Solution1: Reverse the Top k Elements of a Stack Using a Queue

```
reverse_top_k(stk, k):
  queue = empty queue

      //Step 1: Dequeue top k elements from the stack into the queue
  for i = 1 to k:
      queue.enqueue(stk.pop())

      //Step 2: Enqueue the queue elements back onto the stack
  while not queue.is_empty():
      stk.push(queue.dequeue())
```

SOLUTION 2: Reverse Elements Between Indices i and j in a Stack Using a Queue

```
reverse_between_indices(stk, i, j):
      queue = empty queue
  //Step 1: Pop the top j elements into the queue
  for x = 1 to j:
      queue.enqueue(stk.pop())

  //Step 2: Call reverse_top_k to reverse top (j - i) elements in the queue
  reverse_top_k(queue, j - i)

  //Step 3: Enqueue the queue elements back onto the stack
  while not queue.is_empty():
    stk.push(queue.dequeue())
```

SOLUTION 3: Swap Elements at Indices i and j in a Stack Using a Queue

```
swap_indices(stk, i, j):
        queue = empty queue

  //Step 1: Reverse elements between i and j to bring both into queue
  reverse_between_indices(stk, i, j)

  //Step 2: Swap the first and last elements in the queue
  first = queue.dequeue()  //Element at index i
  last = queue.dequeue()   //Element at index j
  queue.enqueue(last)
  queue.enqueue(first)

  //Step 3: Reverse the elements back to restore order except swapped
  reverse_between_indices(stk, i, j)
```

SOLUTION 4: Post-Order Traversal for an M-Ary Tree Stored in FirstChild-NextSibling Form

```
post_order_traversal(node):
  if node is None:
     return

  //Traverse all children in FirstChild-NextSibling order
  child = node.firstChild
  while child is not None:
        post_order_traversal(child)
        child = child.nextSibling
```

```
//Visit the current node
visit(node)
```

SOLUTION 5: Modifications to a Binary Search Tree (BST)

(a) Additional Data Members to Store

To ensure that the BST can return the minimum value in constant time O(1), we need to store an additional pointer/reference or data member in each node (or in the root) to track the node that contains the minimum value in the subtree.

Reasoning: In a BST, the minimum value is always found in the leftmost node of the tree (the left child of the root or left descendants). Keeping track of this node during insertion allows us to quickly return the minimum value in O(1) time.

(b) Modifying BST Insertion to Update Minimum Value

Modified Pseudocode:

```
Input: ins: new data value to insert
Input: node: current BST node (originally root)
Input: BST.Insert
Input: min (reference to minimum node)

 if ins < node.value then
            if node.left == nil then
                        node.left = Node(ins)
                        node.left.parent = node
                        if ins < min.value then
                                    min = node.left   // Update min if new value is smaller
          else
              BST.Insert(ins, node.left, min)
        end
else
            if node.right == nil then
```

```
            node.right = Node(ins)
            node.right.parent = node
        else
            BST.Insert(ins, node.right, min)
        end
end
```

(c) Modified Deletion Pseudocode

Input: victim: BST node to be deleted

Input: BST.Delete

Input: min (reference to minimum node)

1. Let children = number of non-nil children of victim
2. if children = 0 then
   a. if victim.parent != nil then
      i. Set victim.parent's matching child pointer to nil
   b. if victim == min:
      i. Update min (find new min by traversing leftmost path from root)
   c. delete victim
3. else if children = 1 then
   a. Let child = victim's only child
   b. if victim.parent != nil then
      i. Set victim.parent's matching child pointer to child
   c. child.parent = victim.parent
   d. if victim == min:
      i. Update min (find new min by traversing leftmost path from root)
   e. delete victim
4. else

a. lhsMax = victim.left

b. while lhsMax.right != nil do

i. lhsMax = lhsMax.right

c. Swap victim.data and lhsMax.data

d. BST.Delete(lhsMax)

5. if victim == min:

a. Update min (find new min by traversing leftmost path from root)