INLUSTRO PRACTICAL EXAMINTION

# Data Structures and Algorithms

A R Vaishnavi

813818104105

III CSE-B

## Finding the size of binary tree using level order traversal

Level order traversal - Traversing the nodes level by level also known as breadth first traversal

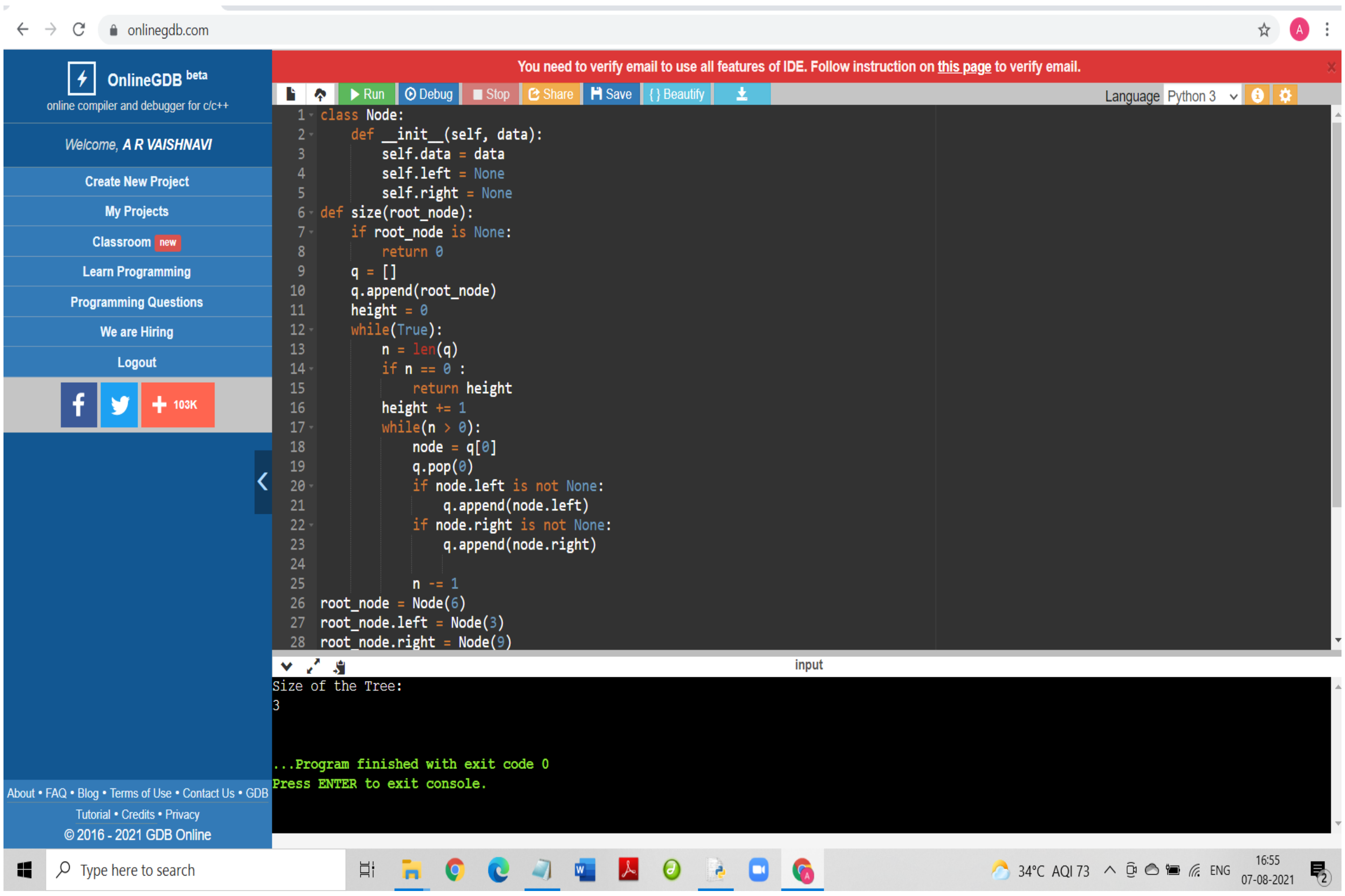
ALGORITHM :

```
Step 1: Create a queue.
Step 2: Push root_node into the queue.
Step 3: set height = 0
Step 4: Repeat the below steps until True
Step 4.1: Assign n to size of queue
step 4.2: if n is 0 goto step 4.2.1 else goto step 4.2.2
step 4.2.1: return Height
step 4.2.2: increase Height
step 4.3: Repeat the below steps until n>0:
Step 4.3.1: pop node from front
Step 4.3.2: push its children to queue
```

