CS1570 Lecture Notes

Claire Mathieu

Fall 2026

1 October 13 and 15: Amortized analysis and the list update problem

In this lecture we study the *list update problem*. It is the standard illustration of the *potential function technique* for amortized analysis.

Here is the list update problem.

Input. A list of n items, and a sequence of item requests $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$. The problem is *online*: the requests arrive one at a time.

Serving a request. A request is served by *accessing* the item, and possibly *rearranging* the order of the list.

- The cost of accessing the item in position i is i.
- The cost of swapping two adjacent items is 1.

Online decisions. For each request, the algorithm must decide what to do to serve it *without knowledge of the future requests*.

Goal. Design an *adaptive, self-organizing heuristic*: a simple rule that reorganizes the list so that the items that are requested often end up near the front.

We compare such a heuristic with OPT, the best *offline* algorithm, i.e. the algorithm that knows the whole request sequence in advance and serves it at minimum total cost (starting from the same initial list).

Note that the exact computation of OPT is NP-hard, so we do not know how it works exactly, and yet, we are able to compare MTF to OPT.

Here are some possible heuristics.

Three natural rules, when an item x is requested:

Frequency count (FC). Keep a counter for each item, increment the counter of the requested item, and keep the list sorted by decreasing counter. (An item only moves ahead of items with a *strictly* smaller counter.)

Move-to-front (MTF). Move x to the front of the list ("most recently used first").

Transposition. Swap x with the item immediately in front of it.

It turns out that transposition and FC both behave badly on some request sequences.

Fact 1. *Transposition is not good.*

Proof. Let the initial list be $A_1, A_2, \ldots, A_\ell, B, C$ and let the requests be $C, B, C, B, C, B, \ldots$. Transposition swaps the requested item with its predecessor, so the list alternates between

$$A_1 A_2 \cdots A_\ell \, B \, C \quad \longleftrightarrow \quad A_1 A_2 \cdots A_\ell \, C \, B,$$

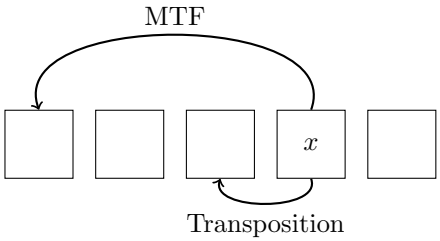


Figure 1: The two local rules, applied to a request for the item x in position 4.

and the requested item is always in the last position. Hence every request costs more than ℓ, forever.

MTF, in contrast, pays only about 3 per request on this sequence: after the first two requests B and C occupy the first two positions, and serving either of them costs 2 for the access plus 1 for the swap. □

Fact 2. *FC is not good.*

Proof. Let the list be $A_1, A_2, \ldots, A_\ell$ and let the requests be

$$A_1^\ell\, A_2^\ell \cdots A_\ell^\ell \qquad \text{(each item requested } \ell \text{ times in a row; } \ell^2 \text{ requests).}$$

When A_i is being requested, its counter is at most ℓ, while all of $A_1, \ldots, A_{i-1}$ already have counter ℓ. So FC leaves the list $A_1, A_2, \ldots, A_\ell$ unchanged. A typical request goes about halfway down the list, so

$$\text{cost(FC)} \approx \ell^2 \cdot \frac{\ell}{2} = \Theta(\ell^3).$$

(Precisely, $\sum_{i=1}^{\ell} \ell \cdot i = \ell^2(\ell+1)/2$.)

MTF pays 1 for each of the ℓ^2 requests, plus for each item A_i the extra cost of its first request. At that moment the list is $A_{i-1} \cdots A_1 A_i A_{i+1} \cdots A_\ell$, so A_i is in position i, and serving it costs $2i-1$ (access i, plus $i-1$ swaps to move it to the front). Since $\sum_{i=1}^{\ell}(2i-1) = \ell^2$,

$$\text{cost(MTF)} \;\approx\; \ell^2 \cdot 1 + \sum_{i=1}^{\ell}(2i-1) \;=\; \Theta(\ell^2).$$

Since OPT can do at least as well as MTF, FC is a factor $\Omega(\ell)$ worse than OPT on this sequence. □

For MTF, serving an item that is in position k costs

$$k \text{ (access)} \;+\; (k-1) \text{ (swaps that move it to the front)} \;=\; 2k-1.$$

Here is the main theoretical result.

