On online labeling with large label set*

Martin Babka†
Charles University, Prague
babkys@gmail.com

Jan Bulánek
Charles University and
Institute of Mathematics
Academy of Sciences CR, Prague
honyai@seznam.cz

Vladimír Čunát
Charles University, Prague
vcunat@gmail.com

Michal Koucký ‡
Institute of Mathematics
Academy of Sciences CR, Prague
koucky@math.cas.cz

Michael Saks§
Department of Mathematics
Rutgers University
saks@math.rutgers.edu

July 17, 2015

Abstract

In the online labeling problem with parameters n and m we are presented with a sequence of n *items* from a totally ordered universe U and must assign each arriving item a label from the label set $\{1, \ldots, m\}$ so that the order of labels respects the order on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

For the case $m = cn$ for constant $c > 1$, there are known algorithms that use at most $O(n(\log n)^2)$ relabelings in total [16], and it was shown recently that this is asymptotically optimal [3, 4]. For the case of $m = \Theta(n^{1+C})$ for constant $C > 0$, algorithms are known that use $O(n \log n)$ relabelings. A matching lower bound was provided in [12]. The lower bound

*A preliminary version of this paper appeared in the Proceedings of the 2012 European Symposium on Algorithms [1]

†The first three authors gratefully acknowledge support from the Charles University Grant Agency (grant No. 265 111 and 344 711) and SVV project no. 265 314.

‡Part of the work done while visiting Aarhus University, partially supported by the Sino-Danish Center CTIC (funded under the grant 61061130540). Supported in part by GA ČR P202/10/0854, grant IAA100190902 of GA AV ČR, Center of Excellence CE-ITI (P202/12/G061 of GA ČR) and RVO: 67985840.

§The work of this author was done while on sabbatical at Princeton University and was also supported in part by NSF under grants CCF-0832787 CCF-1218711.

proof had two parts: a lower bound for a problem they call *prefix bucketing* and a reduction from prefix bucketing to online labeling. In this paper we present a simplified version of their reduction, together with a full proof (which was not given in [12]). Furthermore we give a simplified and improved analysis of the prefix bucketing lower bound. This improvement allows us to extend the lower bounds for online labeling to the case where the number m of labels is superpolynomial in n. In particular, our lower bound for m from n^{1+C} to 2^n is $\Omega((n \log n)/(\log \log m - \log \log n))$. This reduces to the asymptotically optimal bound $\Omega(n \log n)$ when $m = \Theta(n^{1+c})$. We show that our bound is asymptotically optimal for the case of large m by giving a matching upper bound in the case that $m \geq 2^{1+(\log n)^3}$.

1 Introduction

In the online labeling problem with parameters n, m, r, we are presented with a sequence of n *items* from a totally ordered universe U of size r and must assign each arriving item a label from the label set $\{1, \ldots, m\}$ so that the order of labels respects the ordering on U. As new items arrive it may be necessary to change the labels of some items; such changes may be done at any time at unit cost for each change. The goal is to minimize the total cost. An alternative formulation of this problem is the *file maintenance problem*, in which the items, instead of being labeled, are maintained in sorted order in an array of length m, and we pay unit cost for moving an item.

The problem, which was introduced by Itai, Konheim and Rodeh [16], is natural and intuitively appealing, and has had applications to the design of data structures (see for example the discussion in [12], and the more recent work on cache-oblivious data structures [7, 9, 8]). A connection between this problem and distributed resource allocation was recently shown by Emek and Korman [15].

The parameter m, the *label space* must be at least the number of items n or else no valid labeling is possible. There are two natural range of parameters that have received the most attention. In the case of *linearly many labels* we have $m = cn$ for some $c > 1$, and in the case of *polynomially many labels* we have $m = \theta(n^{1+C})$ for some constant $C > 0$. The problem is trivial if $|U| \leq m$, since then we can fix an order preserving bijection from U to $\{1, \ldots, m\}$ in advance.

Known upper bounds Itai et al. [16] gave an algorithm for the case of linearly many labels having worst case total cost $O(n(\log n)^2)$. Improvements and simplifications were given by Willard [18] and Bender et al. [6]. In the special case that $m = n$, algorithms with cost $O((\log n)^3)$ per item were given [19, 10]. It is also well known that the algorithm of Itai et al. can be adapted to give total cost $O(n \log n)$ in the case of polynomially many labels. An algorithm with $O(\log n)$ worst case cost per insertion was given in [17]. All of these algorithms make no restriction on the size of universe U of items.

Known lower bounds For the case of polynomially many labels, Dietz at al. [12] (also in [19]) proved a matching lower bound for the $O(n \log n)$ upper bound.

For the case of linearly many labels ($m = O(n)$), Dietz and Zhang ([14, 13], also available in Zhang's Ph.D. thesis [19]) proved an $\Omega(n \log^2 n)$ lower bound for a restricted class of algorithms, called *smooth algorithms*. A subset of the present authors [3] showed that the same (tight) lower bound holds for any online labeling algorithm. These bounds hold even when the size of the universe U is only a constant multiple of m. The bound remains non-trivial (superlinear in n) for $m = O(n(\log n)^{2-\varepsilon})$ but becomes trivial for $m \in \Omega(2^{\log^2 n})$.

Our results In this paper we extend and clarify the lower bound of Dietz et al. [12]. Their result consists of two parts; a lower bound for a problem they call *prefix bucketing* and a reduction from prefix bucketing to online labeling. Our results are:

- We provide a simpler and more precise lower bound for the prefix bucketing problem. (This is given in Section 3.2)
- We clarify the reduction from prefix bucketing to online labeling and give a full proof of correctness. (This is given in Sections 2.2 and 3.1.) In [12], only a proof sketch is provided.[1]
- Using our improved analysis for prefix bucketing, we extend the range of validity of the lower bound results from polynomial space to exponential space. Specifically we prove a lower bound of $\Omega((n \log n)/(1 + \log\log m - \log\log n))$ that is valid for m between n and 2^n. (This appears in Section 3.3.) Note that for polynomially many labels this reduces to $\Omega(n \log n)$ which matches the known upper bound up to a constant factor.
- We give an algorithm for the case that $m \geq 1+2^{(\log n)^3}$ whose cost is $\Omega((n \log n)/(\log\log m))$. (This appears in Section 4.) This matches our lower bound up to a constant factor. Thus the lower bound is tight for the case that $m = n^{1+C}$ and $C > 0$ is a constant, and also for the case $m \geq 1 + 2^{(\log n)^3}$.[2]

We summarize known results about deterministic algorithms in Table 1. (There is a brief discussion of randomized algorithms below.)

Further developments and open questions The online labeling problem has three parameters: the number m of possible labels , the number n of items, and the size r of the universe of possible items. The lower bounds we obtain in this paper for the case $m = \Omega(n^{1+C})$ for some positive C, require that r be exponential in n. This contrasts with the lower bounds obtained in [3] for the case of $m = O(n)$, where the lower bounds hold even when r is a (sufficiently large) linear function of n. This leaves open the intriguing question whether in the case of polynomially many labels $m = n^{1+C}$, one can improve on the $O(n \log n)$ cost algorithm if the range r is not too large, e.g. polynomial in m, or even $O(m \log n)$.

[1] Our initial study of their proof led us to think that it had a significant gap. In the conference version of this paper we asserted (with the concurrence of one of the authors of the original paper) that we were filling a significant gap in their proof. We have since realized that their proof, though lacking in detail, is essentially correct.

[2] This algorithm appeared as a minor part of a conference paper [3] and is included here rather that in the journal version [4] of [3] because of its close connection to the other results in this paper.

Table 1: Summary of known bounds for deterministic online labeling problem.

Array size (m)	Asymptotic bound	Lower bound	Upper bound
$m = n$	$\Theta\big(n(\log n)^3\big)$	[3]	[19]
$m = cn$, constant $c > 1$	$\Theta\big(n(\log n)^2\big)$	[3]	[16]
$m = n^C$, constant $C > 1$	$\Theta(n \log n)$	[12]	[16]
$m = n^{\omega(1)}$	$\Omega\frac{n \log n}{1+\log\log m-\log\log n}$	[this paper]	
$m \geq 1 + 2^{\log^3 n}$	$\Theta\Big(\frac{n \log n}{\log\log m}\Big)$	[this paper]	[3],[this paper]

All of the work mentioned so far (both upper and lower bounds) is for the case of deterministic algorithms. In work subsequent to this paper, a subset of the authors [5] showed that the $\Omega(n \log n)$ lower bound for the case $m = n^{1+C}$ of polynomially many labels extends for randomized algorithms. The results in that paper do not subsume this one because (a) The lower bounds for randomized algorithms don't extend to the range that m is superpolynomial in n (as do the results here) and (b) The lower bound proofs here are conceptually simpler.

In the case of linearly many labels, the $\Omega(n \log n)$ lower bound for randomized algorithms applies, but this leaves a gap since the best upper bound is the deterministic $O(n \log^2 n)$ algorithm. Is there a better randomized algorithm in the case of linearly many labels, or can the lower bound be improved?

1.1 Overview of the proof

Our proof follows the high level structure of the proof from [12]. In the remainder of the introduction we sketch the two parts, and relate our proof to the one in [12].