Step 4.3.3: decrease n

Step 5: Display the value returned by height

SCREENSHOTS:

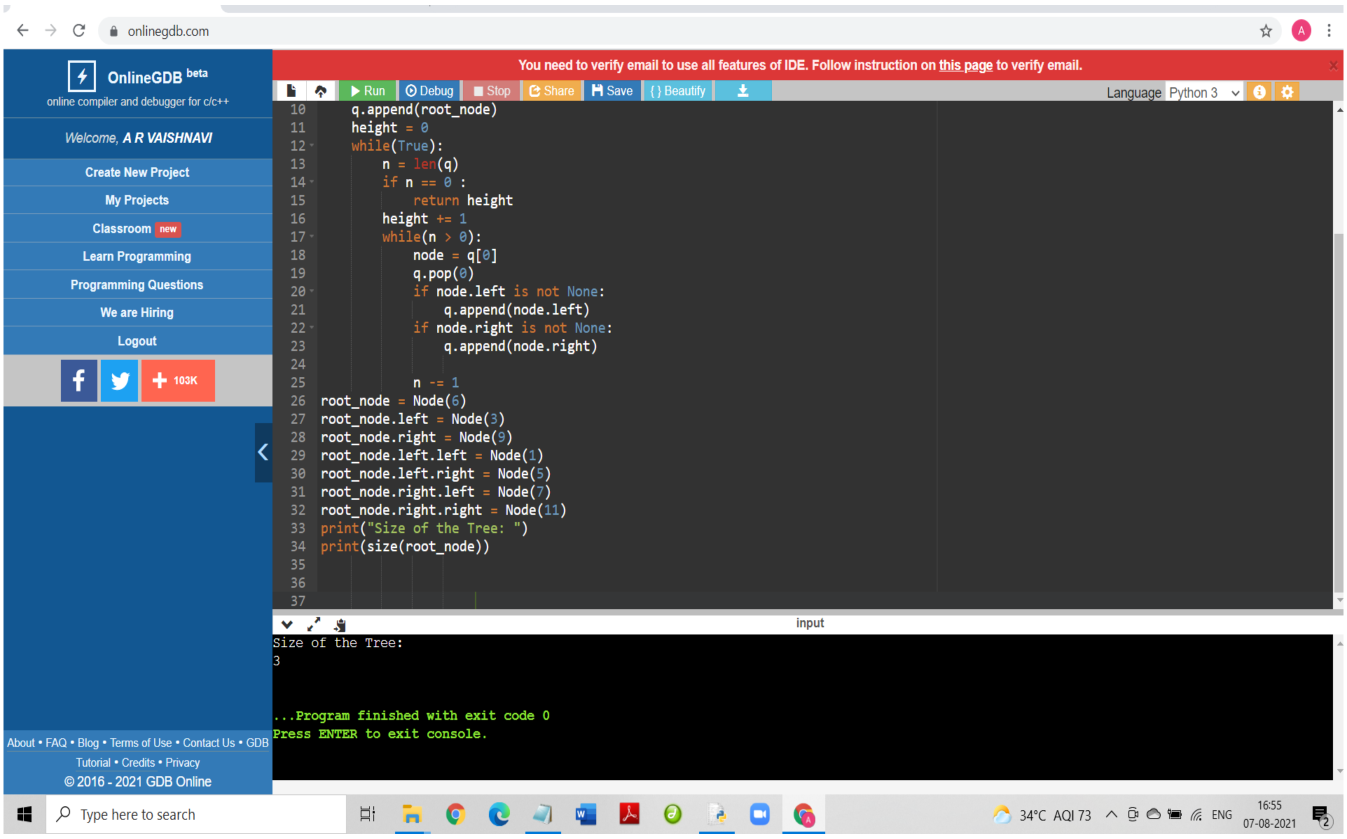


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def size(root_node):
    if root_node is None:
        return 0
    q = []
    q.append(root_node)
    height = 0
    while(True):
        n = len(q)
        if n == 0 :
            return height
        height += 1
        while(n > 0):
            node = q[0]
            q.pop(0)
            if node.left is not None:
                q.append(node.left)
            if node.right is not None:
                q.append(node.right)

