1. Given the preorder traversal of a binary search tree is as follows: 9 5 4 3 7 6 8 13 11 12 15…

a. Draw the described tree.

   9
  / \  
 5   13
 / \ / \  
4 7 11 15
 / / \ \  
3 6 8 12

b. Now that you have the tree, what is its postorder traversal?

   3 4 6 8 7 5 12 11 15 13 9
2. Given a min heap with the structure as defined below, redraw the minheap after performing the following operations…

```
5
/   \
6   7
/   /   \
9   10 11 15
/   /   \
17 18
```

a. Insert 8

```
5
/   \
6   7
/   /   \
9   8 11 15
/   /   \
17 18 10
```

b. Delete min

```
6
/   \n8   7
/   /   \n9 10 11 15
/   \
```
3. Write a function which recursively determines if a binary tree is balanced or not. By balanced, we mean that the right and left subtrees have at most a difference in height of 1. Note: You may use a helper if this helps you.

```python
def is_height_balanced(root):
    def balanced_helper(root):
        if not(root.left) and not(root.right):
            return (1, True)
        elif not(root.right):
            right = balanced(root.right)
            if right[0] > 1:
                return (right[0] + 1, False)
            else:
                bal = right[1]
                return (right[0] + 1, bal)
        elif not(root.left):
            left = balanced(root.left)
            if left[0] > 1:
                return (left[0], False)
            else:
                bal = left[1]
                return (left[0] + 1, bal)
        else:
            right = balanced(root.right)
            left = balanced(root.left)
            if left[0] > 1:
                return (left[0], False)
            else:
                bal = left[1]
                return (left[0] + 1, bal)
        return (1, True)
```
left = balanced(root.left)

bal = right[1] and left[1]

height_condition = abs(right[0] - left[0]) <= 1

return (max(right[0], left[0])+1, bal and height_condition)

if not(root):
    raise Exception("Empty Tree")
else:
    return balanced_helper(root)[1]

4. Define a non-recursive function which will print out the specified level of a tree, using only a stack and a queue and constant additional space.

def print_tree_level(root, level):
    q = ArrayQueue()
    s = ArrayStack()
    q.enqueue(root)
    count = 1
    while not(q.is_empty()) and count < level:
        curr = q.dequeue()
        if curr.left:
            s.push(curr.left)
        if curr.right:
            s.push(curr.right)
        if q.is_empty():
            count += 1
            while not(s.is_empty()):
                q.enqueue(s.pop())
    if q.is_empty():
        return
    else:
        while not(q.is_empty()):
            print(q.dequeue().data)
5. Draw the resultant Hash Table after inserting the below items into the table (of size 13), with hash function $h(k) = k \mod 13$. Use linear probing to deal with the collisions.

Insert 32, 5, 23, 29, 26, 41, 39, 42, 17, 19

26, 29, 41, 29, 42, 5, 32, 17, 19, ___, 23, ___, ___