Reducing Prefix Bucketing to Online Labeling Dietz et al. [12] describe a particular adversary for the labeling problem and show that given any algorithm for online labeling, the behavior of the algorithm against the adversary can be used to construct a strategy for prefix bucketing. In Section 2 we will present a modification of their adversary, and in Section 3.1 we give a full proof of the connection to prefix bucketing. We now sketch the adversary construction and the reduction.

The goal in constructing an adversary is to force any online algorithm to perform many relabelings during insertion of n items. As the adversary proceeds we will refer to items that have been inserted by the adversary as *active*, and two active items are *adjacent* if there is no active item between them. The adversary starts by inserting the seven equally spaced items including the minimum and maximum items 1 and r. Each successive inserted item will be the average (rounded down) of some pair of adjacent active items. The central issue in defining the adversary is to determine at each step which adjacent pair of active items to choose.

It is illuminating to think of this in terms of the file maintenance problem mentioned earlier. In this reformulation the label space $\{1, \ldots, m\}$ is associated to an array indexed by $\{1, \ldots, m\}$ and an item labeled by j is viewed as stored in location j. Intuitively, the adversary

wants to choose each successive insertion to be the average of two adjacent items that appear in a "crowded" region of this array. The hope is that this will eventually force the algorithm to move many items within the array (which corresponds to relabeling them). The problem is to make precise the notion of "crowdedness". Crowding within the array occurs at different scales (so a small crowded region may lie inside a large uncrowded region) and we need to find a pair of adjacent items with the property that all regions containing the pair are somewhat crowded.

With the array picture in mind, we call an interval of labels a *segment*, and say that a label is *occupied* if there is an item assigned to it. The *density* of a segment is the fraction of occupied labels.

As a guide to picking each successive item, the adversary maintains a sequence (*hierarchy*) of nested segments. Each successive segment in the hierarchy has size at most half the previous segment, and its density is within a constant factor of the density of the previous segment. The hierarchy ends with a segment having between 2 and 7 items. The next item to be inserted is the average (rounded down) of the two middle items in the lowest segment of the hierarchy.

In [12], the authors show that there is always a *dense point*, which is a point with the property that every segment containing it has density at least half the overall density of the label space. They use this as the basis for building the hierarchy at each step. We build a hierarchy with similar properties to theirs using a sowmewhat simpler argument.

Before the 8th insertion, the hierarchy consists of just the single segment $\{1, \ldots, m\}$. After each subsequent insertion, the algorithm $\mathcal{A}$ specifies the label of the next item and (possibly) relabels some items. Say that a label is *impacted* if it is either assigned by the algorithm to some item during this step, or is freed up (because the item previously assigned to it was assigned to a different label.) The adversary then updates the hierarchy as follows. For the hierarchy immediately prior to the insertion, the *critical segment* is the smallest segment in the hierarchy that contains all labels impacted by the insertion. The new hierarchy agrees with the previous hierarchy on all segments from the largest segment to the critical segment. Beginning from the critical segment the rest of the hierarchy is "rebuilt" one segment at a time.

If the smallest segment selected so far is S and S has at most seven occupied labels then the hierarchy stops; if will follow from the construction that the final segment has at least two occupied labels. Otherwise we choose a successor to S as follows. Define the *left buffer* of S to be the smallest subsegment of S that starts at the minimum label of S and includes at least $1/8$ of the occupied labels of S, and the *right buffer* of S to be the smallest subsegment that ends at the maximum label of S and includes at least $1/8$ of the occupied items of S. Let S' be the segment obtained from S by deleting the left and right buffers. The successor segment of S in the hierarchy is a shortest subsegment of S' that contains exactly half (rounded down) of the occupied labels of S'.

It remains to prove that the algorithm will make many relabelings on the sequence of items produced by the adversary. Following [12], we do this by relating online labeling to the prefix bucketing game. (Our definition of the game differs slightly from that in [12].)

A *prefix bucketing of n items into k buckets* (numbered 1 to k) is a one player game consisting of n steps. At the beginning of the game all the buckets are empty. In each step a new item arrives and the player selects an index $p \in \{1, \ldots, k\}$. The new item as well as

all items in buckets $p+1, \ldots, k$ are moved into bucket p at a cost equal to the total number of items in bucket p after the move. The run of the game is therefore completely specified by the sequence $p^1, \ldots, p^n$, where p^t is the bin into which the player placed the new item at step t. The goal is to minimize the total cost of n steps of the game. Note that the items in this game are indistinguishable (i.e. they have no "names").

The lower bound on online labeling is obtained by showing that for any online labeling algorithm $\mathcal{A}$, the behavior of $\mathcal{A}$ against the adversary described above, can be used to give a strategy for prefix bucketing. Consider the run of an arbitrary online labeling algorithm $\mathcal{A}$ against the given adversary. At each step t, the adversary determines a particular level p^t of the hierarchy to be the critical level (which is always at most $k = \lceil \log m \rceil$, where in this paper $\log x$ stands for the binary logarithm function.) Consider the sequence $p^1, \ldots, p^n$ as a sequence of placements defining a prefix bucketing of n items into $k = \lceil \log m \rceil$ buckets. It turns out that the total cost of the prefix bucketing is within a constant factor of the total number of relabelings performed by the online labeling algorithm. Hence, a lower bound on the cost of a prefix bucketing of n items into k buckets will imply a lower bound on the cost of the algorithm against our adversary.

The connection between the cost of $\mathcal{A}$ against the adversary, and the cost of the associated prefix bucketing is obtained as follows. We make the assumption (which can be shown to hold without loss of generality) that the algorithm is *lazy*, which means that at each step the set of items that are relabeled is a contiguous block of items that includes the newly inserted items. The cost of the bucketing merge step p^t at step t is at most the number of items in the critical segment, so to relate this to the cost incurred by the online labeling algorithm, it is enough to argue that at step t a constant fraction of the items in the critical segment were relabeled. This is done by arguing that for each successor (sub)segment of the critical segment, either all labels in its left buffer or all labels in its right buffer were reassigned, and the total number of such items is a constant fraction of the items in the critical segment.

An Improved Analysis of Bucketing It then remains to give a lower bound for the cost of prefix bucketing. This was previously given by Dietz et al. [12] for $k \in \Theta(\log n)$. We give a different and simpler proof that gives an asymptotically optimal bound for k between $\log n$ and $O(n^\epsilon)$. We define a family of trees called k-admissible trees and show that the cost of bucketing for n and k, is between $dn/2$ and dn where d is the minimum depth of a k-admissible tree on n vertices. We further show that the minimum depth of a k-admissible tree on n vertices is equal $g_k(n)$ which is defined to be the smallest d such that $\binom{k+d-1}{k} \geq n$. This gives a characterization of the optimal cost of prefix bucketing (within a factor of 2). When we apply this characterization we need to use estimates of $g_k(n)$ in terms of more familiar functions (Lemma 24), and there is some loss in these estimates.

2 The Online Labeling Problem

Here we provide the formal definition of the online labeling problem. Let m be an integer. We have a totally ordered set U of *items* which we assume (without loss of generality) is a set of positive integers. An online labeling algorithm $\mathcal{A}$ with range m is an algorithm that on input

sequence $y^1, y^2, \ldots, y^t$ of distinct elements from U gives an *allocation* $f : \{y^1, y^2, \ldots, y^t\} \to \{1, \ldots, m\}$ that respects the natural ordering of $y^1, \ldots, y^t$, so that for $x, y \in \{y^1, y^2, \ldots, y^t\}$, $f(x) < f(y)$ if and only if $x < y$. We refer to $y^1, y^2, \ldots, y^t$ as *items*. The trace of $\mathcal{A}$ on a sequence $y^1, y^2, \ldots, y^n \in U$ is the sequence $f^0, f^1, f^2, \ldots, f^n$ of functions such that f^0 is the empty mapping and for $t = 1, \ldots, n$, f^t is the output of $\mathcal{A}$ on $y^1, y^2, \ldots, y^t$. For the trace $f^0, f^1, f^2, \ldots, f^n$ and $t = 1, \ldots, n$, we say that $\mathcal{A}$ *relocates* $y \in \{y^1, y^2, \ldots, y^t\}$ *at step* t if $f^{t-1}(y) \neq f^t(y)$. In particular, y^t is relocated at step t. For the trace $f^0, f^1, f^2, \ldots, f^n$ and $t = 1, \ldots, n$, $Rel^t = Rel^t_{\mathcal{A}}(y^1, \ldots, y^n)$ denotes the set of relocated items at step t. The cost of $\mathcal{A}$ incurred on $y^1, y^2, \ldots, y^n$ is $\chi_{\mathcal{A}}(y^1, \ldots, y^n) = \sum_{t=1}^{n} |Rel^t|$.

The maximum cost $\chi_{\mathcal{A}}(y^1, \ldots, y^n)$ over all sequences $y^1, \ldots, y^n$ of distinct items from U is denoted $\chi_{\mathcal{A}}(n, U)$; we write $\chi_{\mathcal{A}}(n)$ in the case that U is the set of positive integers. This maximum is well-defined since the cost of any algorithm on any sequence of length n is at most $\sum_{i=1}^{n} i = n(n+1)/2$. We define $\chi_m(n)$ to be the smallest cost $\chi_{\mathcal{A}}(n)$ that can be achieved by any algorithm $\mathcal{A}$ with range m.

