1. A self-organizing data structure is reorganized during execution in response to a sequence of operations, with the goal of achieving good performance on the actual, initially unknown, sequence. Often, the majority of operations concern a small number of elements, and so optimizing the access time to these elements can improve performance overall. In this problem, we will analyze a simple self-organizing linked list. In this linked list, whenever we access a list node, we move that node to the head of the list (pushing the rest of the elements back one position). We’ll say that it costs one operation (\$1, if you like) to access a node of the list and check its value, so that walking a list to access the \( k \)th element of that list costs \( k \) operations.

(a) Consider some arbitrary list containing \( n \) distinct elements, and suppose we have used the self-organizing algorithm when accessing \( k \) out of the \( n \) elements. For each of the remaining \( n - k \) elements, exactly how many more operations would it take to access that element in the self-organizing list than in the original list?

(b) Now, show that for our arbitrary list containing \( n \) distinct elements, and any sequence of \( T \) accesses to the list elements, the self-organizing algorithm uses at most twice as many operations as if we had kept the list order static.

(c) Finally, show that for every possible static ordering of the \( n \) distinct elements and sequence of \( T \) accesses, if these accesses cost \( C \) operations in total, then the self-organizing algorithm uses at most \( 2 + \frac{n^2}{T} \) operations in total, for a competitive ratio of at most \( 2 + \frac{n^2}{T} \rightarrow 2 \) against all static lists.

2. Kleinberg & Tardos – Chapter 11, exercise 1

3. Kleinberg & Tardos – Chapter 11, exercise 4

4. Kleinberg & Tardos – Chapter 13, exercise 6