Tutorial 3 (Week 7)
Binary Trees

Instructor: Dr. Quah T.S., Jon
Email: itsquah@ntu.edu.sg

Question 1 – Draw binary tree

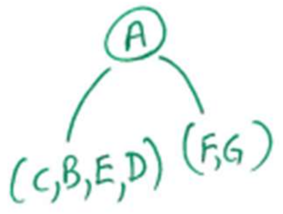
Pre-order: $\boxed{A}$, B, C, D, E, F, G

In-order: C, B, E, D, $\boxed{A}$, F, G

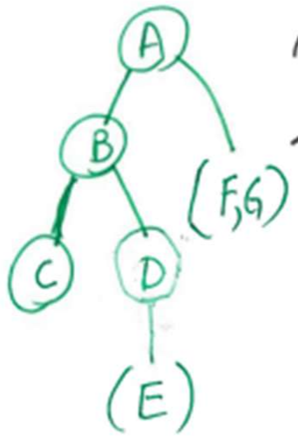
Step 1: "A" must be the root,
∵ it is the First element of Pre-order list

Step 2: "split" the In-order list.

C, B, E, D must be on left sub-tree
F, G must be on right sub-tree



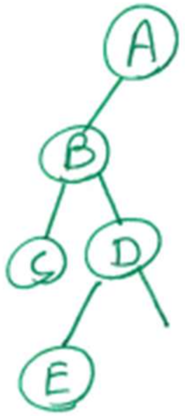
Step 3: next level, 2nd element, "B", of pre-order list should be root of next $\overset{\text{left}}{\wedge}$ sub-tree.



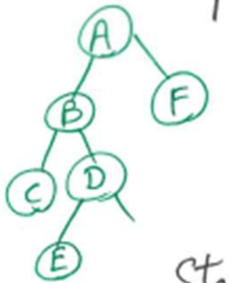
∴ "B" "split" In-order sub-list C, B, E, D into C on left sub-tree and E, D on right sub-tree.

Also, from pre-order list, "D" must be root of right sub-tree

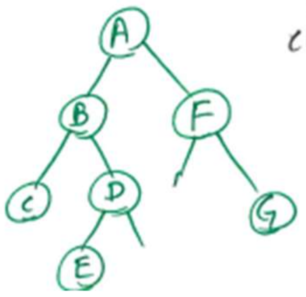
Step 4: since "E" comes before "D" on In-order list, it must be left child (Not right child) of "D"



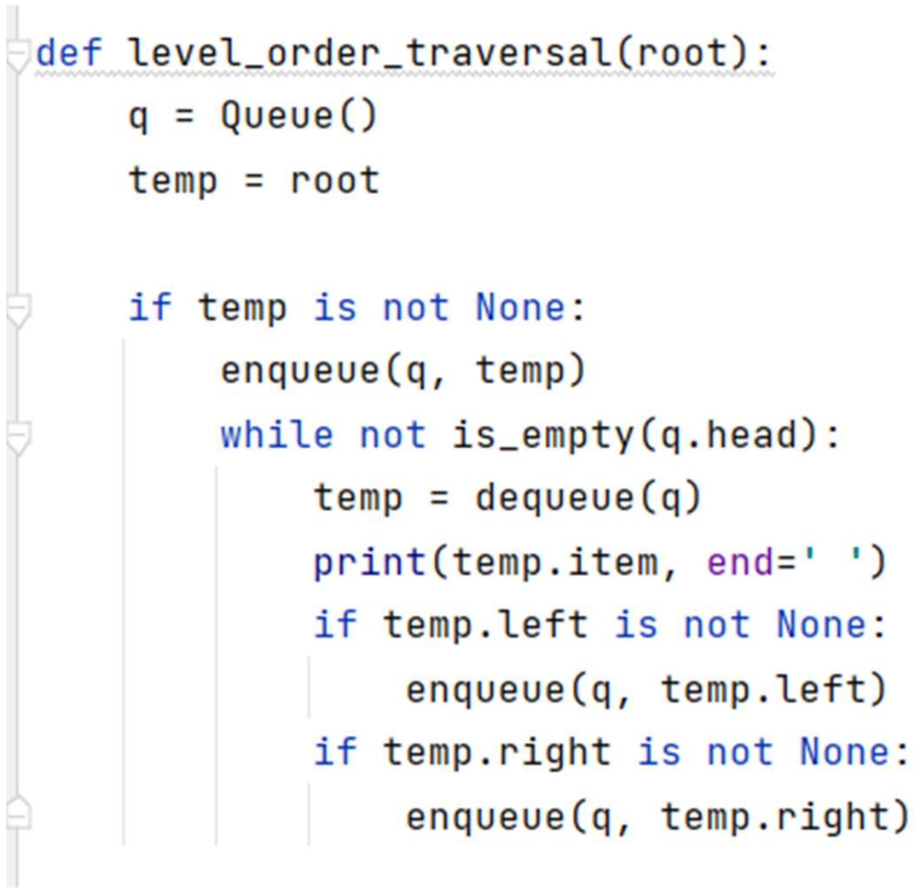
Step 5: From Pre-order list, we know "F" must be root of right subtree of "A"



