Example - Reverse a Singly Linked List

Make a new list, use basic operations: Copy \( L \) to a second list \( R \) one item at a time. Because items are removed from the front of \( L \) and placed at the front of \( R \), we are essentially using \( R \) as a stack where inserting at the front is pushing onto \( R \).

\[
R := \text{CREATE}
\]

\[
\text{while (not \ EMPTY(L)) do begin}
\]
\[
d = \text{data stored in the first vertex of } L
\]
\[
\text{Delete the first vertex of } L.
\]
\[
\text{Insert a new vertex at the front of } R \text{ with data } d.
\]
\[
\text{end}
\]

\[
\text{DESTROY}(L)
\]

\[
L := R
\]

Complexity: \( O(n) \) time and \( O(1) \) space in addition to the space consumed by the list vertices.
Example - Reverse a Singly Linked List In-Place

Use “low-level” operations: Reverse each next pointer and then finish by exchanging the first and last pointers of the header.

previous := nil
current := L->first
while (current ≠ nil) do begin
    temp := current->next
    current->next := previous
    previous := current
    current := temp
end
L->last := L->first
L->first := previous

Complexity: O(n) time and O(1) space in addition to the space consumed by the list vertices.