Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

More Investigations on the Dynamic List Update Problem

S. Zal[1] and A. Moeini*[1]

[1]School of Engineering Science, College of Engineering, University of Tehran, Tehran, Iran

ABSTRACT

There are well-established online algorithms for the static list access problem. However, the competitive ratio of dynamic list updating requires further investigation. We formalize the quantitative analysis of several dynamic algorithms: MTF, RMTF, BIT, and TIMESTAMP. For MTF, we apply the factorization lemma and prove that $2-\frac{2}{\ell+1}$ holds for general lists—a tighter bound than previous analyses. We show that RMTF_p has a lower bound of $\frac{1}{p}-\varepsilon$ for $p \in (0,1)$. For dynamic BIT, we establish a lower bound on the expected competitive ratio for update operations. We prove that TIMESTAMP is 2-competitive by analyzing insert, delete, and access costs. These results improve existing bounds and have potential applications in caching and dynamic data structures.

ARTICLE INFO

Article history:
Research paper
Received 07, November 2025
Accepted 21, December 2025
Available online 31, December 2025

Keyword: Online Algorithms, List Update Problem, MTF, RMTF, BIT, TIMESTAMP.

AMS subject Classification: 68W27, 68P05, 68W20.

1 Introduction

In a self-organizing data structure, items are reorganized after each operation to reduce the cost of future operations, thereby improving overall performance. In the list access

*Corresponding author: A. Moeini. Email: moeini@ut.ac.ir

problem, a list of ℓ records and a request sequence σ are considered. After accessing a record, the list may be reconfigured to reduce future access costs. As a self-organizing data structure, the list access problem is typically implemented using a singly linked list. Based on the types of operations and list behavior, two main models are studied in the literature. The first is the static list access model, where the number of items in the list is fixed and only access operations are permitted. The second is the dynamic list access model, where the list size varies over time, and three types of operations—insert, delete, and access—are allowed. This paper presents a brief review of classical algorithms for the static list access problem and then investigates their dynamic counterparts.
Well-known online deterministic algorithms for the list access problem include Move-to-Front (MTF), Transpose (TRANS), and Frequency Count (FC) [1]:

- **Move-to-Front (MTF)**: After accessing an item, it is moved to the front of the list, while the relative order of the remaining items is preserved.
- **Transpose (TRANS)**: After accessing an item, it is exchanged with its immediate predecessor in the list.
- **Frequency Count (FC)**: Each item maintains a counter that tracks its access frequency. The list is maintained in non-increasing order of these counters.

Online algorithms for the list access problem can be broadly classified into deterministic and randomized categories. Deterministic algorithms produce the same output and follow the same sequence of states for a given input sequence. In contrast, randomized online algorithms make random choices at certain steps during request processing, which can affect their behavior and performance. Prominent randomized algorithms for this problem include Randomized Move-to-Front (RMTF), BIT, and TIMESTAMP [1].

- **BIT**: Each item maintains a single bit that is complemented upon each request. If the bit changes to 1 after complementing, the requested item is moved to the front of the list; otherwise, its position remains unchanged.
- **TIMESTAMP**: After each request, the accessed item x is moved immediately in front of the first item y that precedes x in the list and has been requested at least once since the last request to x. If no such item y exists, or if x is requested for the first time, the position of x remains unchanged.
- **RMTF** (Randomized Move-to-Front): Upon a request for item x, it is moved to the front with probability p, where $p \in (0, 1]$. This variant is denoted as RMTF_p in the literature.

The list update problem was first studied by McCabe [2], who introduced the concept of relocatable records in serial files and proposed two fundamental algorithms: Move-to-Front (MTF) and Transpose (TRANS). Hester and Hirschberg [3] provided a comprehensive

survey of permutation algorithms that modify the order of linear search lists, with an emphasis on average-case analysis.

A seminal contribution was made by Sleator and Tarjan [4], who formally introduced competitive analysis for online deterministic list update algorithms, including MTF, TRANS, and Frequency Count (FC), using amortized analysis and potential functions. They proved that MTF is 2-competitive, while FC and TRANS are not competitive. Irani [5] proposed the first randomized online list update algorithm, SPLIT, achieving a competitive ratio of 1.932. Albers, von Stengel, and Werchner [6] introduced COMB, a simple randomized algorithm that achieves a competitive ratio of 1.6. Albers [7, 8] further improved the analysis of randomized algorithms and introduced the concept of look-ahead, obtaining improved competitive ratios for deterministic algorithms.

