1. node(node(node(leaf,3,leaf),6,node(leaf,8,leaf)),7,node(leaf,9,leaf))

2. node 8 is placed in the left sub-tree of node 7.
   Redraw the BST yourself. Solution is not unique.

3.

```
Algorithm: max(BST)
Requires: a non-empty binary search tree
Returns: maximum value in the BST
--------------------------------------------
if isLeaf(right(BST))
    return root(BST)
else
    return max(right(BST))
endif
```

   Time complexity is $O(h)$. (Linearly related to the height)

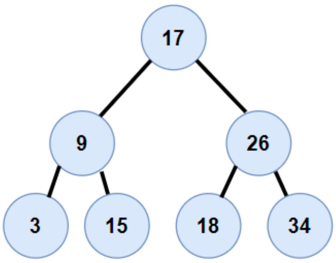
4.

```
Algorithm: insert(x,BST)
Requires: a number x and a BST
Returns: a BST after inserting x
--------------------------------------------
if isLeaf(BST)
    return node(leaf,x,leaf)
elseif x>root(BST)
    return node(left(BST),root(BST),insert(x,right(BST)))
else // x<root(BST)
    return node(insert(x,left(BST)),root(BST),right(BST))
endif
```

5. i. [7, 6, 9, 3, 8, 1, 2]
   ii. [7, 6, 3, 8, 1, 2, 9]
   iii. [3, 6, 1, 8, 2, 7, 9]
   iv. [3, 1, 2, 8, 6, 9, 7]

6. i. [15, 9, 26, 3, 18, 34, 17]
   ii. [15, 9, 3, 26, 18, 17, 34]
   iii. [3, 9, 15, 17, 18, 26, 34]
   iv. [3, 9, 17, 18, 34, 26, 15]

7. First, obtain a sorted list using inorder traversal scheme: [3, 9, 15, 17, 18, 26, 34].
(or using other traversal scheme for a unsorted list, then sort it; no need to include the sorting process).
Next, show a similar process to the example in Seminar 8 slides Page 9.
Final BST with minimum depth is:



8. Refer to Seminar 8 slides (Page 8). Do include details for two sub-algorithms maxBT and minBT.

9. Notice the way for making comparisons there: compare to current root node, then left (recursion), finally right (recursion). Therefore, we were using the NLR scheme (preorder traversal scheme) there.

10.

```
Algorithm: inorder(BST)
Requires: a binary search tree
Returns: a list for inorder traversal scheme

1:  if isLeaf(T)
2:      return nil
3:  else
4:      let L1 = cons(root(BST),nil)
5:      let L2 = inorder(left(BST))
6:      let L3 = inorder(right(BST))
7:      return concat(L2, concat(L1,L3)) // LNR order
8:  endif
```

Note: line 7 can be replaced by calling the merge() sub-algorithm instead

```
7:  return merge(L2,merge(L1,L3))
```

In which the order of L1, L2 and L3 does no matter much, because each of them is sorted, and all will be merged into a longer sorted list.