2.1 The Main Theorem

Our main lower bound result for $\chi_m(n)$ is:

Theorem 1 *There are positive constants C_0, and C_1 so that the following holds: For integers m, n satisfying $C_0 \leq n \leq m \leq 2^n$.*

$$\chi_m(n) \geq C_1 \cdot \frac{n \log n}{1 + \log\log m - \log\log n}.$$

To prove the theorem we fix an online labeling algorithm $\mathcal{A}$ and describe an adversary that, based on the behavior of $\mathcal{A}$, selects a sequence $y^1, y^2, \ldots, y^n$ of items that will cause the algorithm to incur the desired cost. The input sequence selected by the adversary will be a subset of $\{1, \ldots, 2^n\}$.

In the next subsection we describe the adversary, and state Lemma 4, which asserts a lower bound on the cost incurred by algorithm $\mathcal{A}$ on the sequence produced by the adversary. Theorem 1 follows immediately from this lemma.

2.2 Adversary Construction

Fix n and m with $m \geq n$ and fix an online labeling algorithm $\mathcal{A}$ with range m. Our adversary will select the sequence of items $y^1, \ldots, y^n$, one by one. During step t the adversary chooses y^t, and the algorithm specifies the labeling of $y^1, \ldots, y^t$, which is denoted f^t. All of the y^j will be chosen from the set $\{1, \ldots, 2^n\}$.

The choice of y^t will depend on the labeling f^{t-1} of $y^1, \ldots, y^{t-1}$ from the previous step, and we'll need some notation and observations to describe this choice.

Any interval $\{a, \ldots, b\} \subseteq \{1, \ldots, m\}$ of label values is called a *segment*. The *population of a segment S after step $t-1$* is $\mathbf{pop}^{t-1}(S) = (f^{t-1})^{-1}(S)$ and the *weight of S after step $t-1$* is $\mathbf{weight}^{t-1}(S) = |\mathbf{pop}^{t-1}(S)|$. (As usual, $g^{-1}(A) = \{x : g(x) \in A\}$.) In particular, $\mathbf{pop}^0(S) = \emptyset$ and $\mathbf{weight}^0(S) = 0$. The *density of S after step $t-1$*, denoted $\rho^{t-1}(S)$, is

defined to be $\mathbf{weight}^{t-1}(S)/|S|$. For a positive integer b, and for any segment S such that $\mathbf{weight}^{t-1}(S) \geq 2b$, we define $\mathbf{densify}^{t-1}(S, b)$ to be the minimum subsegment T of S (with respect to the order that orders segments by size, and orders segments of the same size by their left endpoint) satisfying:

- $pop^{t-1}(T)$ does not contain any of the b largest or smallest elements of $\mathbf{pop}^{t-1}(S)$.
- $\mathbf{weight}^{t-1}(T) = \lfloor(\mathbf{weight}^{t-1}(S) - 2b)/2\rfloor$.

Hence, $\mathbf{densify}^{t-1}(S, b)$ is a densest subsegment of S that contains half (rounded down) of the middle $n - 2b$ items stored in S.

Proposition 2 *For a segment S, if $\mathbf{weight}^{t-1}(S) \geq 2b$ then $|\mathbf{densify}^{t-1}(S, b)| \leq |S|/2$.*

Proof: Let S' be the largest subsegment of S such that $\mathbf{pop}^{t-1}(S')$ excludes the smallest b members and the largest b members of $\mathbf{pop}^{t-1}(S)$. Let L' be the smallest subsegment of S' that starts at the left endpoint of S' and satisfies $\mathbf{weight}^{t-1}(L) = \lfloor\mathbf{weight}^{t-1}(S')/2\rfloor$. Let R' be the smallest subsegment of S' that ends at the right endpoint of S' and satisfies $\mathbf{weight}^{t-1}(L) = \lfloor\mathbf{weight}^{t-1}(S')/2\rfloor$. L' and R' are disjoint subsegments of S so the smaller of them has size at most $|S|/2$. Since $\mathbf{densify}^{t-1}(S, b)$ is the smallest subsegment T of S' with $\mathbf{weight}^{t}T = \lfloor\mathbf{weight}^{t-1}(S')/2\rfloor$, the proposition follows. □

Suppose S is a segment with $\mathbf{weight}^{t-1}(S) = \ell \geq 2$. Let $x_1 < \ldots < x_\ell$ denote the elements of $\mathbf{pop}^{t-1}(S)$. We define $\mathbf{midpoint}^{t-1}(S) = \lceil(x_{\lceil(\ell-1)/2\rceil} + x_{\lceil(\ell+1)/2\rceil})/2\rceil$. Thus $\mathbf{midpoint}^{t}(S)$ is obtained by averaging the two middle items whose labels belong to S.

Let $y^1, y^2, \ldots, y^t$ be the first t items inserted and let Rel^t be the items that are relabeled by $\mathcal{A}$ in response to the insertion of y^t. The *busy segment* $B^t \subseteq \{1, \ldots, m\}$ *at time* t is the smallest segment that contains $f^t(Rel^t) \cup f^{t-1}(Rel^t \setminus \{y^t\})$. (Equivalently, say that a label is *impacted at step* t if either it is unassigned under f^{t-1} and assigned under f^t or assigned under f^{t-1} and unassigned under f^t or assigned to different items under f^{t-1} and f^t. Then B^t is the smallest segment of labels that contains all labels that are impacted at step t.)

We say that the algorithm $\mathcal{A}$ is *lazy* if all the items that are mapped by f^{t-1} to B^t are relocated at step t, i.e., $\mathbf{pop}^{t-1}(B^t) = Rel^t \setminus \{y^t\}$. Proposition 4 in [3] shows that any algorithm $\mathcal{A}$ can be modified to get a lazy algorithm $\mathcal{A}'$ whose cost on any insertion sequence is no more than $\mathcal{A}$. (The intuition behind this fact is that if there is an item y such that $f^t(y) = f^{t-1}(y) \in B^t$ and (without loss of generality) $y < y^t$ then we could defer relocating all of the items less than y, thereby shrinking B^t.) Therefore it suffices to prove our lower bound for lazy algorithms, and from now on we assume that $\mathcal{A}$ is lazy. We record the following simple observation:

Proposition 3 *Consider an execution of a lazy algorithm $\mathcal{A}$ on $y^1, \ldots, y^n$. For any time $t \in \{1, \ldots, n\}$, $|Rel^t| = 1 + \mathbf{weight}^{t-1}(B^t)$.*

We are now ready to present our adversary. For the first 7 steps, the adversary chooses $y^1, y^2, \ldots, y^7$ by $y^i = 1 + 2^{n-7}(i-1)$; note that the difference between consecutive y^i is 2^{n-7} and that $y^7 < 2^n$.

From then on the adversary operates at each step t by constructing a nested sequence of segments $\{1,\ldots,m\}\, S_1^t \supseteq \cdots \supseteq S^t_{\mathbf{depth}(t)}$ called the *hierarchy at step t*. The procedure for specifying this hierarchy is given below. The final (smallest) segment will (necessarily) have weight at least 2, and y^t is selected to be $\mathbf{midpoint}^{t-1}(S^t_{\mathbf{depth}(t)})$. The construction of the hierarchy requires that when S_i^t is defined, we also specify an integer b_i^t (which is used as the buffer parameter in the function **densify**.) After y^t is selected, the algorithm $\mathcal{A}$ determines the labeling f^t of $y^1,\ldots,y^t$. This (together with f^{t-1}) determines the busy segment B^t defined earlier. The adversary defines the integer p^t, called the *critical level at step t*, to be the largest level such that $B^t \subseteq S^t_{p^t}$, and p^t is used in the next iteration to help define the hierarchy at step $t+1$.

Adversary$(\mathcal{A}, n, m)$ (We assume that $n \geq 8$)

- *The first 7 steps:* For t from 1 to 7, set $y^t = 1 + (i-1)2^{n-7}$.
- *Initialization for remaining steps:*
  - $\mathbf{depth}(7) = 1$.
  - $S_1^7 = \{1,\ldots,m\}$.
  - $p^7 = 1$.
  - $b_1^7 = 1$.
- *The remaining steps.* For t from 8 to n do
  - Set $i = 1$ (*i indexes the levels of the hierarchy*)
  - *Preservation Rule for first p^{t-1} levels:*While $i \leq p^{t-1}$
    - Set $S_i^t = S_i^{t-1}$
    - Set $b_i^t = b_i^{t-1}$.
    - Increase i by 1.

    (*The hierarchy segments and buffer parameters at step t agree with those at step $t-1$ up through level p^{t-1}.*)
  - *Rebuilding Rule for $i > p^{t-1}$:* While $\mathbf{weight}^{t-1}(S_i^t) \geq 8$
    - Increase i by 1.
    - Set $S_i^t = \mathbf{densify}^{t-1}(S_{i-1}^t, b_{i-1}^t)$.
    - Set $b_i^t = \lceil \mathbf{weight}^{t-1}(S_i^t)/8 \rceil$.
  - *The hierarchy is complete; record the depth.* $\mathbf{depth}(t) = i$.
  - *Choose the next item to insert:* Set $y^t = \mathbf{midpoint}^{t-1}(S^t_{\mathbf{depth}(t)})$.
  - *Determine the next critical level:* Run $\mathcal{A}$ on $y^1, y^2, \ldots, y^t$ to get f^t. Calculate Rel^t and B^t. Set p^t to be the largest integer $j \in \{1,\ldots,\mathbf{depth}(t)\}$ such that $B^t \subseteq S_j^t$.

