7.1. How to reverse a singly linked list in $O(n)$ time with $O(1)$ additional memory?

7.2. To save memory in doubly linked lists, instead of two pointers (to the next and previous elements), you can store only their bitwise XOR. (for example, if `prev[i] = 5`, `next[i] = 3`, then we store `prevnext[i] = 6` instead). How to iterate over such a list?

7.3. Given a set of n nodes, each has a pointer to some another node from the set. Check if these nodes form a circular list. (Time $O(n)$, memory $O(1)$).

7.4. Merge two sorted singly linked lists into one in $O(n)$ time with $O(1)$ additional memory.

7.5. Sort the singly linked list in $O(n \log n)$ time with $O(1)$ additional memory.

7.6. There is a table $n \times n$. M operations of the following type are carried out with it: cut out a rectangular piece, rotate it 180° and paste in the same place. Print the state of the table after all operations. Time $O(nm)$.

7.7. Figure out how to store the linked list so that it can be reversed in $O(1)$ time (with all other operations preserved).

7.8. Given a set of n nodes, each with a link to the next and the previous one. Check if these elements really form several linked lists and, if so, concatenate these lists into one (in any order). (Time $O(n)$, memory $O(1)$).