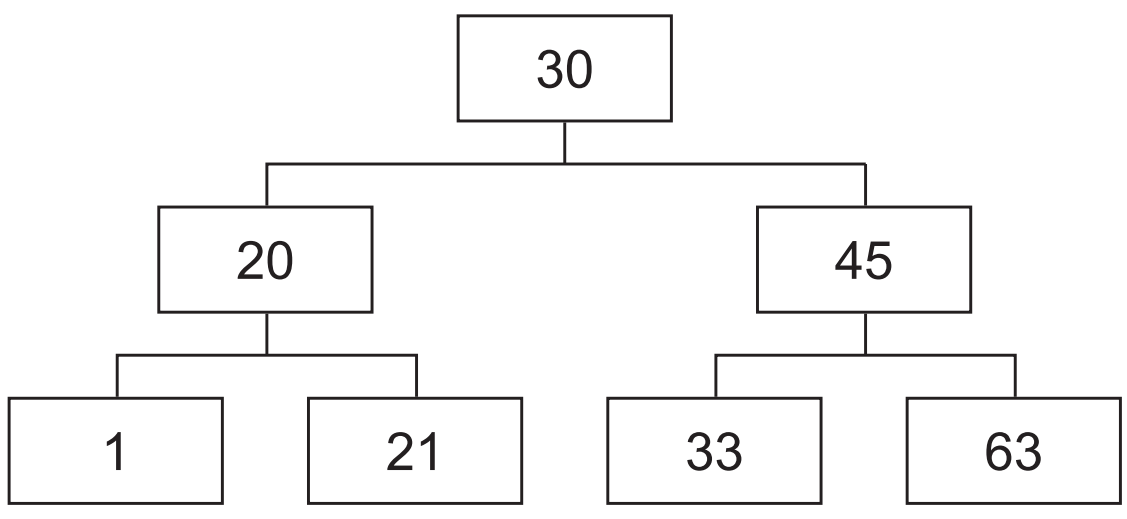
3 A program stores data in a binary tree that is designed using Object-Oriented Programming (OOP).

The tree stores data in ascending numerical order, for example:



The class `Node` stores data about the nodes.

Node	
`NodeData : Integer`	stores the node's integer data
`LeftNode : Node`	stores the node that is stored to the left of the current node, or a null value if there is no node to the left
`RightNode : Node`	stores the node that is stored to the right of the current node, or a null value if there is no node to the right
`Constructor()`	initialises `NodeData` to its parameter value; initialises `LeftNode` and `RightNode` to a null value
`GetLeft()`	returns `LeftNode`
`GetRight()`	returns `RightNode`
`GetData()`	returns `NodeData`
`SetLeft()`	takes an object of type `Node` as a parameter and stores it in `LeftNode`
`SetRight()`	takes an object of type `Node` as a parameter and stores it in `RightNode`

(a) (i) Write program code to declare the class `Node` and its constructor.

Do **not** declare the other methods.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program as **Question3_J25**.

Copy and paste the program code into part **3(a)(i)** in the evidence document.

[4]

(ii) Write program code for the **three** get methods.

Save your program.

Copy and paste the program code into part **3(a)(ii)** in the evidence document.

[3]

(iii) The method `SetLeft()` takes an object of type `Node` as a parameter. The method stores the parameter in the attribute `LeftNode`

The method `SetRight()` takes an object of type `Node` as a parameter. The method stores the parameter in the attribute `RightNode`

Write program code for `SetLeft()` and `SetRight()`

Save your program.

Copy and paste the program code into part **3(a)(iii)** in the evidence document.

[3]

(b) Write the main program to declare **five** objects of type `Node` :

- Node 1 with the data `10`
- Node 2 with the data `20`
- Node 3 with the data `5`
- Node 4 with the data `15`
- Node 5 with the data `7`

Save your program.

Copy and paste the program code into part **3(b)** in the evidence document.

[2]

(c) The class `Tree` stores the tree.

Tree	
`FirstNode : Node`	stores the root node in the tree
`Constructor()`	initialises `FirstNode` to its parameter value
`GetRootNode()`	returns the node stored in `FirstNode`
`Insert()`	stores its parameter node in the correct position in the tree

(i) Write program code to declare the class `Tree` and its constructor.

Do **not** declare the other methods.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program.

Copy and paste the program code into part **3(c)(i)** in the evidence document.

[2]

(ii) Write program code for `GetRootNode()`

Save your program.

Copy and paste the program code into part **3(c)(ii)** in the evidence document.

[1]

(iii) The method `Insert()` takes a node as a parameter and then searches the tree to find the position to insert the new node by:

- checking whether the node's data is less than or greater than the root node's data
- moving to the left node if the data is less than the root node's data
- moving to the right node if the data is greater than or equal to the root node's data
- repeating until the final position of the node is found and stores the node in that position.

Write program code for `Insert()`

Save your program.

Copy and paste the program code into part **3(c)(iii)** in the evidence document.

[6]

(d) The recursive procedure `OutputInOrder()` outputs the data stored in the binary tree in ascending numerical order.

The procedure takes a `Node` object as a parameter and then:

- checks if the left node is null. If there is a left node, the function calls itself with the left node
- outputs the data of the current node
- checks if the right node is null. If there is a right node, the function calls itself with the right node.

Write program code for `OutputInOrder()`

Save your program.

Copy and paste the program code into part **3(d)** in the evidence document.

[5]

(e) (i) Write program code to extend the main program to:

- create a new object of type `Tree` with the node containing the data 10 as the first node
- insert the nodes with the values 20, 5, 15 and 7 into the tree in the order given
- call `OutputInOrder()` with the tree's root node as a parameter.

Save your program.

Copy and paste the program code into part **3(e)(i)** in the evidence document.

[3]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot into part **3(e)(ii)** in the evidence document.

[1]