Output: $y^1, y^2, \ldots, y^n$.

For $t \geq 8$, y^t is the midpoint between two adjacent earlier items. After step 7, two adjacent items differ by 2^{n-7} so an easy induction shows that after step t, any two adjacent items differ by a multiple of 2^{n-t}. Therefore the selected items are distinct integers in the set $\{1, \ldots, 2^n\}$.

The following claim about the adversary implies Theorem 1.

Lemma 4 *Let m, n be positive integers such that $n \leq m$. Let $\mathcal{A}$ be a lazy online labeling algorithm with range m. Let $y^1, y^2, \ldots, y^n$ be the output of* **Adversary**$(\mathcal{A}, n, m)$. *Then:*

$$\chi_{\mathcal{A}}(y^1, y^2, \ldots, y^n) \geq \frac{1}{512} \cdot \frac{n \log n}{1 + \log\lceil \log m \rceil - \log\log n} - \frac{n}{6}.$$

The constants are chosen for ease of exposition and can certainly be improved.

To prove the lemma we model the interaction between the adversary and the algorithm by a so-called *prefix bucketing game* and relate the cost of the prefix bucketing to the cost $\chi_{\mathcal{A}}(y^1, y^2, \ldots, y^n)$ (Lemma 12). Then we bound the cost of prefix bucketing from below (Lemma 25). These results are combined at the end of Section 3.3 to prove Lemma 4.

In preparation for this, we prove several useful properties of the adversary.

Lemma 5 *For any $t \in \{7, \ldots, n\}$,* $\mathbf{depth}(t) \leq \log m$.

Proof: We will show that for each fixed t, $|S_i^t| \leq |S_{i-1}^t|/2$ for each $i \in \{2, \ldots, \mathbf{depth}(t)\}$, from which the result follows immediately. We prove this by induction on t; the result is vacuous for $t = 7$ since $\mathbf{depth}(7) = 1$. Assume $t \geq 8$. If $i \leq p^{t-1}$, we have $S_i^t = S_i^{t-1}$ and $S_{i-1}^t = S_{i-1}^{t-1}$ so the result follows by induction on t. If $i > p^{t-1}$, then $S_i^t = \mathbf{densify}^{t-1}(S_{i-1}^t, b_{i-1}^t)$ where $\mathbf{weight}^{t-1}(S_{i-1}^t) \geq 8$ and $b_{i-1}^t \leq \lceil \mathbf{weight}^{t-1}(S_{i-1}^t)/8 \rceil < \mathbf{weight}^{t-1}(S_{i-1}^t)/2$. In this case Proposition 2 applies. □

For any $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}(t) - 1\}$, the difference $S_i^t \setminus S_{i+1}^t$ is a pair of segments denoted L_i^t (the portion of S_i^t to the left of S_{i+1}^t) and R_i^t (the portion to the right of S_{i+1}^t).

Lemma 6 *For any $t \in \{1, \ldots, n\}$ and any $i \in \{1, \ldots, \mathbf{depth}(t) - 1\}$,* $\mathbf{weight}^{t-1}(L_i^t) \geq b_i^t$ *and* $\mathbf{weight}^{t-1}(R_i^t) \geq b_i^t$.

Proof: We prove this by induction on t.

If $i > p^{t-1}$, then S_i^t is rebuilt at time t and by the *Rebuilding Rule*, L_i^t and R_i^t will each have weight at least b_i^t.

If $i \leq p^{t-1}$ then S_i^t and S_{i-1}^t are preserved at time t and so $L_i^t = L_i^{t-1}$ and $R_i^t = R_i^{t-1}$ are also unchanged, and since $i \leq p^{t-1}$, we have $B^{t-1} \subseteq S_i^{t-1} = S_i^t$ and consequently, applying the induction hypothesis we have $\mathbf{weight}^{t-1}(L_i^t) = \mathbf{weight}^{t-1}(L_i^{t-1}) \geq b_i^{t-1} = b_i^t$ and $\mathbf{weight}^{t-1}(R_i^t) = \mathbf{weight}^{t-1}(R_i^{t-1}) \geq b_i^{t-1} = b_i^t$. □

This lemma reflects a subtle point in the adversary. We defined $b_i^t = \lceil \mathbf{weight}^{t-1}(S_i^t)/8 \rceil$ only for $i > p^{t-1}$, while it might seem more natural to use this definition for all i. The given definition which sets $b_i^t = b_i^{t-1}$ for $i \leq p^{t-1}$ is needed for the induction step in the above proof.

Next we relate the cost of relabelings at step t to $(b_i^t : 1 \leq i \leq \mathbf{depth}(t))$:

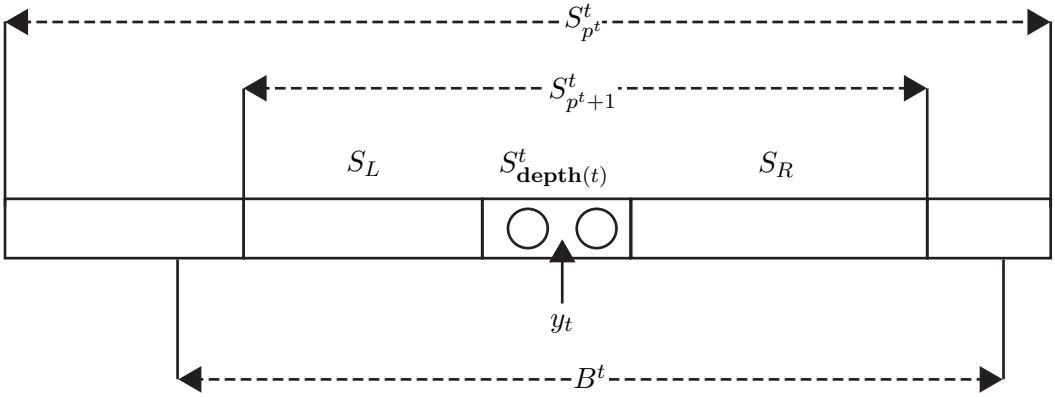

Figure 1: A typical position of S_L, S_R and B^t.

Lemma 7 *If $\mathcal{A}$ is lazy then for any $t \in \{1, \ldots, n\}$, $|Rel^t| \geq \sum_{i=p^t+1}^{\mathbf{depth}(t)} b_i^t$.*

Proof: We have $|Rel^t| \geq 1$. If $p^t = \mathbf{depth}(t)$ then the sum in the inequality evaluates to zero. If $p^t = \mathbf{depth}(t) - 1$ then the sum is just $b^t_{\mathbf{depth}(t)}$ which is 1, since by the termination condition in the *Rebuilding Rule*, $\mathbf{weight}^{t-1}(S^t_{\mathbf{depth}(t)}) \in \{2, \ldots, 7\}$, so $b^t_{\mathbf{depth}(t)} = \lceil \mathbf{weight}^{t-1}(S^t_{\mathbf{depth}(t)})/8 \rceil = 1$.

So assume $p^t < \mathbf{depth}(t) - 1$. First we note that $B^t \cap S^t_{\mathbf{depth}(t)} \neq \emptyset$. By the definition of the adversary, y^t is between two items u and v such that $f^{t-1}(u), f^{t-1}(v) \in S^t_{\mathbf{depth}(t)}$. If $f^t(y^t) \in S^t_{\mathbf{depth}(t)}$ then $B^t \cap S^t_{\mathbf{depth}(t)} \neq \emptyset$; otherwise at least one of u and v is relabeled at step t in which case $f^{t-1}(u) \in B^t$ or $f^{t-1}(v) \in B^t$ and again $B^t \cap S^t_{\mathbf{depth}(t)} \neq \emptyset$.

The set $S^t_{p^t+1} \setminus S^t_{\mathbf{depth}(t)}$ is the union of the left subsegment S_L and the right subsegment S_R (Fig. 1). Since B^t is a segment that has nonempty intersection with $S^t_{\mathbf{depth}(t)}$ and is not a subset of $S^t_{p^t+1}$ (by the definition of p^t), B^t contains at least one of S_L and S_R.

So assume, without loss of generality, that $S_L \subseteq B^t$. Since $\mathcal{A}$ is lazy, $|Rel^t| = 1 + |\mathbf{pop}^{t-1}(B^t)| \geq 1 + |\mathbf{pop}^{t-1}(S_L)|$. Now S_L is the disjoint union of the sets L_i^t defined prior to Lemma 6, for $i \in [1 + p^t, \mathbf{depth}(t) - 1]$. By Lemma 6, we have $|Rel^t| \geq 1 + \sum_{i=p^t+1}^{\mathbf{depth}(t)-1} b_i^t$. Since $b^t_{\mathbf{depth}(t)} = 1$ we obtain the inequality of the lemma. □

Next we bound b_i^t from below. For $t \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, \mathbf{depth}(t)\}$, let $\mathbf{start}(t, i)$ denote the largest $t' \leq t$ such that $p^{t'-1} < i$. Thus $S_i^{\mathbf{start}(t,i)}$ was rebuilt, and S_i^u was preserved for $u \in (\mathbf{start}(t, i), t]$. Similarly, let $\mathbf{end}(t, i)$ be the least $t' > t$ such that $p^{t'-1} < i$, if such a t' exists, and $n + 1$ otherwise. Thus, $\mathbf{end}(t, i)$ is the earliest time $t' > t$ such that $S_i^{t'}$ was rebuilt.