Reingold and Westbrook [9, 10] proposed an optimal offline algorithm running in $O(2^\ell \ell! n)$ time, where ℓ is the list length and n is the request sequence length. They also developed randomized competitive algorithms for the list update problem. Andrew and Gleich [16] showed that the randomized BIT algorithm is 7/4-competitive using a potential function argument. They introduced the pairwise property and demonstrated that COMB—a combination of BIT and TIMESTAMP—achieves a competitive ratio of 8/5.

Boyar et al. [11] studied the online list update problem under the advice model of computation, where an online algorithm receives partial information about unknown input in the form of advice bits generated by a benevolent offline oracle. Albers and Janke [12] proved that no randomized online algorithm can achieve a competitive ratio smaller than 2 in the partial cost model, where accessing the i-th item incurs a cost of $i - 1$ rather than i. Ehmsen et al. [13] introduced list factoring and relative worst order analysis for the problem. Garefalakis [14] proposed a new family of randomized algorithms for list accessing. Renault [15] provided lower and upper bounds for online algorithms with advice.

In this paper, we focus exclusively on the dynamic model of the list access problem, where the list size varies over time and insert, delete, and access operations are permitted. We present a formal quantitative analysis of several classical algorithms adapted to the dynamic setting, namely MTF, RMTF, BIT, and TIMESTAMP.

The remainder of this paper is organized as follows. Section 2 analyzes the dynamic MTF algorithm. Section 3 presents the analysis of the dynamic RMTF and BIT algorithms. Section 4 examines the dynamic TIMESTAMP algorithm. Finally, Section 5 concludes the paper and discusses directions for future research.

2 Analysis of the Dynamic MTF Algorithm

Theorem 2.1. *Move-to-Front (MTF) is* $\left(2 - \frac{2}{\ell+1}\right)$*-competitive, where* ℓ *is the size of a static list or an upper bound on the size of a dynamic list at any point in time.*

Proof. To establish the desired upper bound, we first recall from [1] that:

$$\mathrm{MTF}(\sigma) \leq 2 \cdot \mathrm{OPT}(\sigma) - n, \tag{1}$$

where σ is the request sequence and $n = |\sigma|$ denotes its length.
It suffices to prove that $\mathrm{OPT}(\sigma) \leq n\left(\frac{\ell+1}{2}\right)$. To this end, we employ a factorization lemma. Consider first the case where the list contains only two items, x and y, and let σ_{xy} denote the request sequence restricted to these two elements. We aim to show that $\mathrm{OPT}(\sigma_{xy}) \leq 3n/2$. Note that in the factorization lemma [13], the cost of accessing the requested element is also taken into account.
We examine the worst-case scenario that an adversary may construct for the optimal algorithm on a list of two items. Suppose that in the request sequence σ_{xy}, the element x appears k times and y appears $n-k$ times. Without loss of generality, assume $k \geq n/2$; otherwise, we can swap the roles of x and y. The optimal algorithm places the more frequently requested element x at the front after its first access. Consequently, accessing each x incurs a cost of 1, while accessing each y incurs a cost of 2. Computing the total cost incurred by OPT leads to:

$$\mathrm{OPT}(\sigma_{xy}) = k \cdot 1 + (n-k) \cdot 2 = 2n - k. \tag{2}$$

Since $k \geq n/2$, we have $2n - k \leq 3n/2$, which yields $\mathrm{OPT}(\sigma_{xy}) \leq 3n/2$.
By the factorization lemma, extending this bound to a list of length ℓ gives:

$$\mathrm{OPT}(\sigma) \leq n\left(\frac{\ell+1}{2}\right). \tag{3}$$

Rearranging this inequality, we obtain:

$$\frac{2 \cdot \mathrm{OPT}(\sigma)}{\ell+1} \leq n. \tag{4}$$

Substituting (4) into (1) yields:

$$\begin{aligned} \mathrm{MTF}(\sigma) &\leq 2 \cdot \mathrm{OPT}(\sigma) - \frac{2 \cdot \mathrm{OPT}(\sigma)}{\ell+1} \\ &= \mathrm{OPT}(\sigma)\left(2 - \frac{2}{\ell+1}\right). \end{aligned} \tag{5}$$

Thus,

$$\frac{\mathrm{MTF}(\sigma)}{\mathrm{OPT}(\sigma)} \leq 2 - \frac{2}{\ell+1}, \tag{6}$$

which completes the proof. □

3 Analysis of the Dynamic RMTF$_p$ Algorithm

Theorem 3.1. *For the Randomized Move-to-Front algorithm RMTF$_p$, which moves the requested item to the front with probability $p \in (0,1)$, there exists a lower bound of $\frac{1}{p} - \varepsilon$ on its competitive ratio for any $\varepsilon > 0$.*

