DS2030 Data Structures and Algorithms for Data Science
Week-4 Practice Set

August 27th, 2025

1 Binary Tree Traversals

You are required to implement a binary tree and perform different traversals.
Attributes:

- `data` : Value stored in the node.
- `left` : Left child.
- `right` : Right child.

Methods:

- `preorder(root)` : Print preorder traversal.
- `inorder(root)` : Print inorder traversal.
- `postorder(root)` : Print postorder traversal.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def preorder(root):
    # TO DO

def inorder(root):
    # TO DO

def postorder(root):
    # TO DO

# Example Tree
#       1
#      / \
#     2   3
#    / \
#   4   5
```

2 Construct Binary Tree from Traversals

Given the inorder and preorder traversals, reconstruct the binary tree.
Methods:

- `buildTree(preorder, inorder)` : Returns the root of the tree.

```
def buildTree(preorder, inorder):
    # TO DO

# Example:
# preorder = [1,2,4,5,3]
# inorder  = [4,2,5,1,3]
# Construct the tree and display inorder traversal.
```

3 Min-Heap Implementation

Implement a min-heap using both **top-down** (insertion with sift-up) and **bottom-up** (heapify) approaches.
Methods:

- `insert(key)` : Insert new key maintaining heap order.
- `removeMin()` : Remove and return the minimum element.
- `heapify(arr)` : Build heap from given array (bottom-up).
- `show()` : Display current heap.

```
class MinHeap:
    def __init__(self):
        self.heap = []

    def insert(self, key):
        # TO DO (top-down approach)

    def removeMin(self):
        # TO DO

    def heapify(self, arr):
        # TO DO (bottom-up approach)

    def show(self):
        print("Heap:", self.heap)
```

4 Max-Heap Implementation

Extend your heap implementation to **Max-Heap** with the same methods.

```
class MaxHeap:
    def __init__(self):
        self.heap = []

    def insert(self, key):
        # TO DO

    def removeMax(self):
        # TO DO

    def heapify(self, arr):
        # TO DO

    def show(self):
        print("Heap:", self.heap)
```

5 Testing

Write test cases for traversals, reconstruction, and heaps.

```
def test():
    # Test Tree Traversals
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)

    print("Preorder: ", end=""); preorder(root); print()
```

```
    print("Inorder: ", end=""); inorder(root); print()
    print("Postorder: ", end=""); postorder(root); print()

    # Test Tree Reconstruction
    preorder_seq = [1,2,4,5,3]
    inorder_seq  = [4,2,5,1,3]
    tree_root = buildTree(preorder_seq, inorder_seq)
    print("Constructed Inorder: ", end=""); inorder(tree_root); print()

    # Test MinHeap
    mh = MinHeap()
    mh.insert(5); mh.insert(3); mh.insert(8)
    mh.show()
    print("Removed:", mh.removeMin()); mh.show()

    # Test MaxHeap
    mx = MaxHeap()
    mx.insert(10); mx.insert(4); mx.insert(7)
    mx.show()
    print("Removed:", mx.removeMax()); mx.show()

# Run test
test()
```

Sample Output

```
Preorder: 1 2 4 5 3
Inorder: 4 2 5 1 3
Postorder: 4 5 2 3 1
Constructed Inorder: 4 2 5 1 3
Heap: [3, 5, 8]
Removed: 3
Heap: [5, 8]
Heap: [10, 4, 7]
Removed: 10
Heap: [7, 4]
```