Algorithm 21 relabel-to-front(G, s, t)
1: initialize preflow
2: initialize node list L containing V \ \{s, t\} in any order
3: foreach u ∈ V \ \{s, t\} do
4: \hspace{1cm} u.current-neighbour ← u.neighbour-list-head
5: \hspace{1cm} u ← L.head
6: while u ≠ null do
7: \hspace{1cm} old-height ← ℓ(u)
8: \hspace{1cm} discharge(u)
9: \hspace{1cm} if ℓ(u) > old-height then // relabel happened
10: \hspace{1cm} move u to the front of L
11: \hspace{1cm} u ← u.next

Lemma 1 (Invariant)
In Line 6 of the relabel-to-front algorithm the following invariant holds.
1. The sequence L is topologically sorted w.r.t. the set of admissible edges; this means for an admissible edge (x, y) the node x appears before y in sequence L.
2. No node before u in the list L is active.

Proof:
▶ Initialization:
1. In the beginning s has label n ≥ 2, and all other nodes have label 0. Hence, no edge is admissible, which means that any ordering L is permitted.
2. We start with u being the head of the list; hence no node before u can be active
▶ Maintenance:
1. Pushes do not create any new admissible edges. Therefore, if discharge() does not relabel u, L is still topologically sorted.
2. After relabeling, u cannot have admissible incoming edges as such an edge (x, u) would have had a difference ℓ(x) − ℓ(u) ≥ 2 before the re-labeling (such edges do not exist in the residual graph).
Hence, moving u to the front does not violate the sorting property for any edge; however it fixes this property for all admissible edges leaving u that were generated by the relabeling.

Note that the invariant means that for u = null we have a preflow with a valid labelling that does not have active nodes. This means we have a maximum flow.
13.2 Relabel to Front

Lemma 2
There are at most $O(n^3)$ calls to discharge($u$).

Every discharge operation without a relabel advances $u$ (the current node within list $L$). Hence, if we have $n$ discharge operations without a relabel we have $u = \text{null}$ and the algorithm terminates.

Therefore, the number of calls to discharge is at most $n(\#\text{relabels} + 1) = O(n^3)$.

13.2 Relabel to Front

Lemma 3
The cost for all relabel-operations is only $O(n^2)$.

A relabel-operation at a node is constant time (increasing the label and resetting $u$ current-neighbour). In total we have $O(n^2)$ relabel-operations.

13.2 Relabel to Front

Note that by definition a saturating push operation $(\min\{c_f(e), f(u)\} = c_f(e))$ can at the same time be a non-saturating push operation $(\min\{c_f(e), f(u)\} = f(u))$.

Lemma 4
The cost for all saturating push-operations that are not also non-saturating push-operations is only $O(mn)$.

Note that such a push-operation leaves the node $u$ active but makes the edge $e$ disappear from the residual graph. Therefore the push-operation is immediately followed by an increase of the pointer $u$ current-neighbour.

This pointer can traverse the neighbour-list at most $O(n)$ times (upper bound on number of relabels) and the neighbour-list has only $degree(u) + 1$ many entries (+1 for null-entry).

13.2 Relabel to Front

Lemma 5
The cost for all non-saturating push-operations is only $O(n^3)$.

A non-saturating push-operation takes constant time and ends the current call to discharge(). Hence, there are only $O(n^3)$ such operations.

Theorem 6
The push-relabel algorithm with the rule relabel-to-front takes time $O(n^3)$. 