```
import os
from collections import deque

class Node(object):
    def __init__(self, obj):
        self.value = obj
        self.left = None
        self.right = None


class BinaryTree(object):
    def __init__(self):
        self.root = None

    def insert(self, obj):
        """
            Inserts the data as a node into the Binary Tree
        """
        if not self.root:
            self.root = Node(obj)
        else:
            current = self.root
            while current:
                if obj < current.value:
                    if not current.left:
                        current.left = Node(obj)
                        return
                    else:
                        current = current.left
                else:
                    if not current.right:
                        current.right = Node(obj)
                        return
                    else:
                        current = current.right

    def insert_many(self, *args):
        """
            Helper function to create and insert many elements into the binary tree in
            one go
        """

        for arg in args:
            self.insert(arg)

    def is_symmetric_recursive(self):
        def is_sym(node1, node2):
            if not node1 and not node2:
                return True
            if not node1 or not node2:
                return False
            return (is_sym(node1.left, node2.right) and is_sym(node1.right,node2.left))
        return is_sym(self.root.left, self.root.right)

    def is_symmetric(self):
```

```
    """
        This will check if the binary tree is symmetric or not.

        Returns a bool
    """

    is_symmetric_flag = True

    # Create 2 deques - for the left subtree and the right subtree
    left_deque = deque()
    right_deque = deque()

    # Add the subroot , if exists
    if self.root.left:
        left_deque.append(self.root.left)
    if self.root.right:
        right_deque.append(self.root.right)

    # This is used for providing the symmetric check at each level
    #  for both the left subtree and the right subtree
    # returns a tuple
    def get_chk(node1,node2,deque):
        chk = []
        if node1:
            chk.append('1')
            deque.append(node1)
        else:
            chk.append('0')
        if node2:
            chk.append('1')
            deque.append(node2)
        else:
            chk.append('0')
        return tuple(chk)

    # Keep consuming from the deques, until one runs out of entries
    while left_deque and right_deque:
        current_left = left_deque.popleft()
        current_right = right_deque.popleft()
        # For the left subtree node - check from left to right
        left_chk = get_chk(current_left.left,current_left.right,left_deque)
        # For the right subtree node - check from right to left,
        # thus getting the mirror
        right_chk = get_chk(current_right.right,current_right.left,right_deque)
        print current_left.value, current_right.value
        print left_chk, right_chk
        # Now match
        if left_chk != right_chk:
            is_symmetric_flag = False              # Mirrored match is false
            break
    # When one of the deques runs out, check if any more are present in the other one.
    # If so then the binary tree can never be symmetrical,
    # because both the left subtree and the right subtree should have equal number
    of nodes.
    if left_deque or right_deque:
```

```
            is_symmetric_flag = False
        return is_symmetric_flag



if __name__ == '__main__':
    t = BinaryTree()
    t.insert_many( 6, 3, 1, 2, 5, 4, 10, 7, 8, 12, 11 )
    print " Binary Tree 1 is symmetric ? %s " % t.is_symmetric()

    s = BinaryTree()
    s.insert_many( 6, 3, 1, 0, 4, 5, 10, 8, 7, 12, 13)
    print " Binary Tree 2 is symmetric ? %s " % s.is_symmetric()

    v = BinaryTree()
    v.insert_many( 6, 2, 1, 4, 5, 3, 10, 7, 6, 12, 13)
    print " Binary Tree 3 is symmetric ? %s " % v.is_symmetric()

    q = BinaryTree()
    q.insert_many( 'D','A','M','A','M')

    print " Binary Tree 4 is symmetric ? %s " % q.is_symmetric()

# H:\safe_desktop_items\projects\python\ds>python binarytree.py
# 3 10
# ('1', '1') ('1', '1')
# 1 12
# ('0', '1') ('0', '1')
# 5 7
# ('1', '0') ('1', '0')
# 2 11
# ('0', '0') ('0', '0')
# 4 8
# ('0', '0') ('0', '0')
 # Binary Tree 1 is symmetric ? True
# 3 10
# ('1', '1') ('1', '1')
# 1 12
# ('1', '0') ('1', '0')
# 4 8
# ('0', '1') ('0', '1')
# 0 13
# ('0', '0') ('0', '0')
# 5 7
# ('0', '0') ('0', '0')
 # Binary Tree 2 is symmetric ? True
# 2 10
# ('1', '1') ('1', '1')
# 1 12
# ('0', '0') ('1', '0')
 # Binary Tree 3 is symmetric ? False
# A M
# ('0', '1') ('1', '0')
 # Binary Tree 4 is symmetric ? False
```