Proof. We construct an adversarial request sequence that forces RMTF_p to incur a cost arbitrarily close to $\frac{1}{p}$ times the optimal cost.
Let k be a sufficiently large integer to be determined later. Consider a dynamic list and define the request sequence:

$$\sigma = (x_i)^k, (x_{i-1})^k, \ldots, (x_1)^k, \tag{7}$$

where each $(x_j)^k$ denotes k consecutive requests to item x_j, and i ranges over the number of distinct items in the list. Since insertions and deletions are permitted in the dynamic model, i can vary over time.
We first analyze the expected number of accesses required for an item to move to the front. For a given item, the probability that it is moved to the front on the t-th access (and not earlier) is $(1-p)^{t-1}p$. Computing this expected value leads to:

$$\begin{aligned}\mathbb{E}[\text{accesses to move to front}] &= \sum_{t=1}^{\infty} t(1-p)^{t-1}p \\ &= \frac{1}{p}.\end{aligned} \tag{8}$$

This is a standard result for the geometric distribution with success probability p.
Now consider the subsequence $(x_i)^k$. Before x_i is moved to the front, each access to x_i incurs a cost equal to its current position i. After x_i is moved to the front (which occurs after approximately $1/p$ accesses in expectation), the remaining accesses cost 1 each. Therefore, the expected cost incurred by RMTF_p for processing $(x_i)^k$ is at least:

$$\mathbb{E}[\mathrm{RMTF}_p((x_i)^k)] \geq \frac{1}{p} \cdot i + \left(k - \frac{1}{p}\right) \cdot 1. \tag{9}$$

For the entire sequence σ, summing over all items gives the total expected cost:

$$\mathbb{E}[\mathrm{RMTF}_p(\sigma)] \geq \sum_{j=1}^{i} \left(\frac{1}{p} \cdot j + \left(k - \frac{1}{p}\right)\right). \tag{10}$$

Computing this sum leads to:

$$\begin{aligned}\mathbb{E}[\mathrm{RMTF}_p(\sigma)] &\geq \frac{1}{p}\sum_{j=1}^{i} j + i\left(k - \frac{1}{p}\right) \\ &= \frac{1}{p} \cdot \frac{i(i+1)}{2} + i\left(k - \frac{1}{p}\right) \\ &= \frac{i}{2p}(i+1) + i\left(k - \frac{1}{p}\right).\end{aligned} \tag{11}$$

The optimal offline algorithm, knowing the sequence in advance, can arrange the list optimally. For the subsequence $(x_j)^k$, OPT will place x_j at the front before processing its k requests, incurring a cost of 1 per access. Thus, for the entire sequence:

$$\mathrm{OPT}(\sigma) \leq i \cdot k. \tag{12}$$

Taking the ratio and letting i and k grow large:

$$\frac{\mathbb{E}[\mathrm{RMTF}_p(\sigma)]}{\mathrm{OPT}(\sigma)} \geq \frac{\frac{i}{2p}(i+1) + i\left(k - \frac{1}{p}\right)}{ik} = \frac{i+1}{2pk} + 1 - \frac{1}{pk}. \tag{13}$$

By choosing k sufficiently large relative to i, the term $\frac{i+1}{2pk}$ becomes arbitrarily small. Moreover, we can let $i \to \infty$ to obtain:

$$\lim_{i,k\to\infty} \frac{\mathbb{E}[\mathrm{RMTF}_p(\sigma)]}{\mathrm{OPT}(\sigma)} = \frac{1}{p}. \tag{14}$$

Therefore, for any $\varepsilon > 0$, we can construct a request sequence such that:

$$\frac{\mathbb{E}[\mathrm{RMTF}_p(\sigma)]}{\mathrm{OPT}(\sigma)} > \frac{1}{p} - \varepsilon, \tag{15}$$

which establishes the lower bound. □

4 Analysis of the Dynamic BIT Algorithm

The BIT algorithm is a randomized extension of MTF. During initialization, BIT assigns a random bit to each element in the list. Formally, for each element x, a corresponding bit value $b(x) \in \{0, 1\}$ is chosen uniformly at random. Whenever x is accessed, its bit is complemented. If after complementing we have $b(x) = 1$, then x is moved to the front of the list; otherwise, the list remains unchanged. It has been established that in the static list access model, BIT is 7/4-competitive against oblivious adversaries [10].

In the dynamic variant of BIT, insertion and deletion operations are also permitted. We show that the same competitive ratio holds in this setting.