Theorem 1. *For any request sequence, starting from the same initial list,*

$$\text{cost(MTF)} \;\le\; 4 \cdot \text{cost(OPT)}.$$

Idea. MTF makes a *compromise*: it pays more now, by reorganizing the list, in order to pay less later. We have to show that the extra work done now really is repaid by savings later.

The potential function technique

Analysis idea. Define a *potential* Φ associated to the current time (it depends on the current state, here on the current lists of MTF and of OPT). When serving request i, let

- c_i be the *actual cost* ($2k-1$ for an item in position k),
- $\tilde{c}_i$ be the *amortized cost*, defined by

$$\tilde{c}_i \;=\; c_i + \Delta\Phi_i \;=\; c_i + \Phi_{\text{after}} - \Phi_{\text{before}}.$$

That's general. Now we apply this technique to analyze MTF for list update.

The magic potential.

$$\Phi \;=\; 2 \cdot \mathrm{dist}\big(\text{list order of MTF},\ \text{order of OPT}\big),$$

where dist is the *number of inversions*: the number of pairs of items x, y that appear in a different relative order in the two lists. In particular $\Phi \geq 0$, and $\Phi = 0$ when the two lists are equal.

The following Lemma is the main part of the proof that is specific to list update and MTF. Intuitively: when MTF, in the process of answering a request, costs a lot more than OPT, then it also reorganizes the list in such a way that the new structure is better, in a way that is quantified by the potential decrease.

For a general problem, in order to design a potential function, one needs to ask oneself: "when the algorithm is more costly than OPT, what side benefit has it accumulated ? In what has the structure improved ?"

Lemma 1. *If* OPT *pays j for a request, then $\tilde{c} \leq 4j$.*

Proof of Theorem 1, assuming Lemma 1. Let there be m requests. The total cost paid by MTF is

$$\begin{aligned}\sum_{i=1}^{m} c_i = \sum_{i=1}^{m} \tilde{c}_i - \sum_i \Delta\Phi_i \;&=\; \sum_{i=1}^{m} \tilde{c}_i \;-\; (\Phi_{\text{final}} \;-\; \Phi_{\text{initial}}) \\ &\leq \sum_{i=1}^{m} \tilde{c}_i \;\leq\; 4 \cdot \mathrm{cost}(\mathrm{OPT}).\end{aligned}$$

Here the sum of the $\Delta\Phi_i$ telescopes to $\Phi_{\text{final}} - \Phi_{\text{initial}}$: that's the whole point of using potential functions in amortized analysis to define amortized costs. Moreover $\Phi_{\text{initial}} = 0$ because both algorithms start from the same list and $\Phi_{\text{final}} \geq 0$. The last inequality follows from Lemma 1 applied to each request.

This is the typical structure of a proof for amortized analysis using potential functions. □

Proof of Lemma 1. First, we need to set up some notations.

Let x denote the item requested. Let L denote the list of MTF at that moment, and let k be the position of x in L. Then $c = 2k - 1$. Let L^* denote the list of OPT at that moment, and let k^* be its position in L^*.We may assume that OPT first accesses x (cost k^*) and then performs s adjacent swaps (cost s), so the cost paid by OPT is $j = k^* + s$.

To analyze the amortized cost, we need to compute the change of the potential Φ. Since both L and L^* change, it is pretty complicated to understand what is going on. To make it more manageable, let's do it in two successive phases. In each phase only one list has its order that changes, so it's easier to analyze.

Phase 1: MTF moves x to the front, so L changes while we leave the order L^* fixed; see Figure 2.

Phase 2: OPT *performs s swaps*, so L^* changes, but here, L stays put in whatever state it was at the end of phase 1.

We will analyze $\Delta\Phi$ in the first phase, and then $\Delta\Phi$ in the second phase, and will sum the two to have $\Delta\Phi$ over the entire request.

Focus on phase 1. We have three orders to examines: L^* (which is fixed), L before, and L after. Regarding the pairs of items that are in the same relative order in L before and after, they do not contribute to $\Delta\Phi$. So we only consider the pairs of items whose relative order has changed in L before and after. Those are exactly the pairs $\{x, y\}$ with y in front of x in L; there are $k - 1$ such items y. To see how they contribute to $\Delta\Phi$, you need to consider their relative order in L^*. Split them into two sets.

