class Node:

def \_\_init\_\_(self, info):

self.info = info

self.left = None

self.right = None

self.level = None

def \_\_str\_\_(self):

return str(self.info)

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

def create(self, val):

if self.root == None:

self.root = Node(val)

else:

current = self.root

while True:

if val < current.info:

if current.left:

current = current.left

else:

current.left = Node(val)

break

elif val > current.info:

if current.right:

current = current.right

else:

current.right = Node(val)

break

else:

break

"""

Node is defined as

self.left (the left child of the node)

self.right (the right child of the node)

self.info (the value of the node)

"""

from collections import deque

def levelOrder(root):

if root is None:

print("")

return

myLevel = ""

#breadth first search (i.e. make queue)

queue = deque([root])

while queue:

#remove from queue

node = queue.popleft()

#print

myLevel += node.\_\_str\_\_() + " "

#add its children to queue

if node.left is not None:

queue.append(node.left)

if node.right is not None:

queue.append(node.right)

print(myLevel)

tree = BinarySearchTree()

t = int(input())

arr = list(map(int, input().split()))

for i in range(t):

tree.create(arr[i])

levelOrder(tree.root)