Step 6: From In-order list, since "G" comes after "F", "G" must be right child (Not left child) of "F"

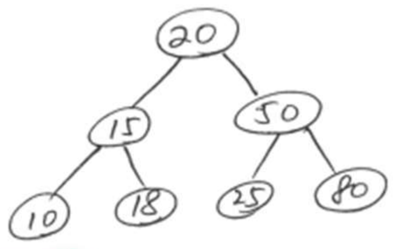


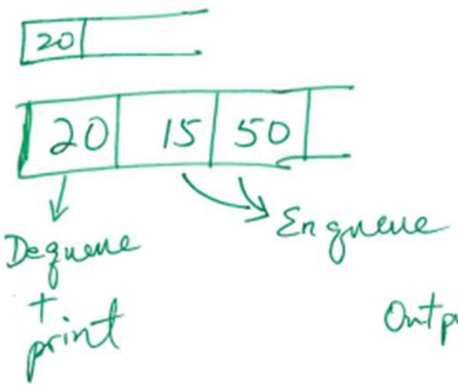
Question 2 - levelOrderTraversal



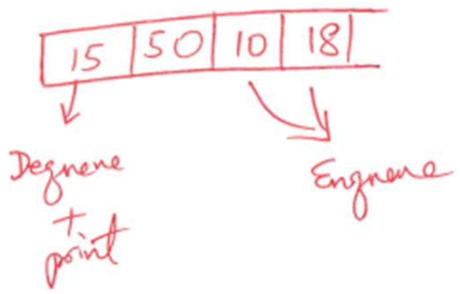
```python
def level_order_traversal(root):
    q = Queue()
    temp = root

    if temp is not None:
        enqueue(q, temp)
        while not is_empty(q.head):
            temp = dequeue(q)
            print(temp.item, end=' ')
            if temp.left is not None:
                enqueue(q, temp.left)
            if temp.right is not None:
                enqueue(q, temp.right)
```