            n -= 1
root_node = Node(6)
root_node.left = Node(3)
root_node.right = Node(9)
```

```
Size of the Tree:
3


...Program finished with exit code 0
Press ENTER to exit console.
```



```
    q.append(root_node)
    height = 0
    while(True):
        n = len(q)
        if n == 0 :
            return height
        height += 1
        while(n > 0):
            node = q[0]
            q.pop(0)
            if node.left is not None:
                q.append(node.left)
            if node.right is not None:
                q.append(node.right)

            n -= 1
root_node = Node(6)
root_node.left = Node(3)
root_node.right = Node(9)
root_node.left.left = Node(1)
root_node.left.right = Node(5)
root_node.right.left = Node(7)
root_node.right.right = Node(11)
print("Size of the Tree: ")
print(size(root_node))
```

```
Size of the Tree:
3


...Program finished with exit code 0
Press ENTER to exit console.
```

## Program to find the merging point of linked list

APPROARCH:

The number of nodes in traversing the first linked list and then from the head of the second linked list to intersection point is equal to the number of nodes involved in traversing the second linked list and then from head of the first list to the intersection point.

ALGORITHM

```
Step 1: create node class
Step 2: In mergingPoint()
  step 2.1: Assign a to head1 and b to head2
  step 2.2: Repeat the below steps until a!=b else goto step 2.3
  step 2.2.1: if a is None goto step 2.2.2 else goto step 2.2.3
  step 2.2.2: assign a to head of second linked list
  step 2.2.3: Else update the pointer to next node
  step 2.2.4: if b is None goto step 2.2.5 else goto step 2.2.6
  step 2.2.5: assign b to head of first linked list
  step 2.2.6: Else update the pointer to next node
  Step 2.3: return the data pointed