Lemma 8 *For $t \in \{7, \ldots, n\}$ and $i \in \{2, \ldots, \mathbf{depth}(t)\}$, $b_i^t \geq \frac{1}{64}(\mathbf{weight}^{t-1}(S^t_{i-1}) - \mathbf{weight}^{t-1}(S_i^t))$.*

Proof: For $t = 7$, $\mathbf{depth}(t) = 1$ and the lemma is true vacuously. Assume $t \geq 8$ and $i \in \{2, \ldots, \mathbf{depth}(t)\}$. The definitions of $\mathbf{start}(t, i)$ and $\mathbf{end}(t, i)$ imply $S_i^u = S_i^t$ for $u \in$

$\{\mathbf{start}(t,i),\ldots,\mathbf{end}(t,i)-1\}$. For any segment S containing S_i^t, and any s,s' satisfying $\mathbf{start}(t,i)\le s<s'<\mathbf{end}(t,i)$,

$$\mathbf{weight}^{s'-1}(S)-\mathbf{weight}^{s-1}(S)=s'-s,$$

since one item is added to S and none are removed at each step $u\in\{\mathbf{start}(t,i),\ldots,\mathbf{end}(t,i)-1\}$. Let $s=\mathbf{start}(t,i)$. Then $\mathbf{start}(t,i-1)\le s\le t<\mathbf{end}(t,i)\le\mathbf{end}(t,i-1)$ so

$$\begin{aligned}\mathbf{weight}^{t-1}(S_{i-1}^t)-\mathbf{weight}^{t-1}(S_i^t)&=\mathbf{weight}^{s-1}(S_{i-1}^t)-\mathbf{weight}^{s-1}(S_i^t)\\&=\mathbf{weight}^{s-1}(S_{i-1}^s)-\mathbf{weight}^{s-1}(S_i^s).\end{aligned}$$

Also

$$b_i^t=b_i^s=\lceil\mathbf{weight}^{s-1}(S_i^s)/8\rceil\ge\mathbf{weight}^{s-1}(S_i^s)/8.$$

Since $8\le\mathbf{weight}^{s-1}(S_{i-1}^t)$ and $\mathbf{weight}^{\mathbf{start}(t,i-1)-1}(S_{i-1}^t)\le\mathbf{weight}^{s-1}(S_{i-1}^t)$,

$$\begin{aligned}b_{i-1}^s=b_{i-1}^{\mathbf{start}(t,i-1)}&=\lceil\mathbf{weight}^{\mathbf{start}(t,i-1)-1}(S_{i-1}^t)/8\rceil\\&\le\mathbf{weight}^{s-1}(S_{i-1}^s)/4\end{aligned}$$

Hence,

$$\begin{aligned}\mathbf{weight}^{s-1}(S_i^s)=\lfloor(\mathbf{weight}^{s-1}(S_{i-1}^s)-2b_{i-1}^s)/2\rfloor&\ge\lfloor\mathbf{weight}^{s-1}(S_{i-1}^s)/4\rfloor\\&\ge\mathbf{weight}^{s-1}(S_{i-1}^s)/8.\end{aligned}$$

Thus, $b_i^t\ge\mathbf{weight}^{s-1}(S_{i-1}^s)/64\ge(\mathbf{weight}^{s-1}(S_{i-1}^s)-\mathbf{weight}^{s-1}(S_i^s))/64$. The claim follows. □

Corollary 9 *For any $t\in\{1,\ldots,n\}$ and $j\in\{1,\ldots,\mathbf{depth}(t)-1\}$,*

$$\sum_{i=j+1}^{\mathbf{depth}(t)}b_i^t\ge\frac{1}{64}\cdot\mathbf{weight}^{t-1}(S_j^t)-\frac{1}{8}.$$

Proof: For fixed t, sum the inequality in Lemma 8 for i from $j+1$ to $\mathbf{depth}(t)$, and note that $\mathbf{weight}^{t-1}(S^t_{\mathbf{depth}(t)})\le 8$. Then divide through by 64. □

We now come to the main lower bound of this section.

Corollary 10 *Let $\mathcal{A}$ be a lazy algorithm. Then*

$$\chi_{\mathcal{A}}(y^1,y^2,\ldots,y^n)\ge\frac{1}{64}\sum_{t=1}^{n}\mathbf{weight}^{t-1}(S_{p^t}^t)-\frac{n}{8}.$$

Proof: $\chi_{\mathcal{A}}(y^1,\ldots,y^n)$ is at least $\sum_{t=1}^n|Rel^t|$. Now combine Lemmas 7 and Corollary 9. (For t such that $p^t=\mathbf{depth}(t)$ use the fact that $|Rel^t|\ge 1>\frac{1}{64}\cdot\mathbf{weight}^{t-1}(S_{p^t}^t)-\frac{1}{8}$.) □

3 Prefix Bucketing

Prefix bucketing is a one player game that models the behavior of an algorithm against our adversary.

A *bucket configuration* for k buckets and t items is a sequence $a_1, \ldots, a_k$ of nonnegative integers summing to t. One should think of a_i as the number of items in bucket i. Given a bucket configuration $a_1, \ldots, a_k$, *placing a new item in bucket* p transforms the configuration as follows: A new item is added to bucket p and all items in buckets higher than p are moved to bucket p. Buckets $i < p$ are unchanged. Formally the configuration b produced from a by the placement p satisfies:

- $b_i = a_i$ for $i < p$,
- $b_p = 1 + \sum_{i \geq p} a_i$, and
- $b_i = 0$ for $i > p$.

The *cost* of this placement is the number of items b_p in bucket p after the placement.

A sequence $p^1, \ldots, p^n$ where each $p^i \in \{1, \ldots, k\}$ is called a *placement sequence*, and corresponds to placing a sequence of n items in the buckets. A placement sequence induces a sequence of configurations $a^0, a^1, \ldots, a^n$ where a^0 has no items, and a^t is obtained from a^{t-1} by placing p^t and applying the above transformation. The sequence $a^0, \ldots, a^n$ is called a *prefix bucketing*. The *cost* $c(a^0, \ldots, a^n) = \sum_{t=1}^{n} a^t_{p^t}$ of $a^0, a^1, \ldots, a^n$ is the sum of the individual placement costs.

3.1 Connecting bucketing to online labeling

Now we show that for any lazy online labeling algorithm $\mathcal{A}$, the adversary defined in the previous section can be associated to a prefix bucketing whose cost provides a lower bound on the cost of $\mathcal{A}$ in labeling the sequence $Y = \{y_1, \ldots, y_n\}$ produced by the adversary.

Fix a lazy online labeling algorithm $\mathcal{A}$ and for $1 \leq t \leq n$, let $f^t, S^t_i, B^t, p^t, y^t$ and f^0, p^0 be as defined by the **Adversary**$(\mathcal{A}, n, m)$. Let Y denote the set $\{y^1, y^2, \ldots, y^n\}$. Recall that our adversary specifies a sequence of critical levels for steps 7 to $n-1$.

Set $k = \lceil \log m \rceil$. For $t = 0, 1, \ldots, n$ we define a sequence $(A^t_i : 1 \leq i \leq k)$ of subsets of Y as follows: For $t \in \{0, \ldots, 7\}$, $A^t_1 = \{y^1, \ldots, y^t\}$ and, for $i \in \{2 \ldots, k\}$, $A^t_i = \emptyset$. For $t \geq 8$

- $A^t_i = A^{t-1}_i$, for $i \in \{1, \ldots, p^t - 1\}$,
- $A^t_{p^t} = \{y^t\} \cup \bigcup_{i \geq p^t} A^{t-1}_i$, and
- $A^t_i = \emptyset$, for $i \in \{p^t + 1, \ldots, k\}$.

For each $t \in \{0, \ldots, n\}$, let $a^t = a^t_1, \ldots, a^t_k$ be the bucket configuration defined by $a^t_i = |A^t_i|$. It is easy to see (by induction on n) that $a^0, \ldots, a^n$ is a prefix bucketing of n items into k buckets with placement sequence $p^1, \ldots, p^n$ (where we define $p^1 = \cdots = p^7 = 1$ since the adversary does not specify critical levels for $t \leq 7$). The cost of this bucketing is $c(a^0, a^1, \ldots, a^n) = \sum_{i=1}^{n} |A^t_{p^t}|$. We now relate this cost to the cost of online labeling. We start by noting:

Lemma 11 *For any $t \in \{1,\ldots,n\}$ and $i \in \{1,\ldots,\mathbf{depth}(t)\}$, $f^{t-1}(A_i^t \setminus \{y^t\}) \subseteq S_i^t$, and therefore* $\mathbf{weight}^{t-1}(S_i^t) \geq |A_i^t| - 1$.

Proof: We prove the claim by induction on t. For $t = 1$, the only non-empty set is $A_1^1 = \{y^1\}$ so the claim is true. Assume $t \geq 2$. The $\mathbf{Adversary}(\mathcal{A}, n, m)$ chooses the sets S_i^t and the item y^t, the algorithm $\mathcal{A}$ outputs $f^t$and then the adversary defines B^t, Rel^t and p^t. We distinguish two cases.

