As mentioned in multiple occasions in the class, there are two major approaches for analysis of algorithms, namely average- or expected- case analysis and worst-case analysis. We saw examples of both in the context of *time complexity* of algorithms such as Quick-Sort.

When discussing online algorithms such as self-adjusting lists, we are mostly concerned with *quality of solutions* (rather than time complexity). We often compare a given algorithm with an optimal algorithm OPT. There are two ways to make that comparison: worst-case analysis and average-case analysis. In the worst-case analysis, an *adversary* generates input sequences with the goal of maximizing the ratio between the cost of the algorithm and that of the optimal algorithm. In the average- or expected- case analysis, there is no adversary. The input is generated randomly from a given distribution. In what follows, we discuss these approaches for the Move-To-Front (MTF) algorithm.

1 Average-case analysis of MTF

Theorem 1 *$C_{MTF} \leq 2 \cdot C_{OPT}$, where C_{OPT} is the expected cost under optimal ordering.*

Let's assume that our keys are $k_1, .., k_n$ with probabilities $p_1, .., p_n$ such that $p_1 \geq p_2 \geq ... \geq p_n$. Then the optimal ordering is $k_1, .., k_n$, and

$$C_{OPT} = \sum_{j=1}^{n} j p_j$$

Let's assume we are in a steady state (and every key has been searched for at least once), then:

$$\begin{aligned} C_{MTF} &= \sum_{j=1}^{n} p_j (\text{cost of finding } k_j) \\ &= \sum_{j=1}^{n} p_j (1 + \text{number of keys before } k_j) \end{aligned}$$

What is the number of keys before k_j? For a given $i \neq j$, consider the last time the following event happened: we searched for key i OR key j

$$\text{Prob } i \text{ before } j = \frac{p_i}{p_i + p_j}$$

Denominator is probability of searching for key i OR j. Expected number of keys before k_j is $\sum_{i \neq j} \frac{p_i}{p_i + p_j}$. Therefore,

$$\begin{aligned}
C_{MTF} &= \sum_{j=1}^{n} p_j (1 + \sum_{i \neq j} \frac{p_i}{p_i + p_j}) \\
&= \sum_{j=1}^{n} p_j + \sum_{j=1}^{n} (\sum_{i \neq j} \frac{p_i p_j}{p_i + p_j}) \\
&= 1 + \sum_{j=1}^{n} 2 (\sum_{i<j} \frac{p_i p_j}{p_i + p_j}) \\
&= 1 + 2 \sum_{j=1}^{n} p_j (\sum_{i<j} \frac{p_i}{p_i + p_j}) \\
&\leq 1 + 2 \sum_{j=1}^{n} p_j (\sum_{i<j} 1) \\
&= 1 + 2 \sum_{j=1}^{n} p_j (j-1) \\
&= 1 + 2C_{OPT} + 2 \sum_{j=1}^{n} (-p_j) \\
&= 2C_{OPT} - 1.
\end{aligned}$$

2 Worst-case analysis of MTF

In the worst-case analysis, an adversary generates the input sequence. Consider a list of n items initially ordered as $[a_1, \ldots, a_n]$. Consider a *phase* of requests formed by adversary by repeatedly asking the last item in the list maintained by MTF. Such phase would like like $\langle a_n, a_{n-1}, \ldots, a_1 \rangle$. Note that after asking for the first a_n, it is moved to front; now a_{n-1} is the last item in the list (hence, second request), and this pattern is repeated for other items. So, MTF incurs a cost of $n \times n$ for the phase. A static algorithm that does not move any item incurs a cost of $n + (n-1) + \ldots + 1$ for the phase. Hence, the cost of the optimal algorithms is at most $n(n+1)/2$. The ratio for the cost of MTF and that of OPT for this adversarial sequence would be at least $\frac{n^2}{n(n+1)/2} = 2 - 2/(n+1)$ which approaches to 2 for long lists. Note that the state of the list is similar to the beginning of phase for MTF and the static algorithm. So, we can repeated the above argument for multiple

phases to get sequences of arbitrary length.

- There is a classic result [Sleator/Tarjan, 1984] that shows that the above lower bound is tight, i.e., the ratio between the cost of MTF and OPT is never more than 2. This result implies the result in Section 1, i.e., we can say since MTF is at most two times worst than OPT when the adversary generates input, it is for sure at most two times worst than OPT for any other input and particularly when the input is generated randomly.
- Think of similar adversarial arguments for Transpose and MTF2 (where an accessed item is moved half way to front).