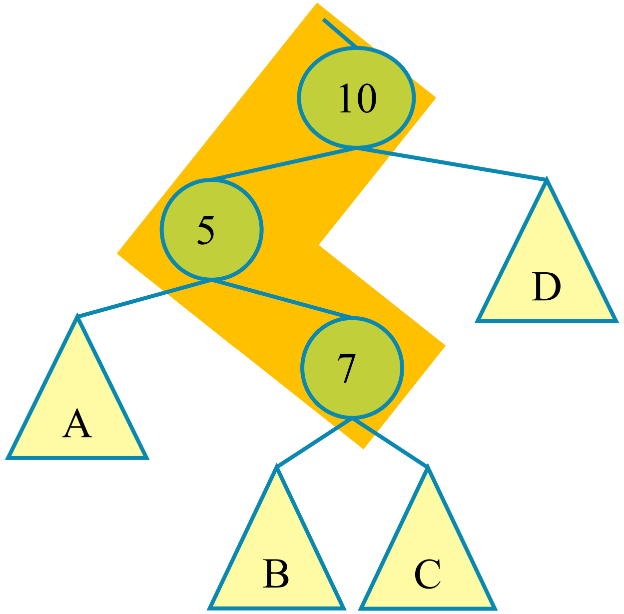
RL

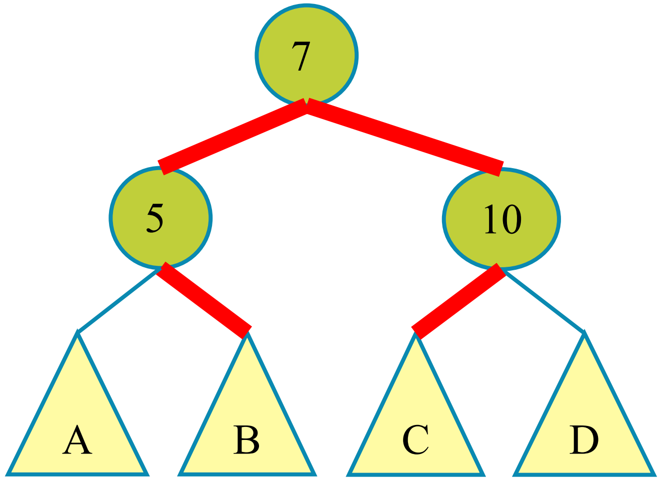


AVL TREES

- Algorithm:
  2. If not Balanced then **ROTATE**

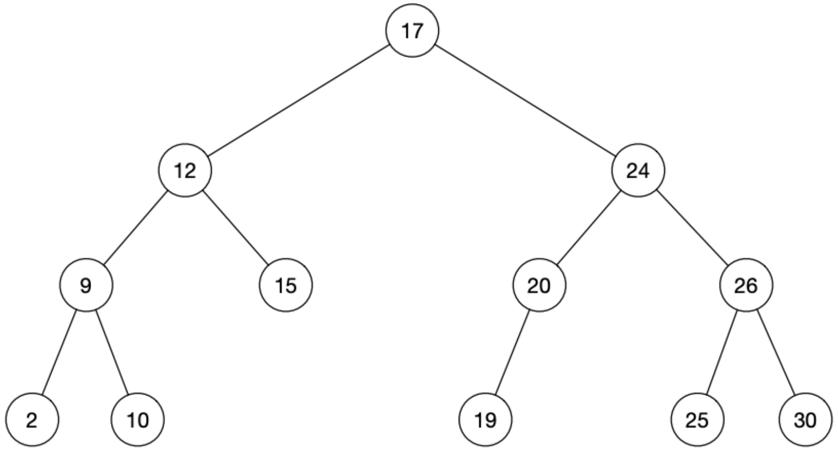
LR





AVL TREES

- Cost of Checking Height is O(log n).
  - The Check_height is conducted only when a node is inserted of removed.
  - The max number of nodes in a branch is log n, where n is the max number of nodes.
- Cost of a Rotation is constant
  - 2-4 operations per rotation
- The overall cost is O(log n) for insertion and removal
- The cost is O(log n) for search.



CS210: THE JOURNEY SO FAR

	Runtime		
Data Structure / Algorithm	Bestcase	Average Case	Worst Case
Singly Linked Lists	O(n)	O(n)	O(n)
Doubly Linked Lists	O(n)	O(n)	O(n)
Circular Linked Lists	O(n)	O(n)	O(n)
Stacks*	O(1)	O(1)	O(1)
Queues*	O(1)	O(1)	O(1)
Binary Search Trees	O(1)	O(1.39 log n)	O(n)
AVL Trees	O(1)	O(log n)	O(log n)
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2/2)$	$O(n^2/2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2/2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Tim Sort	O(n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(1.39 n log n)	$O(n^2/2)$