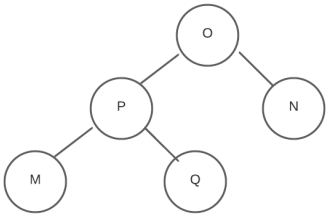
These are sample solutions. Some questions can have alternative solutions.

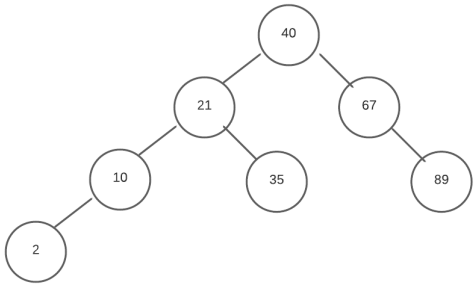
1. The post-order of a binary tree is : M, Q, P, N, O. What will be the tree? **(2 marks)**

   Answer:



2. Suppose, we are constructing a binary search tree by inserting values in this sequence: 40, 21, 35, 67, 89, 10, 2. What is the number of leaf nodes in this tree? **(2 marks)**

   Answer: (You do not need to draw the tree in this solution. This is for reference only.)
   Number of leaf nodes: 3



3. What will be the output of this code? Explain with a function call stack. **(2 marks)**

```
1  def rec(n):
2    if n!=9:
3      rec(n-1)
4      print(n)
5
6  rec(13)
```

Answer:

10
11
12
13

rec (9)
rec (10) line 4
rec (11) line 4
rec (12) line 4
rec (13) line 4

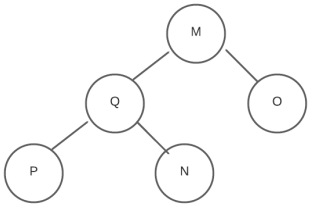
4. Write a function ‘addMinMax(self)’ that returns the sum of the minimum and maximum node (data) in a binary search tree. **(4 marks)**

   Answer:

```
def addMinMax(self):
    min = self.root
    while min.left_child:
      min = min.left_child
    max = self.root
    while max.right_child:
      max = max.right_child
    return min.data+max.data
```

1. The pre-order of a binary tree is : M, Q, P, N, O. What will be the tree? **(2 marks)**

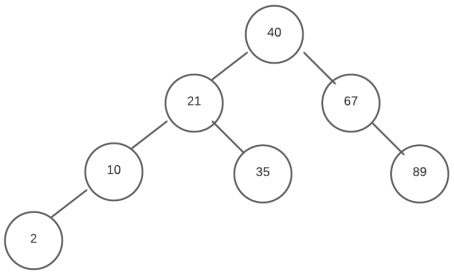
   Answer: (You do not need to draw the tree in this solution. This is for reference only.)



2. Suppose, we are constructing a binary search tree by inserting values in this sequence: 40, 21, 35, 67, 89, 10, 2. What is the height of the tree? **(2 marks**)

   Answer:

   Height: Number of levels = 4



3. What will be the output of this code? Explain with a function call stack. **(2 marks)**

```
1  def rec(n):
2      if n!=9:
3          rec(n+1)
4          print(n)
5
6  rec(5)
```

Answer:
8
7
6
5

rec (9)
rec (8) line 4
rec (7) line 4
rec (6) line 4
rec (5) line 4

4. Write a function ‘get_node_with_parent(self, search_data) that returns the parent node of the given search_data of a binary search tree. **(4 marks)**

   Answer:

```
def get_node_with_parent(self, search_data):
    parent = None
    current = self.root
    if current is None:
      return parent
    while True:
      if current.data == search_data:
        return parent
      elif current.data > search_data:
        parent = current
        current = current.left_child
      else:
        parent = current
        current = current.right_child
    return parent
```