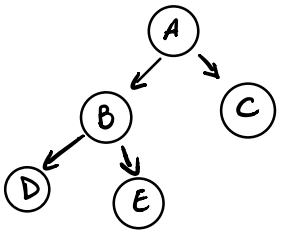
Tree Traversals



- Preorder: ABDEC
- Inorder: DBEAC
- Postorder: DEBCA
- Level-order: ABCDE

[LC] Left Child: $2n+1$; $N = RC/2 - 1$
[RC] Right Child: $2(n+1)$; $N = (LC-1)/2$

Recursion simplifies the implementation of tree traversals.

↳ Pre order: n | r
- visit node
- Traverse (left child)
- Traverse (Right Child)

↳ Postorder: l r n
- Traverse (left child)
- Traverse (Right Child)
- visit node

↳ Inorder: l n r
- Traverse (left child)
- Visit node
- Traverse (right child)

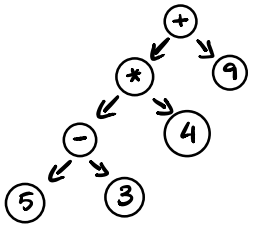
↳ Level-order: more complicated.
requires use of extra data structures (queues and/or lists) to create the necessary order.

↑ depends on type

Binary trees are ADT → has no add/delete

Expression Trees

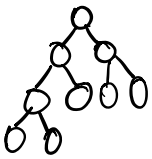
- shows the relationships among operators and operands in an expression
- It's evaluated from the bottom



→ (5-3) * 4 + 9

Full: every node has `0` or `2` children

Complete: all levels are completely filled, except possibly the last level and the last level has all nodes as left as possible. Top-to-bottom, left-to-right



Maximum number of nodes at level L: $2^{(L-1)}$
Maximum number of nodes in a tree of height H: $2^{(H+1)} - 1$

Binary Search Tree (BST)

- **search tree:**
  tree whose elements are organized to facilitate finding a particular element when needed.

- **binary search tree:**
  binary tree that, for each node n:
  * the left subtree of n contains elements less than the element stored in n.
  * the right subtree of n contains elements greater than or equal to the element stored in n.

- The particular shape of a binary search tree depends on the order in which the elements are added.
- The shape may also be dependant on any additional processing performed on the tree to reshape it.
- Binary search trees can hold any type of data, so long as we have a way to determine relative ordering
- Objects implementing the Comparable interface provide such capability.

ADDING

- Process of adding an element is similar to finding an element ⇒ adding ≈ finding
- New elements are added as leaf nodes
- Start at the root, follow path dictated by existing elements until you find no child in the desired direction.
- Then add the new element.

REMOVAL

- Removing a target in a BST is not as simple as that for linear data structures.
- After removing the element, the resulting tree must still be valid.
- Three distinct situations must be considered when removing an element:
  - ☐ The node to remove is a leaf. → It can simply be deleted
  - ☐ The node to remove has one child. → the deleted node is replaced by the child
  - ☐ The node to remove has two children. → an appropriate node is found lower in the tree and used to replace the node
    - * Good choice: inorder successor (node that would follow the removed node in an inorder traversal.
    - * The inorder successor is guaranteed not to have a left child.
    - * Thus, removing the inorder successor to replace the deleted node will result in one of the first situations (it's a leaf or has a child).