DAY 12:

Task 3:

Q) 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.

EXPLANATION:

Overview

The objective is to sort a queue of integers in ascending order using an auxiliary stack. The process involves repeatedly finding the minimum element and placing it in its correct position within the queue.

Steps Explained

1. Initialization:

- Create an empty stack that will temporarily hold elements from the queue during the sorting process.

- Determine the size of the queue, as this will help in iterating through the elements.

2. Outer Loop (Main Sorting Loop):

- Run a loop for n iterations, where n is the size of the queue. Each iteration of this loop will place the next smallest element in its correct position in the queue.

3. Finding the Minimum Element:

- For each iteration, initialize a variable min to the maximum possible integer value (Integer.MAX_VALUE) and a counter minCount to keep track of occurrences of the minimum value.

- Calculate the current size of the queue.

- Run a nested loop to traverse all elements in the queue to find the minimum element.

  - Poll each element from the queue and compare it with the current min.

  - If the element is smaller, update min and reset minCount to 1.

  - If the element is equal to min, increment minCount.

  - Push each element onto the stack, temporarily storing them.

4. Rebuilding the Queue:

- Once the minimum element is identified, the elements in the stack are moved back to the queue, except the minimum element (which should now be skipped).

- As each element is popped from the stack, it is either added back to the queue or skipped if it is the identified minimum value. If there are multiple occurrences of the minimum, decrement minCount each time the minimum is skipped until all its occurrences are correctly handled.

5. Placing the Minimum Element:

- After the stack is emptied and all elements are moved back to the queue (except the minimum elements handled earlier), add the identified minimum element back to the queue. This places it in its correct sorted position for the current iteration.

6. Repeat:

- Repeat the process for n iterations, each time correctly positioning the next smallest element until the entire queue is sorted.

Example Workflow

Consider the queue with elements [5, 8, 12, 16, 21, 25, 24]:

1 First Iteration:

- Find the smallest element (5).

- Rebuild the queue without 5 and then place 5 at the end.

- Queue becomes: [8, 12, 16, 21, 25, 24, 5]

2. Second Itetation:

- Find the smallest element (8) from the remaining elements.

- Rebuild the queue without 8 and then place 8 at the end.

- Queue becomes: [12, 16, 21, 25, 24, 5, 8]

3. Subsequent Iteration:

- Continue the process, each time finding the next smallest element and positioning it correctly until the queue is sorted.

Result

After the completion of all iterations, the queue will be sorted in ascending order.

Conclusion

The provided algorithm efficiently sorts a queue using an auxiliary stack by repeatedly identifying the minimum element and repositioning it within the queue. This approach ensures that the queue is sorted in O(n^2) time complexity due to the nested loops. While this is not the most efficient sorting algorithm, it demonstrates the use of auxiliary data structures (stack) to manipulate and sort the primary data structure (queue).