I suggested the following extension to the Gibbons D algorithm in order to support the *deq(⟘)* operation. This will require us to add a step to the algorithm (denoted as step 4’) as well as make a slight change to step 4 to avoid early termination. The changes are presented as follows:

Given an input of *N* operations of the type *enq/deq*, *K < N* of which are of the form *deq(⟘)* adjust step 4 of Gibbons’ algorithm to ignore all *deq(⟘)* operations. As these operations are not matched this will avoid an early termination on step 4. Note that all standard limitations still apply on the values. Specifically, that they are still distinct integer pairs. In addition, step 4’ is added between steps 4 and 5 in which we add the pairs to according to the following rule: .   
In principle, we are making sure that *deq(⟘)* operations never appear between the *enq* and *deq* operations of any single value.

Theorem:

The revised algorithm’s correctness is still sound. Specifically, if the algorithm approves a trace it means that all *deq(⟘)* operations in it behave correctly.   
Proof for *deq(⟘)*: for a *deq(⟘)* operation to be invalid, it means that it occurs when a queue is not empty. Therefore, the following statement should be correct: . However, since are an event pair, this is in direct contradiction with the pairs added in step 4’. Moreover, the transitive pairs added in step 5 make sure that this applies globally and iteratively throughout the algorithm’s entire run.  
The full algorithm’s correctness still remains sound as we only added pairs and never removed pairs from the original algorithm’s run.

Algorithm D’

*Given*: A distinct-value trace for a (possible buggy) FIFO queue. The trace contains events and may include a subset of arbitrary size of events.  
*Question*: Does there exist a queue sort of the trace?

1. Match up event pairs. If there exists an event that is not in an event pair and is not , return NO.
2. Let .
3. For each event pair , add .
4. Let and be a pair of event pairs. If , then add . If , then add .
5. Let be an event pair and an instance of such operation. If , then add . If , then add .
6. Add all transitive pairs to .
7. Repeat steps 4-6 until no new pair can be added to .
8. If the resultant relation has a cycle, return NO.
9. Return a topological sort using the following greedy schedule. At each step of the schedule, an event in is eligible if it is unscheduled and all events in such that have been scheduled. Starting with an empty schedule and an empty queue, repeat until all events are scheduled: Schedule an eligible (if any). If there are no eligible events, schedule an eligible enqueue

(if any), adding its value to the tail of the queue. If there are no eligible enqueue

events, schedule the eligible dequeue for the value that is presently at the head

of the queue.

Theorem 3.6’. *Let be a distinct-values trace with n events, k < n of which are of type . Algorithm D’ runs in time and returns a queue sort of if and only if one exists.*

*Proof.* All constraints added in steps 3-6 must be satisfied by all queue sorts of. This if the resultant relation has a cycle, then there is no queue sort.

Assume that we have partially constructed the topological sort and there are no eligible or enqueue events and all eligible dequeue events violate the queue sort property. Since is acyclic, there must be an eligible dequeue event, . Since by step 3, where is an event pair, and is eligible, has been scheduled. Thus, the queue is not empty. Let be the value at the head of the queue and let be an event pair. Since no enqueue events are eligible, there is an eligible dequeue event , such that , and is in the queue. By step 4, we have that , where is an event pair. Thus precedes in the FIFO queue, a contradiction. Therefore, step 9 succeeds in constructing a queue sort.

Algorithm D’ can be implemented in time using known techniques for maintaining transitive closure under edge insertions [It], [LPvL].