Algorithms Group **Summer 2020**
Department of Computer Science - IBR
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Phillip Keldenich

Online Algorithms
Exercise 3
June 3, 2020

Hand in your solutions as PDF file until June 17, 2020, 11:30 AM via e-mail to `v.sack@tu-bs.de`, with CC to `keldenich@ibr.cs.tu-bs.de`. If you cannot turn your solution into a PDF file (for example by writing it in LaTeX or Word), you can also submit photographs or scans. In that case, be careful to keep the file size acceptable (about 3 MB per page) by using appropriate compression and resolution; however, make sure that your solutions are still readable.

In this homework assignment, we consider the List Update Problem. Suppose that we are storing a set $S = \{s_1, \ldots, s_n\}$ of values as unsorted singly-linked list. Our input sequence consists of queries $s_i \in S$ that ask us to find a certain element in our list. In order to find an element, we iterate through the list starting from the front and return the element once we find it. Because iterating through our list takes time, we incur a cost of 1 for each element that we touch during the search. For example, searching for 2 in the list $[1, 2, 3]$ costs two units, and searching for 2 in $[2, 1, 3]$ costs just one unit.
After each query for an item s_i, we are allowed to move the item s_i to any point closer to the front of the list; this exchange does not cost us anything. For example, if we search for 2 in the list $[1, 4, 2, 3]$, we can change the list to $[2, 1, 4, 3]$, $[1, 2, 4, 3]$ or $[1, 4, 2, 3]$ after finding 2 for a total cost of three units. Intuitively speaking, we want our algorithm to maintain its list such that frequently requested elements are closer to the front than less frequently requested elements.

Exercise 1 (Bad Algorithms):
In this task, we consider natural (but non-competitive) algorithms for the List Update Problem.

a) After each request, the algorithm Transpose swaps the requested element with the preceeding element in the list. For example, after searching 2 in the list $[1, 4, 2, 3]$, the list becomes $[1, 2, 4, 3]$; searching for 2 again results in the list $[2, 1, 4, 3]$. Prove that Transpose is not c-competitive for any constant c.

b) The algorithm FrequencyCount maintains a frequency count for each element that is incremented each time the element is requested. After each request, it moves the requested item as far as possible to the front such that the list stays in nonincreasing order of frequency count. Prove that FrequencyCount is not c-competitive for any constant c.

(5+10 pts.)

Exercise 2 (Move To Front):
In this exercise, we consider the MoveToFront algorithm for the List Update Problem. After each request s_i, the algorithm moves the requested item s_i to the front of the list. For example, after searching for 2 in $[1, 4, 2, 3]$, the list becomes $[2, 1, 4, 3]$.

a) Prove that there is no constant $c < 2$ such that Move To Front is c-competitive.

b) Prove that Move To Front is 2-competitive.

Hint: Use the number of inversions in Move To Front's list w.r.t. OPT's list after request i as a potential function $\phi(i)$. An inversion is a pair x, y of elements such that x comes before y in Move To Front's list, but after y in OPT's list.

In order to prove $c_{\text{MTF}}(i) + \phi(i) - \phi(i-1) \leq 2c_{\text{OPT}}(i)$ for a request s_i, consider the number of items k that come before s_i in both OPT's and Move To Front's list, the number of items m that come before s_i in OPT's list but after s_i in Move To Front's list, and the number of items ℓ that come before s_i in Move To Front's list but after s_i in OPT's list.

(10+15 pts.)