Case $p^{t-1} \leq p^t$: Suppose first that $i < p^{t-1}$. Since $i \leq p^{t-1} < p^t$, $A_i^t = A_i^{t-1}$. We have $B^{t-1} \subseteq S_{p^{t-1}}^{t-1} \subseteq S_i^{t-1} = S_i^t$, where the first containment follows from the definition of p^{t-1} and the last equality follows from the definition of S_i^t. Since $y^t \notin A_i^{t-1} = A_i^t$:

$$f^{t-1}(A_i^t \setminus \{y^t\}) = f^{t-1}(A_i^t) \subseteq f^{t-2}(A_i^t \setminus \{y^t\}) \cup B^{t-1} = f^{t-2}(A_i^{t-1} \setminus \{y^t\}) \cup B^{t-1} \subseteq S_i^{t-1} = S_i^t,$$

where the first containment follows from the fact that for any $y \in \{y^1,\ldots,y^{t-2}\}$ either $f^{t-1}(y) = f^{t-2}(y)$ or $f^{t-1}(y) \in B^{t-1}$, and the second containment follows from the induction hypothesis.

Next suppose that $i > p^{t-1}$. Then $A_i^{t-1} = \emptyset$. Hence, $A_{p^t}^t = \{y^t\}$, and if $i \neq p^t$ then $A_i^t = \emptyset$. In either case the lemma follows trivially.

Case $p^{t-1} \geq p^t$: For $j < p^t$, $A_j^t = A_j^{t-1}$, for $j = p^t$, $A_i^t = \{y^t\} \cup \bigcup_{j \geq p^t} A_j^{t-1}$, and for $j > p^t$, $A_j^t = \emptyset$. Again, for all $i \leq p^t$, $B^{t-1} \subseteq S_{p^{t-1}}^{t-1} \subseteq S_i^{t-1} = S_i^t$. For all $i < p^t$, $y^{t-1} \notin A_i^{t-1} = A_i^t$ so using the induction hypothesis $f^{t-1}(A_i^t) \subseteq f^{t-2}(A_i^t) \cup B^{t-1} \subseteq S_i^{t-1} = S_i^t$. It only remains to consider the case of $i = p^t$ as the claim is trivial for $i > p^t$. Since $p^{t-1} \geq p^t$, for $i > p^t$, $S_i^{t-1} \subseteq S_{p^t}^{t-1} = S_{p^t}^t$. Using the induction hypothesis:

$$f^{t-1}(A_{p^t}^t \setminus \{y^t\}) \subseteq f^{t-1}(\bigcup_{j \geq p^t} A_j^{t-1}) \subseteq \bigcup_{j \geq p^t} f^{t-2}(A_j^{t-1} \setminus \{y^{t-1}\}) \cup B^{t-1} \subseteq \bigcup_{j \geq p^t} S_j^{t-1} \cup B^{t-1} \subseteq S_{p^t}^t.$$

□

Using this lemma, we can replace $\mathbf{weight}^{t-1}(S_{p^t}^t)$ by $|A_{p^t}^t| - 1$ in Corollary 10 to obtain the following connection between the cost of online labeling and prefix bucketing.

Lemma 12 *Let $\mathcal{A}$ be an algorithm and let $y^1,\ldots,y^n$ be the sequence produced by adversary $\mathbf{Adversary}(\mathcal{A}, n, m)$. Let the prefix bucketing $a^0, a^1, \ldots, a^n$ be defined by $a_i^t = |A_i^t|$, for all $t = 0,\ldots,n$ and $i = 1,\ldots,k$ with $k = \lceil \log m \rceil$, as described above. Then:*

$$\chi_{\mathcal{A}}(y^1, y^2, \ldots, y^n) \geq \frac{1}{64} \cdot c(a^0, a^1, \ldots, a^n) - \frac{9}{64}n.$$

3.2 Lower Bound for Bucketing

In this subsection we derive a lower bound (Lemma 25) on the cost of any prefix bucketing. To do so we map any prefix bucketing to a k-tuple of ordered rooted trees. We prove a lower bound on the sum of the depths of the nodes of the trees, and this will imply a lower bound on the cost of the bucketing.

Ordered trees An *ordered rooted tree* is a rooted tree where the children of each node are ordered from left to right. Since these are the only trees we consider, we refer to them simply as trees. The *i-th subtree of T* is the tree rooted at the i-th child of the root from the left. If the root has less than i children, we consider the i-th subtree to be empty. The *size* of T, denoted $|T|$ is the number of nodes. The *depth* of a node is one more than its distance to the root, e.g., the root has depth 1. The depth of a tree is the maximum depth of its nodes. The *cost* of T, denoted $\kappa(T)$, is the sum of the depths of its nodes. The cost and size of an empty tree is 0.

To each prefix bucketing $\bar{a} = a^0, a^1, \ldots, a^n$ into k buckets, we associate a k-tuple of trees $T(\bar{a})_1, T(\bar{a})_2, \ldots, T(\bar{a})_k$ inductively as follows: The trivial bucketing $\bar{a} = a^0$ is mapped to the k-tuple of empty trees. For bucketing $\bar{a} = a^0, a^1, \ldots, a^n$ with placement sequence $p^1, \ldots, p^n$ let $\bar{a}'$ be the bucketing $a^0, a^1, \ldots, a^{n-1}$, and assume $T(\bar{a}')$ has been defined. We define $T(\bar{a})$ by:

- For $1 \leq i < p^n$, $T(\bar{a})_i = T(\bar{a}')_i$.
- $T(\bar{a})_{p^n}$ consists of a root node whose children are the non-empty trees among $T(\bar{a}')_{p^n}, T(\bar{a}')_{p^n+1}, \ldots, T(\bar{a}')_k$ ordered left to right by increasing index.
- $T(\bar{a})_i$ is an empty tree for $p^n < i \leq k$.

A straightforward induction on n yields:

Proposition 13 *For any positive integer k, if $\bar{a} = a^0, a^1, \ldots, a^n$ is a prefix bucketing into k buckets then for each $i \in \{1, \ldots, k\}$, $|T(\bar{a})_i| = a_i^n$.*

The next lemma relates the cost of bucketing to the cost of its associated trees.

Lemma 14 *For any positive integer k, if $\bar{a} = a^0, a^1, \ldots, a^n$ is a prefix bucketing into k buckets then $c(\bar{a}) = \sum_{i=1}^{k} \kappa(T(\bar{a})_i)$.*

Proof: By induction on n. For $n = 0$, both sides of the equality are 0. Suppose $n \geq 1$ and assume that the claim is true for $n-1$. Let $\bar{a}' = a^0, a^1, \ldots, a^{n-1}$ and p^n be as in the definition of prefix bucketing.

$$\begin{aligned} c(\bar{a}) = c(\bar{a}') + 1 + \sum_{i=p^n}^{k} a_i^{n-1} &= \sum_{i=1}^{k} \kappa(T(\bar{a}')_i) + 1 + \sum_{i=p^n}^{k} |T(\bar{a}')_i| \\ &= \sum_{i=1}^{p^n-1} \kappa(T(\bar{a})_i) + 1 + \sum_{i=p^n}^{k} \left(\kappa(T(\bar{a}')_i) + |T(\bar{a}')_i|\right) \end{aligned}$$

where the second equality uses the induction hypothesis with Proposition 13, and the last equality follows from the definition of $T(\bar{a})_i$ for $i = 1, \ldots, p^n - 1$. For $i \geq p^n$ the depth of each node in $T(\bar{a}')_i$ increases by one when it becomes a child of $T(\bar{a})_{p^n}$, hence

$$\kappa(T(\bar{a})_{p^n}) = 1 + \sum_{i=p^n}^{k} \left(\kappa(T(\bar{a}')_i) + |T(\bar{a}')_i|\right).$$

For $i > p^n$, $\kappa(T(\bar{a})_i) = 0$ so the lemma follows. $\square$

Thus to get a lower bound on the cost of a prefix bucketing it suffices to prove a lower bound on the sum of the costs of the trees that occur in the associated k-tuple. The following definition will help describe the structure of trees that occur in such a k-tuple.

Definition 15 (k-admissible) *Let k be a positive integer. The empty tree is k-admissible. A non-empty tree T is k-admissible if its root has at most k children and the i-th subtree of T is $(k+1-i)$-admissible.*

For example, T is 1-admissible if and only if T is empty or a rooted path. We collect some basic properties of k-admissibility.

Proposition 16 *Let T be a (rooted ordered) tree and $k \geq 1$, and suppose T is k-admissible. Let v be a leaf of T.*

1. *If $k' > k$ then T is k'-admissible.*
2. *If v is deleted from T then the resulting tree is k-admissible.*
3. *If a new node is added as a child of v then the resulting tree is k-admissible.*
4. *If T has at least two nodes and $k \geq 2$, then the tree obtained from T by removing its first subtree is $(k-1)$-admissible.*

Proof: The first and last parts are immediate from the definition of k-admissibility. We prove the other two parts by induction on $|T|$. Let T' be the tree resulting from deleting v and T'' be the tree resulting from adding a child to v. If $|T| = 1$ then T, T' and T'' are k-admissible for all $k \geq 1$. Suppose $|T| > 1$. Let v belong to the i-th subtree T_i of T. By definition of k-admissible, T_i is $(k-i+1)$-admissible, and by induction the corresponding subtree T_i'' is also $(k-i+1)$-admissible. It follows immediately that T'' is k-admissible.