For an insertion operation, the new item is appended to the end of the list, and its corresponding bit value is initialized randomly. If this bit is 1, the new element is immediately moved to the front. Observe that inserting a new element is equivalent to the first access to this $(n + 1)$-st element of the list, where n is the current list size. Consequently, the expected cost of an insertion is identical to that of an access operation. Formally:

$$\mathbb{E}[\mathrm{BIT}(\mathrm{insert}(x))] \leq \frac{7}{4} \cdot \mathrm{OPT}(\mathrm{insert}(x)). \tag{16}$$

A deletion operation can also be analyzed similarly to an access, but with an important distinction: upon deletion, all inversions involving the deleted element are removed. This reduces the potential function value compared to an access operation. Therefore, the competitive ratio for deletions does not exceed that for accesses:

$$\mathbb{E}[\mathrm{BIT}(\mathrm{delete}(x))] \leq \frac{7}{4} \cdot \mathrm{OPT}(\mathrm{delete}(x)). \tag{17}$$

Combining (16) and (17) with the known bound for access operations, we conclude that BIT maintains its 7/4-competitive ratio in the dynamic model.

5 Analysis of the Dynamic TIMESTAMP Algorithm

The TIMESTAMP algorithm, introduced in [8], is a deterministic 2-competitive online algorithm for the list update problem. To describe its behavior, consider a request sequence $\sigma = r_1, r_2, \ldots, r_m$. For a request r_j to an item x, let r_i be the most recent previous request to x (if such exists). Let y be the item nearest to the front of the list that satisfies either of the following conditions: (i) y was not requested between r_i and r_j, or (ii) y was requested exactly once between r_i and r_j, and that request was processed according to the TIMESTAMP procedure [15].

Procedure TIMESTAMP(r_j):

1. If r_j is the first request to x, do not move x.
2. Otherwise, x has been requested previously and y is well-defined. Move x to the position immediately in front of y [15].

In the dynamic list access model, three types of operations are handled according to this procedure:

- **Access**(i): access the item at position i in the list.
- **Delete**(i): remove the item at position i from the list.
- **Insert**(i): insert a new item at position i in the list.

The ordering properties of TIMESTAMP can be characterized by examining the restricted request sequence σ_{xy}, which contains only requests to items x and y. After processing σ, item x precedes item y if and only if one of the following holds:

- x was initially before y and y was requested at most once during σ, or
- σ_{xy} ends in one of the patterns: xx, xyx, or xxy [16].

These patterns can be interpreted as follows:

1. If σ_{xy} ends in xx or xyx, then y is requested at most once between the two final requests to x, causing x to be moved before y [16].
2. If σ_{xy} ends in xxy, then x is requested twice consecutively, placing x before y. Moreover, if y is requested only once in this pattern, it never moves in front of x; consequently, if x starts before y, it ends before y [16].

Consider σ_{xy} as an arbitrary request sequence restricted to elements x and y. Assuming that x is initially at the front of the list, the subsequence can take one of the following forms:

- $x^\ell yy$,

- $x^\ell(yx)^k yy$,
- $x^\ell(yx)^k x$,

where $\ell \geq 0$ and $k \geq 1$. If the sequence ends in xx, then x moves to the front and remains the first element. Similarly, if the subsequence ends in yy, then y moves to the front and becomes the first element. By symmetry, when y is initially at the front, the corresponding forms are:

- $y^\ell xx$,
- $y^\ell(xy)^k xx$,
- $y^\ell(xy)^k y$,

with $\ell \geq 0$ and $k \geq 1$ [16].
This classification provides a unique partition of possible request patterns, as one of these forms must match at each step. If the request sequence does not end with two consecutive accesses to the same element, the last request can be repeated to achieve such a partition, incurring only a small additional cost. Moreover, by symmetry, analyzing the case where x precedes y suffices, as the case where y precedes x follows analogously [16].
The TIMESTAMP algorithm is deterministic. Therefore, its cost for each pattern, under the standard cost model where accessing the i-th element incurs cost i, is given in Table 1.

Table 1: The cost of various request subsequence patterns for TIMESTAMP [16].

Request sequence	**TIMESTAMP cost**
$x^\ell yy$	2
$x^\ell(yx)^k yy$	$2k$
$x^\ell(yx)^k x$	$2k-1$

The cost analysis for each pattern in Table 1 can be explained as follows:

1. For the sequence $x^\ell yy$, the cost is 2 because y does not move to the front until its second access.
2. In the sequence $x^\ell(yx)^k yy$, the first access to y costs 1, and the subsequent access to x costs 0 (since x is already at the front). All remaining requests to x and y cost 1 each, as they are repeatedly moved to the front upon each access. Computing the total cost gives $1 + 0 + 2(k-1) + 1 = 2k$ [16].
3. For the sequence $x^\ell(yx)^k x$, the analysis is similar, but without the final pair of y's, resulting in a total cost of $2k-1$ [16].