- P_a: the items y that are also in front of x in L^*. Before the move the pair $\{x, y\}$ is *not* an inversion; afterwards it is, so they each contribute $+1$ to $\Delta\Phi$. Since P_a is contained in the set of the $k^* - 1$ items in front of x in L^*, we have $|P_a| \leq k^* - 1$.
- P_b: the remaining $k - 1 - |P_a|$ items, which are behind x in L^*. The pair $\{x, y\}$ was an inversion and no longer is, so they each contribute -1 to $\Delta\Phi$.

So the number of inversions changes by

$$|P_a| - |P_b| \;=\; 2|P_a| - (k-1) \;\leq\; 2(k^* - 1) - (k - 1),$$

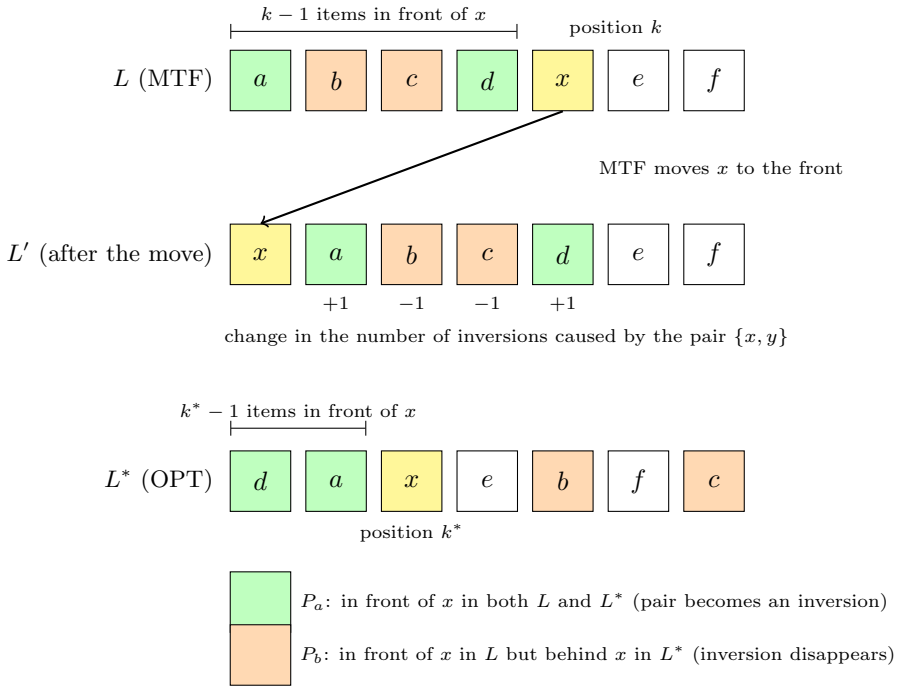


Figure 2: Phase 1 in the proof of Lemma 1: MTF moves x to the front (example with $k = 5$, $k^* = 3$, $|P_a| = |P_b| = 2$). The $k-1$ items in front of x in L are split into P_a and P_b. Since P_a is contained in the k^*-1 items in front of x in L^*, we have $|P_a| \leq k^*-1$, and the number of inversions changes by $|P_a| - |P_b|$.

and Φ changes by at most $4(k^*-1) - 2(k-1)$.

In phase 2, a swap of two adjacent items in L^* changes the relative order of exactly that pair, so the number of inversions between L^* and L changes by ± 1 and Φ changes by at most 2. In total, after s swaps in L^* Φ increases by at most $2s$.

Altogether,

$$\tilde{c} \;=\; c + \Delta\Phi \;\leq\; (2k-1) + 4(k^*-1) - 2(k-1) + 2s \;=\; 4k^* + 2s - 3 \;\leq\; 4(k^*+s) \;=\; 4j. \quad \square$$

What happened: the $2k$ from the real cost of serving the request canceled with the $-2k$ from the amortized analysis of Phase 1.

Intuitively, if $k^* << k$ then the algorithm pays a lot but this situation can only happen when the potential is high, that is, the two orderings diverged because of some priori choices that had increased the potential, and then the move-to-front heuristic uses up some of that high potential and gets to a new configuration that has much lower potential, so the $2k$ part of the real cost is cancelled by the $-2k$ part of potential change.