  Step 3: Display the returned value
```

2. Can we solve this using the sorting technique?

We can combine two techniques to get a solution to the problem. This can be done as:-

Create an array and store all the addresses of the nodes.

Now sort this array.

For each element in the second list, from the beginning search for the address in the array.

We can use a very efficient search algorithm like Binary Search which gives us the result in O(log n).

If we find a same memory address, that means that is the merging point of the 2 lists.

3. Can we solve it using hash tables?

Yes we can solve it using hash table. This method is similar to the brute force method. The following steps are involved in it:-

- Select a List, that has fewer number of elements. We can get the number of elements, by a single scan on both the lists. If both the lists have same number of elements, select any list at random.
- Create a hash table using the list with fewer elements. Creating a hash table means storing the address of each of the nodes of the smaller list in a separate data structure such as an array.

- Now, traverse the other list and compare the address of each of the node with the values in the hash table.
- If there exists an intersection point, certainly we will find a match in the hash table and we will obtain the intersection point

4. Can we use stacks for solving?

Yes, we can solve this problem using the concept of stacks. A stack is a data structure which follows the principle of LAST IN FIRST OUT (LIFO).

5. Is there any other way of solving this ?

Yes, we can use Brute force method to solve this

6. Can we improve the complexity for?

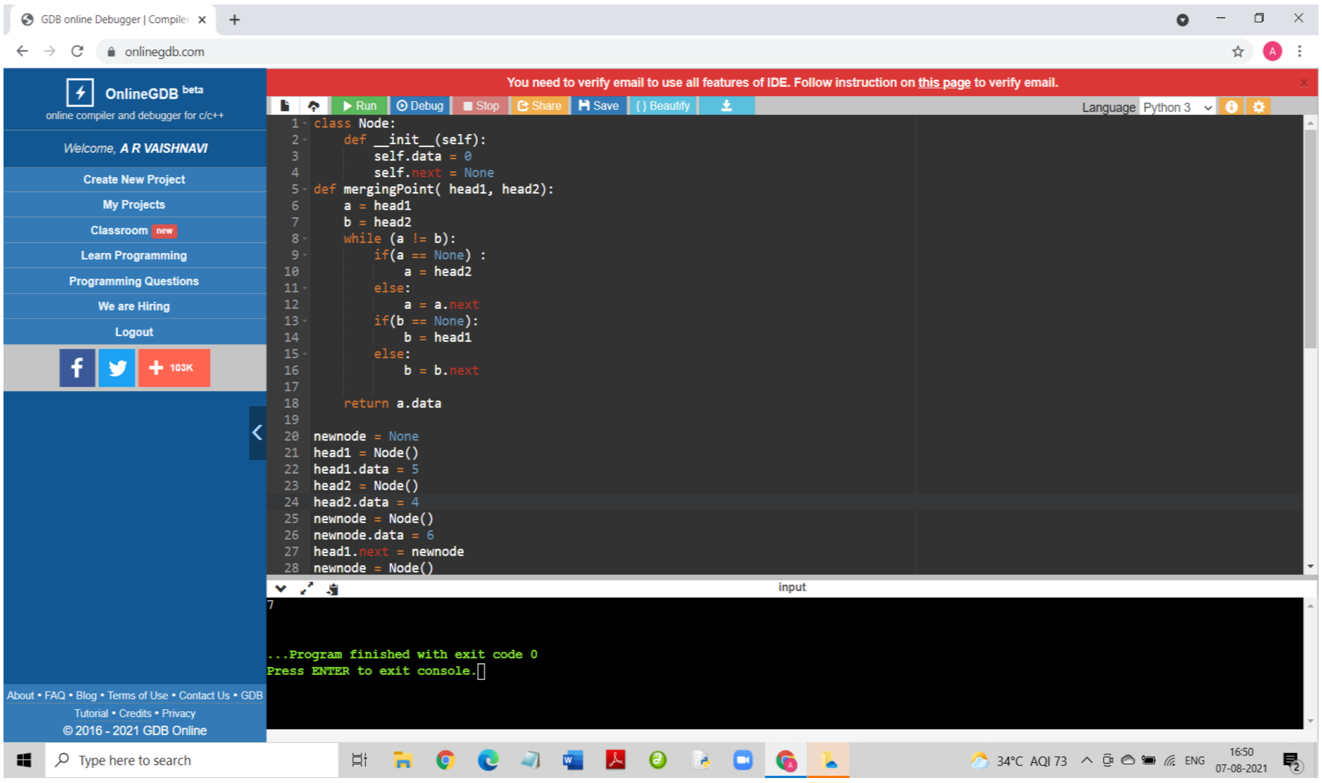
Yes, we can improve complexity.

The steps involved are:-

- Find the length of both the lists. Let ‘m’ be the length of List 1 and ‘n’ be the length of List 2.