To show that T' is k-admissible, we split into cases according to whether $|T_i| > 1$ or $|T_i| = 1$. If $|T_i| > 1$ let T_i' be obtained by deleting v. By induction T_i' is still $(k-i+1)$ admissible, and every other subtree is unchanged so T' is still k-admissible. If $|T_i| = 1$ then v is a child of the root and removing it eliminates the ith subtree. Thus for any $j \geq i$, the j-th subtree T_j of T (which must be $(k-j+1)$-admissible) becomes the $(j-1)$-st subtree of T' (and is $(k-(j-1)+1)$-admissible by the first part of the proposition) and so T' is k-admissible. $\square$

The connection of admissibility to prefix bucketing is given by the following proposition:

Proposition 17 *For any positive integer k, if $\bar{a} = a^0, a^1, \ldots, a^n$ is a prefix bucketing into k buckets then for each $i \in \{1, \ldots, k\}$, $T(\bar{a})_i$ is $(k+1-i)$-admissible.*

Proof: We proceed by induction on n. The result is immediate for $n = 0$. Assume $n \geq 1$ and let $\bar{a}' = a^0, \ldots, a^{n-1}$. By induction $T(\bar{a}')_i$ is $(k+1-i)$-admissible. By the definition of $T(\bar{a})$, for $i < p_n$, $T(\bar{a})_i = T(\bar{a}')_i$ and is $k+1-i$-admissible by induction. We need to show that $T(\bar{a})_{p^n}$ is $(k+1-p^n)$-admissible. Its subtrees are $T(\bar{a}_i')$ for $p^n \leq i \leq k$ (some of which

may be empty) so it has at most $k+1-p^n$ children. We also need that its ith nonempty subtree is $(k+1-p^n+1-i)$-admissible. Its ith nonempty subtree is equal to $T(\bar{a}'_j)$ for some $j \geq p^n+i-1$. By induction the subtree is $(k+1-j)$ admissible and therefore (by the first part of Proposition 16) is $(k+1-p^n+1-i)$-admissible. □

Let us define $\mu(n,k)$ to be the minimum cost of a k-admissible tree of n vertices.

Proposition 18 *For any bucketing $\bar{a}$ of n items into k buckets, we have $c(\bar{a}) \geq \mu(n,k)-n+1$.*

Proof: Modify $\bar{a}$ to the bucketing $\bar{b} = b^0, \ldots, b^n$ where $b^i = a^i$ for $i < n$ and $b^n = (n, 0, \ldots, 0)$. This corresponds to placing the final item in bucket 1. This can increase the cost by at most $n-1$ so $c(\bar{a}) \geq c(\bar{b}) - n + 1$. The first tree U in the k-tuple $T(\bar{b})$ has size n and is k-admissible by Proposition 17. By Lemma 14, $c(\bar{b}) \geq \kappa(U)$ which is at least $\mu(n,k)$. □

It remains to give a lower bound on $\mu(n,k)$.

Lemma 19 *Let d be a positive integer and T be an arbitrary rooted tree with all leaves of depth at least d. Then $\kappa(T) \geq \frac{(d+1)}{2} \cdot |T|$.*

Proof: Let n_i denote the number of nodes of T at the depth $i = 1, \ldots, d-1$, and let n_d denote the number of nodes at depth at least d. $|T| = n_1 + n_2 + \cdots + n_d$. By the hypothesis on T (that every leaf has depth at least d), each node of T at depth $i < d$ has at least one child at depth $i+1$. Therefore $0 \leq n_1 \leq n_2 \leq \cdots \leq n_d$. Clearly, $c(T) \geq \sum_{i=1}^{d} i n_i$. Given the constraints, this sum is minimized (over reals) when $n_1 = n_2 = \cdots = n_d = |T|/d$. Thus $c(T) \geq \frac{d(d+1)}{2} \cdot \frac{|T|}{d}$. □

A k-admissible tree may have leaves of low depth, so the above lemma is not immediately useful. We need the following:

Definition 20 (Balanced tree) *A tree of depth d is balanced if all its leaves are of depth d or $d-1$.*

Lemma 21 *Let T be a k-admissible tree of size n having cost $\mu(n,k)$. Then T is balanced.*

Proof: Let d be the depth of T. Suppose for contradiction that T is unbalanced. Let u be a leaf of depth at most $d-2$ and let v be a leaf of depth d. Let T' be the tree obtained by removing v and reattaching v as a child of u. Then $\kappa(T') < \kappa(T)$, and by the second and third parts of Proposition 16, T' is k-admissible. This contradicts the assumption that T has minimum cost among k-admissible trees. Thus T must be balanced. □

Lemma 22 *Let $k, d \geq 1$. If T is a k-admissible tree of depth d, then $|T| \leq \binom{k+d-1}{k}$.*

Proof: If $k = 1$ then T must be a rooted path of depth d and clearly $|T| = d \leq \binom{d}{1}$. For $k \geq 2$, we prove the result by induction on $|T|$. If $|T| = 1$ the result is trivial so assume $|T| \geq 2$. Let L be the first subtree of T and let R be the tree created by removing L from T. By the definition of k-admissibility L is a k-admissible tree of depth at most $d-1$, and by the last part of Proposition 16, R is a $(k-1)$-admissible tree of depth at most d. By the induction hypothesis, $|T| = |L| + |R| \leq \binom{k+d-2}{k} + \binom{(k-1)+(d-1)}{k-1} = \binom{k+d-1}{k}$. □

Corollary 23 *For any $n \geq k \geq 1$, $\mu(n,k)$ is at least $n(w+1)/2$ where w is the smallest integer such that $\binom{k+w}{k} \geq n$.*

Proof: Let T be a k-admissible tree of size n having minimum cost. Let d be the depth of T. By Lemma 22 we have $\binom{k+d-1}{k} \geq n$ and so $d-1 \geq w$. By Lemma 21, T is balanced so all leaves are at depth at least $d-1 \geq w$ and so by Lemma 19, $\kappa(T) \geq (w+1)n/2$. □

Lemma 24 *Let n, k, w be integers such that $n \geq 2$ and $k \geq \log n$. If $\binom{k+w}{k} \geq n$, then $w \geq \frac{\log n}{4(\log 8k - \log\log n)}$.*

Proof: If $w \geq k$ then the conclusion holds, so assume $w \leq k$. Recall that $\log \binom{r}{s} \leq H(s/r)r$ where H stands for the binary entropy function defined on $[0,1]$ by $H(x) = x\log\frac{1}{x} + (1-x)\log\frac{1}{1-x}$, where the base of the logarithm is 2. For $x \in (0, 1/2]$, $H(x) \leq 2x\log\frac{1}{x}$. Therefore:

$$\log n \leq \log\binom{k+w}{k} = \log\binom{k+w}{w} \leq (k+w)H\left(\frac{w}{k+w}\right) \leq 2w\log\left(\frac{k+w}{w}\right) \leq 2w\log\left(\frac{2k}{w}\right).$$

Defining $a = \frac{1}{2}\log n$, we get $a \leq 2\log(2ak/\log(n)) = 2(\log(8k) + \log(a/4) - \log\log n)$ and using $\log x < x$ we get $a \leq 2(\log(8k) + a/4 - \log\log n)$ and so $a/2 \leq 2(\log(8k) - \log\log n)$, which implies the claimed bound on w. □

We now deduce a lower bound on the cost of prefix bucketing:

Lemma 25 *Let k, n be positive integers such that $k \geq \log n$. The cost of any prefix bucketing of n items into k buckets is at least $\frac{n\log n}{8(\log 8k - \log\log n)} - n$.*

Proof: By Proposition 18, the cost of any such bucketing is at least $\mu(n,k) - n$, and by Corollary 23, this is at least $\frac{nw}{2} - n$ where w is the smallest integer such that $\binom{k+w}{k} \geq n$, which is at least $\log n/4(\log 8k - \log\log n)$ by Lemma 24. □

3.3 Proof of Lemma 4

Finally we return to the online labeling problem and prove the main lemma: Fix a lazy online labeling algorithm $\mathcal{A}$ and let $n \leq m$ be positive integers as in the hypothesis. Lemma 12 implies that for the adversary sequence $y^1, \ldots, y^n$ constructed from $\mathcal{A}$, the cost $\chi_{\mathcal{A}}(y^1, \ldots, y^n)$ incurred by the algorithm is at least $\frac{1}{64}M(n,k) - \frac{9}{64}n$, where $M(n,k)$ is the minimum cost of any prefix bucketing of n items into $k = \lceil \log m \rceil$ buckets. Substituting the lower bound on $M(n,k)$ given by the previous lemma, we get that the worst case cost of $\mathcal{A}$ satisfies:

$$\chi_{\mathcal{A}}(y^1, y^2, \ldots, y^n) \geq \frac{1}{512} \cdot \frac{n\log n}{3 + \log\lceil\log m\rceil - \log\log n} - \frac{n}{6}.$$

4 An Upper Bound for large array size

The lower bound of Theorem 1 generalizes the $\Omega(n \log n)$ lower bound for $m = n^{\Theta(1)}$ of [12] to $\Omega(n \log n \frac{1}{\log\log m - \log\log n})$ for m up through $m < 2^n$. When $m = n^{\Theta(1)}$ the upper and lower bounds match at $\Theta(n \log n)$. On the other hand, when $m \geq 2^n$, n items can be inserted without every moving a placed item (and thus the cost is n): when the jth item arrives, store it in location $L + 2^{n-j}$ where L is the location of the largest item less than it (or is 0 if there is no such item.)