Full program is uploaded to our tutorial folder

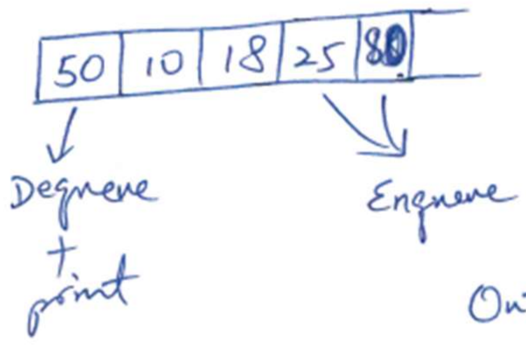




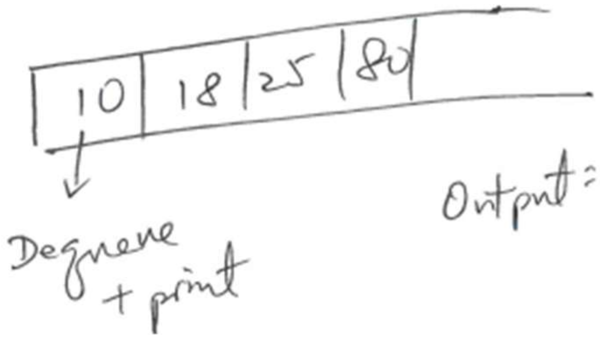
Output: 20



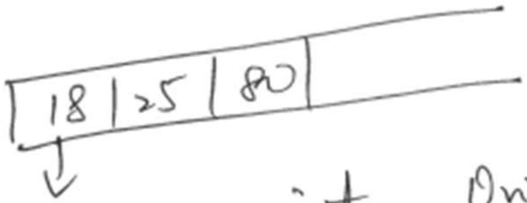
Output: 20, 15



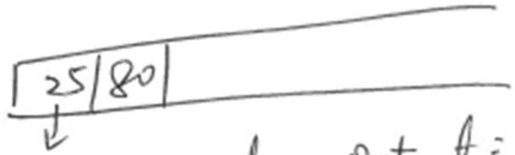
Output: 20, 15, 50



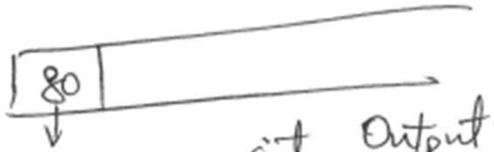
Output: 20, 15, 50, 10



Dequeue + print   Output: 20, 15, 50, 10, 18

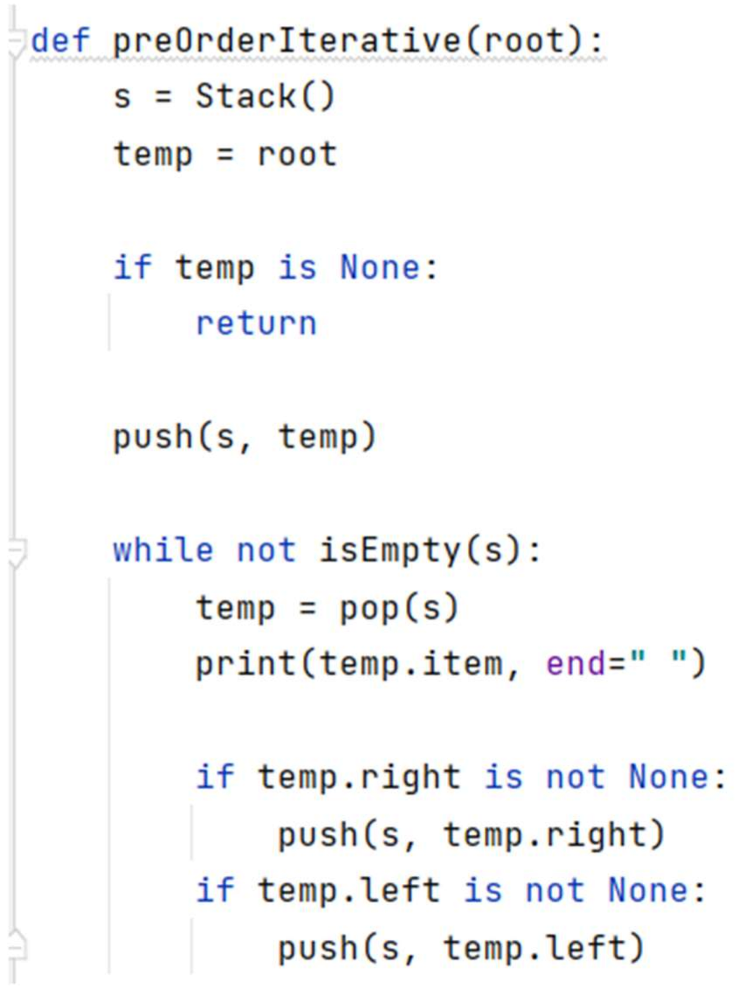


Dequeue + print   Output: 20, 15, 50, 10, 18, 25



Deque + print   Output: 20, 15, 50, 10, 18, 25, 80

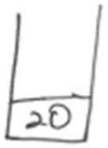
Question 3 - preOrderIterative



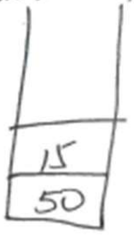
```
def preOrderIterative(root):
    s = Stack()
    temp = root

    if temp is None:
        return

    push(s, temp)

    while not isEmpty(s):
        temp = pop(s)
        print(temp.item, end=" ")

        if temp.right is not None:
            push(s, temp.right)
        if temp.left is not None:
            push(s, temp.left)
```

Full program is uploaded to our tutorial folder

Step 1:



Step 2: Iteration 1



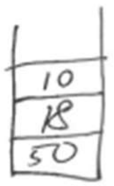
pop 20 + (print)

push 50

push 15

Due to LIFO, push "Right" child first.

Iteration 2

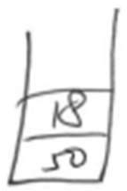


pop 15 + (print)

push 18

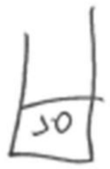
push 10

Iteration 3



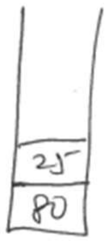
pop 10 + (print)

Iteration 4



pop 18 + (print)

Iteration 5

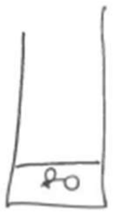


pop 50 + (print)

push 80

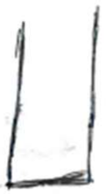
push 25

Iteration 6



pop 25 + (print)