- Find the difference in length of both the lists. d = m – n
- Move ahead ‘d’ steps in the longer list.
- This means that we have reached a point after which, both of the lists have same number of nodes till the end.
- Now move ahead in both the lists in parallel, till the ‘NEXT’ of both the lists is not the same.
- The NEXT at which both the lists are same is the merging point of both the lists.

Time Complexity: O(max(m,n))

Space Complexity: O(1)



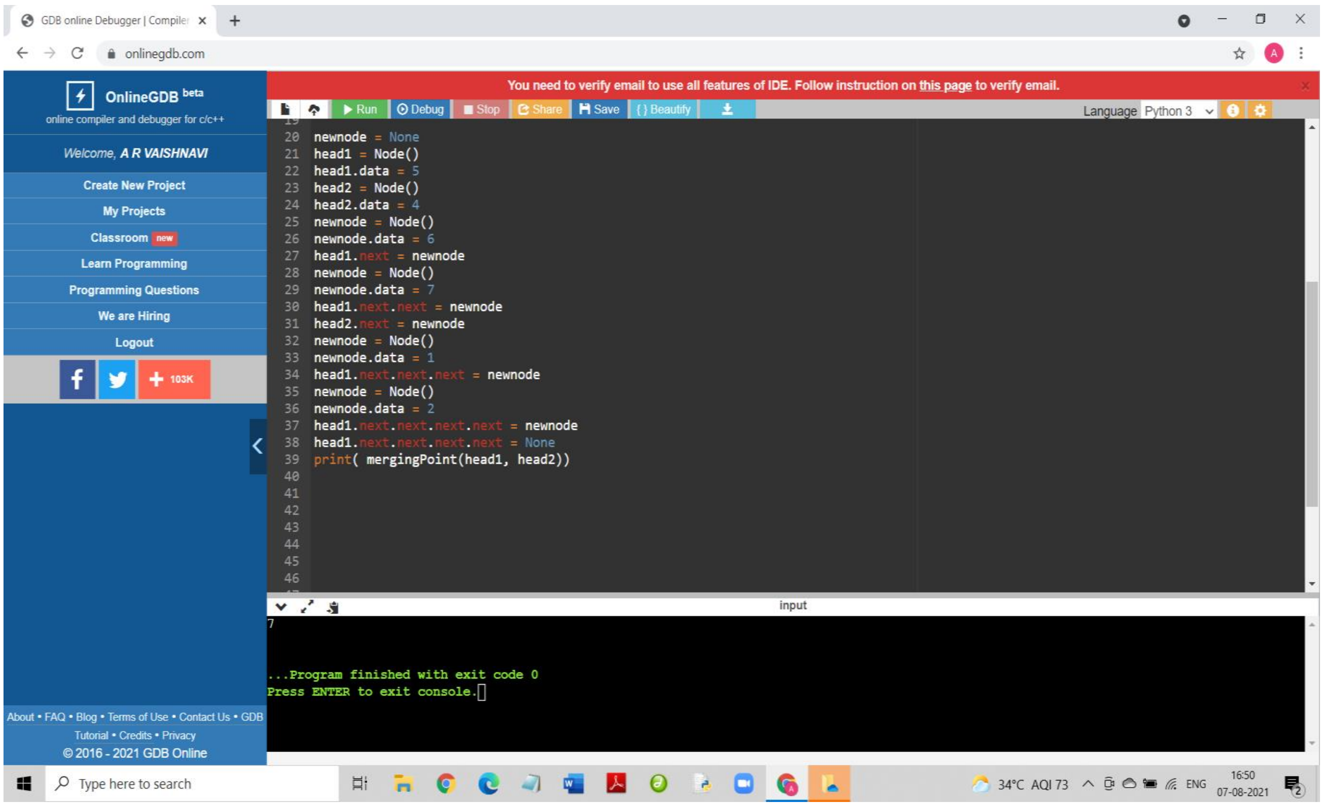
```python
class Node:
    def __init__(self):
        self.data = 0
        self.next = None
def mergingPoint( head1, head2):
    a = head1
    b = head2
    while (a != b):
        if(a == None) :
            a = head2
        else:
            a = a.next
        if(b == None):
            b = head1
        else:
            b = b.next

    return a.data

newnode = None
head1 = Node()
head1.data = 5
head2 = Node()
head2.data = 4
newnode = Node()
newnode.data = 6
head1.next = newnode
newnode = Node()
```

```
7


...Program finished with exit code 0
Press ENTER to exit console.
```



```
newnode = None
head1 = Node()
head1.data = 5
head2 = Node()
head2.data = 4
newnode = Node()
newnode.data = 6
head1.next = newnode
newnode = Node()
newnode.data = 7
head1.next.next = newnode
head2.next = newnode
newnode = Node()
newnode.data = 1
head1.next.next.next = newnode
newnode = Node()
newnode.data = 2
head1.next.next.next.next = newnode
head1.next.next.next.next.next = None
print( mergingPoint(head1, head2))
```

```
7


...Program finished with exit code 0
Press ENTER to exit console.
```

## GITHUB REPOSITORY LINK

[https://github.com/VaishnaviChandran8/Inlustropracticals-repository.git](https://github.com/VaishnaviChandran8/Inlustropracticals-repository.git)