So the lower bound of Theorem 1 is tight at the extremes $m = n^{\Theta(1)}$ and $m = 2^n$. What happens for intermediate values of m? In this section we show that the lower bound of Theorem 1 is tight for all m at least some quasi-polynomial function of n.

For a real number $m \geq 2$ and for positive integer k, define:

$$r(m) = \lceil \sqrt{\log(m/2)} \rceil \text{ and } b_k(m) = \binom{r(m)}{k}.$$

Let $n_k(m)$ be the maximum number of items that can be inserted into an array of size m such that the cost does not exceed kn. We prove:

Theorem 26 *For positive integers m and k, we have $n_k(m) \geq b_k(m)$.*

As an immediate consequence of Theorems 1 and 26 we get:

Corollary 27 *For m and n satisfying $2^{1+\log^3 n} < m \leq 2^n$, $\chi_m(n) = \Theta(n \log n \frac{1}{\log\log m})$.*

Proof: The lower bound follows from Theorem 1 using $\log\log m - \log\log n \geq \frac{2}{3}\log\log m$ for $m \geq 2^{\log^3 n}$. For the upper bound, since $\binom{a}{k} \geq (a/k)^k$ for $k \in [1, a]$, if $k \leq r(m)^{1/3}$ then $b_k(m) \geq (\frac{r(m)}{k})^k \geq r(m)^{2k/3} \geq (\log(m/2))^{k/6}$. Given n and $m \geq 2^{1+(\log^3 n)}$, let $k = 6\lceil \log n / \log\log(\frac{m}{2}) \rceil$. Then $n \leq (\log(m/2))^{k/6} \leq b_k(m)$ and so $\chi_m(n) \leq kn = 6n\lceil \log n / \log\log(\frac{m}{2}) \rceil$. □

The exponent 3 in the hypothesis $m \geq 1 + 2^{\log^3 n}$ is not crucial; a more careful argument can reduce it to any number greater than 2, and perhaps even further.

The proof of Theorem 26 will need a few elementary facts about $r(m)$ and $b_k(m)$.

Proposition 28 *1. For $m \geq 2$ and integer $j \in [2, r(m)/2]$, $b_j(m) \leq 2^{2r(m)-3} - 1$.*

2. $r(\frac{m}{w}) \geq r(m) - 1$ provided that $m \geq 2$ and $1 \leq w \leq 2^{2r(m)-3}$.

Proof: For the first part, $2 \leq j \leq r(m)/2$ implies $r(m) \geq 4$ and thus $r(m) \leq 2r(m) - 4$ and so $b_j(m) \leq \binom{r(m)}{j} \leq 2^{r(m)} \leq 2^{2r(m)-4} \leq 2^{2r(m)-3} - 1$.

For the second part, we want $\sqrt{\log(m/2) - \log(w)} \geq \sqrt{\log(m/2)} - 1$. Squaring both sides, it suffices to show $\log(w) \leq 2\sqrt{\log(m/2)} - 1$, which is true by the assumption $\log w \leq 2r(m) - 3 \leq 2\sqrt{\log(m/2)} - 1$. □

Proof of Theorem 26: We proceed by induction on k. We assume (without loss of generality) that throughout the algorithm cells 1 and m are occupied by items $y_{\min}$ and $y_{\max}$ which are, respectively, lower and upper bounds on all items.

At any time certain array cells are occupied. A segment of cells whose only occupied cells are the leftmost and rightmost cell is an *open segment*; the items stored in these cells are denoted $y_L(S)$ and $y_R(S)$. The initial open segment has size m. An open segment S is *usable* if $|S| \geq 3$ (so it has at least one unoccupied cell). For an item y not stored in the array, there is a unique open segment S such that $y_L(S) < y < y_R(S)$; we say that S is *compatible with* y. Storing y in an unoccupied cell $c \in S$ splits S into two open segments that overlap at c and and c can be chosen to be a *middle cell* for which both new segments have size at least $|S|/2$.

For each $k \geq 1$ we define algorithm A_k whose cost per item inserted is at most k. We'll show that A_k can insert $b_k(m)$ items when run on an array of size m

Algorithm A_1 is: While no two adjacent cells are occupied, insert the next item y into the middle cell of the open segment S compatible with y. A simple induction shows that after inserting t items, each open segment has length at least $m/2^t$. Since A_1 can continue as long as this exceeds 2, it can do at least $\lceil \log m/2 \rceil \geq b_k(m) = \lceil \sqrt{\log(m/2)} \rceil$ insertions.

For $k > 1$ we define A_k. If $m = 2$ $b_k(2) = 0$ items are inserted, so assume $m \geq 3$. If $k > r(m)/2$ then let $k' = \lfloor r(m) \rfloor /2$ and run $A_{k'}$. By induction, $b_{k'}(m) \geq b_k(m)$ items can be inserted. For $m \geq 3$, and $2 \leq k < r(m)/2$, A_k consists of two *phases*. In phase 1, A_{k-1} is used to insert $b_{k-1}(m)$ items. We then redistribute the items evenly so that each of the $w = 1 + b_{k-1}(m)$ segments has size at least $\lceil m/w \rceil$. We call the segments defined by these items the *phase 1 segments*. The total cost per item inserted in phase 1 is at most k. In phase 2, items placed during phase 1 do not move. We run w separate instances of A_k, one for each phase 1 segment. Each arriving item is assigned to the phase 1 segment that is compatible with it, and is inserted into that segment using A_k. Each segment has length at least $\lceil m/w \rceil$ and so we can handle $b_k(m/w)$ insertions even if all items in phase 2 are assigned to the same phase 1 segment. The total cost per item in phase 2 is at most k.

Thus we can handle $b_{k-1}(m) + b_k(m/w)$ insertions in the two phases, where $w = 1 + b_{k-1}(m)$. By part 1 of Proposition 28, $w \leq 2^{2r-3}$ and by part 2, $r(m/w) \geq r(m) - 1$ and so $b_k(m/w) \geq \binom{r(m)-1}{k}$. Thus at least $\binom{r(m)}{k-1} + \binom{r(m)-1}{k} \geq \binom{r(m)}{k} = b_k(m)$ items are inserted. □

References

[1] Martin Babka, Jan Bulánek, Vladimír Cunát, Michal Koucký, Michael Saks. On Online Labeling with Polynomially Many Labels. ESA 2012: 121-132.

[2] Martin Babka, Jan Bulánek, Vladimír Čunát, Michal Koucký, and Michael Saks. On Online Labeling with Superlinearly Many Labels. Manuscript 2012.

[3] Jan Bulánek, Michal Koucký, and Michael Saks. Tight lower bounds for the online labeling problem. In *Proc. of 66th Symp. of Theory of Computation, (STOC'12)*, Howard J. Karloff and Toniann Pitassi, editors, pages 1185–1198. ACM, 2012.

[4] Jan Bulánek, Michal Koucký, and Michael Saks. Tight lower bounds for the online labeling problem. SIAM J. on Computing, to appear.

[5] Jan Bulánek, Michal Koucký, and Michael Saks. On Randomized Online Labeling with Polynomially Many Labels. In *Proc. of 40th ICALP*, Lecture Notes in Computer Science 7965, pages 291-302, Springer, 2013.

[6] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proc. of 10th Annual European Symposium on Algorithms, (ESA)*, Rolf H. Möhring and Rajeev Raman, editors, volume 2461 of *LNCS*, pages 152–164. Springer, 2002.

[7] Michael A. Bender, Eik D. Demaine, and Martin Farach-Colton. Cache-oblivious B-trees. *Journal on Computing*, 35(2):341–358, 2005.

[8] Michael A. Bender, Ziyang Duan, John Iacono, and Jing Wu. A locality-preserving cache-oblivious dynamic dictionary. *Journal of Algorithms*, 53(2):115–136, 2004.

[9] Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache oblivious search trees via binary trees of small height. In *Proc. of 13th ACM-SIAM Symp. on Discrete Algorithms, (SODA)*, D. Eppstein, editor, pages 39–48. ACM/SIAM, 2002.

[10] Richard S. Bird and Stefan Sadnicki. Minimal on-line labelling. *Information Processing Letters*, 101(1):41–45, 2007.

[11] Thomas M. Cover, Joy A. Thomas. *Elements of Information Theory.* New York: Wiley, 1991.

[12] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. A tight lower bound for online monotonic list labeling. *SIAM J. Discrete Mathematics*, 18(3):626–637, 2004.

[13] Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. *Lower bounds for smooth list labeling.* Manuscript, 2005. (Listed in the references of [12]).

[14] Paul F. Dietz and Ju Zhang. Lower bounds for monotonic list labeling. In *SWAT*, pages 173–180, 1990.

[15] Yuval Emek and Amos Korman. New bounds for the controller problem. *Distributed Computing*, 24(3-4):177–186, 2011.

[16] Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In *Proc. of 8th International Colloquium on Automata, Languages and Programming, (ICALP'81)* Shimon Even and Oded Kariv, editors, volume 115 of *LNCS*, pages 417–431. Springer, 1981.

[17] Tsvi Kopelowitz. On-Line Indexing for General Alphabets via Predecessor Queries on Subsets of an Ordered List. In *Proceedings 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 283–292, 2012.

[18] Dan E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Information and Computation*, 97(2):150–204, 1992.

[19] Ju Zhang. Density Control and On-Line Labeling Problems. *PhD thesis*, University of Rochester, 1993.