These calculations confirm the costs presented in Table 1 and establish that TIMESTAMP is 2-competitive for access operations [16].

Delete(i):

The relative order of any two elements x and y depends only on the pattern of requests to these elements, not on other elements in the list or request sequence [3]. When an element y is deleted from any of the three patterns ($x^\ell yy$, $x^\ell (yx)^k yy$, or $x^\ell (yx)^k x$), there is no longer a y to compare with x. Consequently, x remains stationary at the front of the list, incurring zero additional cost. The same reasoning applies symmetrically when x is deleted.

Insert(i):

Inserting new elements x and y does not introduce any new request patterns beyond the three forms analyzed above ($x^\ell yy$, $x^\ell (yx)^k yy$, and $x^\ell (yx)^k x$). Therefore, the cost of insertion operations follows the same analysis as access operations, and TIMESTAMP remains 2-competitive for insertions as well.
Combining the analyses for access, delete, and insert operations, we conclude that TIMESTAMP maintains its 2-competitive ratio in the dynamic list access model.

6 Conclusion and Future Directions

This paper presented a theoretical analysis of dynamic online list access algorithms—MTF, RMTF, BIT, and TIMESTAMP—where insertions and deletions are permitted alongside accesses. While the static variant is well-studied, the dynamic setting has received less attention. Our work addresses this gap by establishing tight competitive ratios for these algorithms in the dynamic model.
We proved that MTF is $\left(2 - \frac{2}{\ell+1}\right)$-competitive, refining the classical 2-competitive bound by incorporating list size. For RMTF_p, we established a lower bound of $\frac{1}{p} - \varepsilon$. We showed that BIT maintains its 7/4-competitive ratio, and TIMESTAMP remains 2-competitive for all three operations. These results extend foundational work by Sleator and Tarjan [4], Irani [5], and Albers et al. [6, 8] to the dynamic setting.
Our findings have implications for systems requiring dynamic list management, such as caching, database indexing, and network buffers. Future directions include establishing matching lower bounds, investigating hybrid algorithms, incorporating locality of reference, and empirical validation of these theoretical results.

References

[1] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.

[2] J. McCabe, "On serial files with relocatable records," *Operational Research*, vol. 12, pp. 609-618, 1965.

[3] J. H. Hester and D. S. Hirschberg, "Self-organizing linear search," *ACM Computing Surveys*, vol. 17, no. 3, pp. 295-312, 1985.

[4] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202-208, 1985.

[5] S. Irani, "Two results on the list update problem," *Information Processing Letters*, vol. 38, no. 6, pp. 301-306, 1990.

[6] S. Albers, B. von Stengel, and R. Werchner, "A combined BIT and TIMESTAMP algorithm for the list update problem," *Information Processing Letters*, vol. 56, no. 3, pp. 135-139, 1995.

[7] S. Albers, "Improved randomized on-line algorithms for the list update problem," in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1995, pp. 412-419.

[8] S. Albers, "Improved randomized online algorithms for the list update problem," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 670-681, 1998.

[9] N. Reingold and J. Westbrook, "Optimum offline algorithms for the list update problem," Yale University, Technical Report YALEU/DCS/TR-805, 1990.

[10] N. Reingold, J. Westbrook, and D. Sleator, "Randomized competitive algorithms for the list update problem," *Algorithmica*, vol. 11, no. 1, pp. 15-32, 1994.

[11] J. Boyar, S. Kamali, K. S. Larsen, and A. López-Ortiz, "On the list update problem with advice," *Information and Computation*, vol. 253, pp. 411-423, 2017.

[12] S. Albers and M. Janke, "New bounds for randomized list update in the paid exchange model," in *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[13] M. R. Ehmsen, J. S. Kohrt, and K. S. Larsen, "List factoring and relative worst order analysis," *Algorithmica*, vol. 66, no. 2, pp. 287-309, 2013.

[14] T. Garefalakis, "A new family of randomized algorithms for list accessing," Department of Computer Science, University of Toronto, Canada.

[15] M. Renault, "Lower and upper bounds for online algorithms with advice," Ph.D. dissertation, Université Paris Diderot–Paris, 2014. [Online]. Available: https://www.irif.univ-paris-diderot.fr/ mrenault/papers/renaultPhD.pdf

[16] K. Andrew and D. Gleich, "MTF, BIT, and COMB: A guide to deterministic and randomized online algorithms for the list access problem," Advanced Algorithms, Harvey Mudd College, Final Project, 2004.