This paper presents the amortized complexity of “move-to-front” rule in unsorted linear list and “least recently used” replacement rule in paging. Amortization is a process computing average running time of an algorithm over a succession of executions, which offers a better efficiency analysis than average-case and worst-case measures.

Firstly, a dictionary problem, which refers to the maintenance of a set of items that undergo a sequence of actions consisting of accessing, insertion and deletion, is introduced. A simple solution is to put the set in an unordered list and operations are carried out correspondingly: access/delete an item by looking through the list to find/delete the target one; insert an item by checking duplicate items in the list first and appends it to the rear; sometimes swapping of neighboring items is performed to fasten future operations. The costs of different operations are also brought in: assessing/deleting the i^{th} item cost i; inserting cost i+1; free exchange has no cost while paid exchange costs 1.

Three rule regarding the self-organizing linear lists are examined: move-to-front (MF), pushing the item to the front of the list; transpose (T), or exchange with item before itself; frequency count (FC), or keeping an access counter. In a situation of a fixed set of n items, when accesses are allowed and exchanges are disallowed, an optimum off-line algorithm called decreasing probability (DP), is to implement a list in non-increasing probability order. This proves that transpose performs better than move-to-front, which is contrary to what people see in practice. The reason for this is MF converges rapidly to its asymptotic behavior for arbitrary initial list. It is also discovered that an optimum static algorithm called decreasing frequency costs much less than the above three rules for an initial orderly list. The tests in real data show that transpose performs worse than FC, which itself is less competitive than move-to-front. All these lead to the first theorem, which states that the total operational cost of MF is not more than (2xcost of algorithm A+ its number of paid exchanges-its number of free exchanges-m).

In the proof of Theorem 1 the author introduces the concept of potential function, which is number of inversions between MF’s list and A’s list. With the assumption of zero potential in initial configuration and nonnegative potential in final configuration, the actual cost to MF is bounded by the sum of the operations’ amortized times, which are in turn bounded by right-hand side of the inequality in theorem 1. Then after some derivations it is concluded that the amortized time for access, insertion or deletion is at most 2i-1, where i is the position of target item, while that for exchange is the increase of number of inversions due to the action. In all, Theorem 1 generalizes the conditions of nonempty initial set and different lists of MF and A. And it could be used to bound the cost of MF in exchange.

The abovementioned proof applies any update rule and this is followed by Theorem 2, which states the movement of target item at position k closer to the front of the list by at least k/d-1 unites will result in a cost less than d(RHS of Theorem 1). With the consideration of the potential function being d times the number of inversions, the author shows the amortized time for MF(d) is at most d(2i-1) to access, d(2i+1)-1 to insert, i to delete, –d for free exchange and d for paid exchange. For transpose and FC, two counterexamples are given respectively: inserting n items

Comment [S1]: Content: A
Organization: A-
Writing: A
Grading Criteria
Content: Does your summary contain the most important information presented in the paper? Is the content correct, or does it contain technical errors?
Organization: Do you have good ordering of the sections/subsections? Are the sections linked together well or do they appear abruptly? Does the paper flow? Did you motivated/introduce/define items appropriately before using them?
Writing: Did you use words and grammar correctly. Does the writing have a clear and easy to understand style?

and then repeatedly access the last two, interchanging between them; inserting items one by one and accessing them in decreasing number of times starting from k+n.

Later, Theorem 3 states that there exists less expensive algorithm performing a sequence of insertions and accesses, s, without paid exchange for any algorithms doing the same sequence. Here there is no assumption of empty initial set. Since insertion and access cost the same, it is assumed that s consists of only accesses. To remove the paid exchanges, we either convert the paid exchange to free or move it after an access; this completes the derivation of theorem 3. Unlike Theorem 1, it does not require convexity of the cost function involved.

Theorem 4 shows that for convex cost function f, cost of MF is not more than 2*(2xcost of algorithm A+ its number of paid exchanges-m*f(1)). By following a similar pattern of proof, it could be derived that the amortized time for MF is at most 2f(i)-f(1) to access item I, 2f(i+1)-f(1) to insert, $\triangle$f(i) to interchange item i and item i+1. To sum up, Theorem 3 and 4 provide further theoretical support to the favorable efficiency of MF.

The above model applies to the case when access cost is not convex, i.e. paging. It could be understood by a two-level memory consisting of slow memory and fast memory; any access to a page in fast memory cost nothing while there is a cost called page fault for access to slow memory because the system needs to move that page to fast memory. Five paging rules have been studies: Least-Recently-Used (LRU), replacing page with earliest recent visit; First-in-first-out (FIFO), replacing the page that stays the longest; Last-in-first-out (LIFO), replacing the page staying with shortest time; Least-Frequently-Used (LFU), replacing the page with least access; Longest-Forward-Distance (MIN), replacing page with latest next access.

The good performance of MIN can be shown by comparing it with any on-line algorithms and this is theorized in Theorem 5, which states the number of page faults of other algorithms is not less than $(n_a/(n_a-n_{min}+1))$*number of page faults in MIN where n_x indicates number of pages in A's fast memory.

Subsequently the paper presents Theorem 6, which states the number of page faults of LRU is less than $(n_{LRU}/(n_{LRU}-n_{MIN}+1))F_{MIN}(s)+n_{MIN}$ for any sequence s. One important step for deriving this is that when LRU faults at least f different pages ($f \leq$ nLRU), MIN must fault at least f-nMIN+1 times.