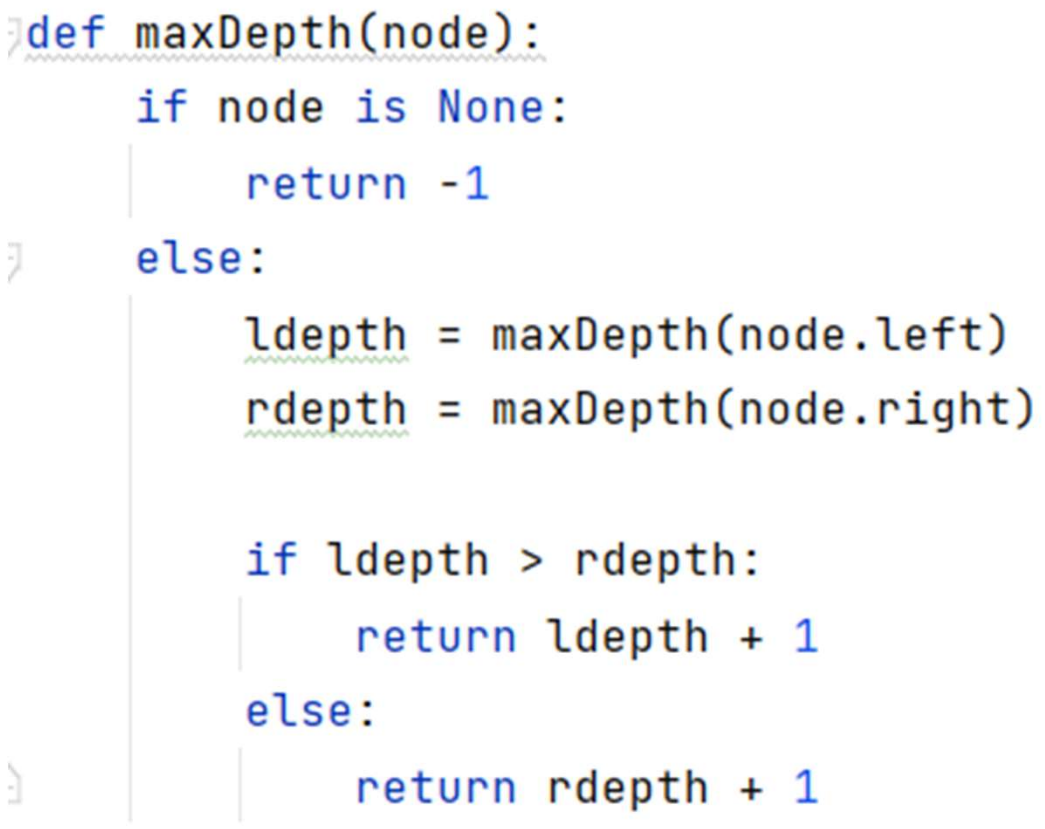
Iteration 7



empty

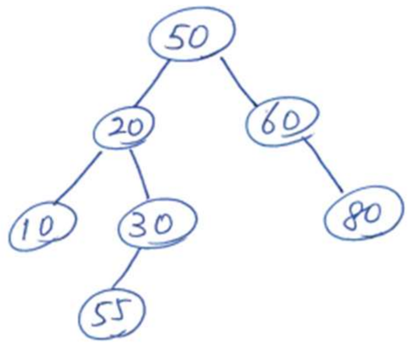
pop 80 + (print)

Question 4 - maxDepth



```python
def maxDepth(node):
    if node is None:
        return -1
    else:
        ldepth = maxDepth(node.left)
        rdepth = maxDepth(node.right)

        if ldepth > rdepth:
            return ldepth + 1
        else:
            return rdepth + 1
```

Full program is uploaded to our tutorial folder



Resulting b-tree: Inorder: 10, 20, 55, 30, 50, 60, 80

Iteration:
(Recursive)

```
maxD (50)
        ldepth = maxD (20) = 2
        rdepth = maxD (60) = 1

        ∴ return ldepth + 1 = 2+1 = 3
```

maxD (20)

ldepth = maxD (10) = 0

rdepth = maxD (30) = 1

∴ return rdepth + 1 = 1+1
= 2

maxD (10)

ldepth = -1

rdepth = -1

∴ return rdepth+1 = -1+1 = 0

maxD (30)

l depth = maxD (55) = 0

r depth = -1

∴ return ldepth +1 = 0+1 = 1

maxD (55)

ldepth = -1
rdepth = -1

∴ return rdepth + 1 = -1 + 1 = 0

maxD (60)

ldepth = -1
rdepth = maxD (80) = 0

∴ return rdepth + 1 = 0 + 1
= 1

maxD (80)

ldepth = -1
rdepth = -1

∴ return rdepth + 1 = -1 + 1 = 0