Task 3: Queue Sorting with Limited Space You have a queue of integers that you need to sort. You can only use additional space equivalent to one stack. Describe the steps you would take to sort the elements in the queue.

1. **Approach**
   - The idea is to repeatedly extract the smallest element from the queue, move it to the stack, and then back to the queue in sorted order. We'll use the stack to temporarily hold elements while we find the minimum element from the remaining part of the queue.
2. **Steps**
   1. **Initialize:**
      - Start with an empty stack.
      - Determine the number of elements in the queue (let's call this n).
   2. **Outer Loop:**
      - Repeat the following process n times (once for each element that needs to be sorted).
   3. **Inner Loop to Find Minimum:**
      - Initialize a variable min to a very large value (e.g., Integer.MAX_VALUE).
      - Move all elements from the queue to the stack, keeping track of the minimum element and its count.
      - Push each element onto the stack unless it is the minimum element.
      - If an element is the minimum, skip pushing it onto the stack (thus removing it from the queue temporarily).
   4. **Repopulate Queue:**
      - Once the minimum element is identified, push all elements from the stack back into the queue.
   5. **Reinsert Minimum Element:**
      - Insert the identified minimum element back into the queue. This ensures that it moves towards the front of the queue with each iteration.
   6. **Repeat:**
      - Continue this process until all elements have been sorted.

Pseudo Code:

```
function sortQueueWithStack(queue):
   stack = new Stack()
   n = queue.size()
   for i = 0 to n - 1:
      min = Integer.MAX_VALUE
      minCount = 0
```

```
    // Move elements from queue to stack, find the min element
    for j = 0 to queue.size() - 1:
        element = queue.dequeue()
        if element < min:
            min = element
            minCount = 1
        else if element == min:
            minCount += 1
        stack.push(element)

    // Move elements back from stack to queue, skipping min element
    while not stack.isEmpty():
        element = stack.pop()
        if element != min:
            queue.enqueue(element)
        else:
            minCount -= 1
            if minCount > 0:
                queue.enqueue(element)

    // Enqueue the min element to its sorted position
    queue.enqueue(min)

return queue
```

Java Implementation

Here's a Java implementation:

```
/*
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;
public class QueueSorter {
    public static void sortQueueWithStack(Queue<Integer> queue) {
        Stack<Integer> stack = new Stack<>();
        int n = queue.size();

        for (int i = 0; i < n; i++) {
            int min = Integer.MAX_VALUE;
            int minCount = 0;
            int currentSize = queue.size();

            // Find the minimum element
            for (int j = 0; j < currentSize; j++) {
                int element = queue.poll();
                if (element < min) {
                    min = element;
                    minCount = 1;
                } else if (element == min) {
                    minCount++;
                }
                stack.push(element);
            }
            // Move elements back from stack to queue, skipping the min element
            while (!stack.isEmpty()) {
                int element = stack.pop();
                if (element != min) {
                    queue.add(element);
                } else {
                    minCount--;
                    if (minCount > 0) {
                        queue.add(element);
                    }}
            }
            // Enqueue the min element to its sorted position
            queue.add(min);
        }
    }
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        queue.add(3);
        queue.add(1);
        queue.add(4);
        queue.add(1);
        queue.add(5);
        queue.add(9);
        queue.add(2);
        System.out.println("Original Queue: " + queue);
        sortQueueWithStack(queue);
        System.out.println("Sorted Queue: " + queue);
    }
}  */
```

Explanation

Initialization:

- A stack is used to temporarily hold the queue's elements.
- The number of iterations is equal to the number of elements in the queue.

Finding the Minimum Element:

- The inner loop finds the minimum element in the current queue by moving elements to the stack.
- The minimum element is identified and not pushed to the stack on the first pass.

Repopulating the Queue:

- Elements are moved back from the stack to the queue, except for the minimum element (which is re-added later).

Placing the Minimum Element:

- The identified minimum element is then added back to the queue.
- This process is repeated, which ensures that the smallest elements are moved to their correct positions in the queue with each pass.