SEARCHING, SORTING AND INFORMATION THEORY

Kurt Mehlhorn
Fachbereich 10 - Angewandte
Mathematik und Informatik
Universität des Saarlandes
6600 Saarbrücken, BRD

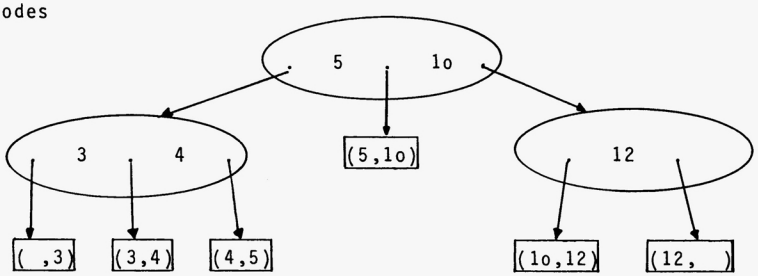
I. Introduction

We consider the relationship between searching, sorting and information theory. In the first section algorithms for the construction of optimal and nearly optimal search trees and a-priori bounds for the cost of such search trees are presented. Many of these results become readily available if search trees are interpreted as alphabetic prefix codes. Next TRIES and dynamic search trees are briefly discussed. In the last section the results are applied to sorting. Recent developments on sorting presorted files are described.

II. Searching and Information Theory

II.1 Search Trees and Alphabetic Codes

Consider the following ternary search tree. It has 3 internal nodes



and 6 leaves. The internal nodes contain the keys {3,4,5,10,12} in sorted order and the leaves represent the open intervals between keys.

The standard strategy to locate X in this tree is best described by the following recursive procedure SEARCH

```
proc SEARCH (int X ; node v)
if v is a leaf
then "X is not in the tree"
else begin let K1,K2 be the keys in node v;
          if X < K1 then SEARCH (X, left son of v)
          if X = K1 then exit (found);
          if K2 does not exist
          then SEARCH (X, right son of v)
          else begin if X < K2 then SEARCH (X, middle son of v);
          if X = K2 then exit (found);
          SEARCH (X, right son of v)
          end
     end
end
```

Apparently, the search strategy is unsymmetric. It is cheaper to follow the pointer to the first subtree than to follow the pointer to the second subtree and it is cheaper to locate K_1 than to locate K_2.

We will also assume that the probability of access is given for each key and each interval between keys. More precisely, suppose we have n keys $B_1,\ldots,B_n$ out of an ordered universe with $B_1 < B_2 <\ldots< B_n$. Then β_i denotes the probability of accessing B_i, $1 \leq i \leq n$, and α_j denotes the probability of accessing elements X with $B_j < X < B_{j+1}$, $0 \leq j \leq n$. α_0 and β_n have obvious interpretations. In our example $n = 5$, β_2 is the probability of accessing 4 and α_4 is the probability of accessing $X \in (4,5)$. We will always write the distribution of access probabilities as $\alpha_0,\beta_1,\alpha_1,\ldots,\beta_n,\alpha_n$.

Ternary trees, in general (t+1)-ary trees, correspond to prefix codes in a natural way. We are given letters $a_0,a_1,a_2,\ldots,a_{2t}$ of cost $c_0,c_1,c_2,\ldots,c_{2t}$ respectively; $c_\ell \in \mathbb{R}_+$ for $0 \leq \ell \leq 2t$. Here letter $a_{2\ell}$ corresponds to following the pointer to the $(\ell+1)$-st subtree, $0 \leq \ell \leq t$, and letter $a_{2\ell+1}$ corresponds to a successful search terminating in the

$(\ell+1)$-st key of a node, $0 \leq \ell < t$.

A search tree is a prefix code $C = \{V_0, W_1, V_1, \ldots, W_n, V_n\}$ with

1) $V_j \in \Sigma^*$, $W_i \in \Sigma^* \Sigma_{end}$

where $\Sigma = \{a_0, a_2, \ldots, a_{2t}\}$, $\Sigma_{end} = \{a_1, a_3, \ldots, a_{2t-1}\}$, $0 \leq j \leq n$, $1 \leq i \leq n$. W_i describes the search process leading to key B_i and V_j describes the search process leading to interval (B_j, B_{j+1}).

2) the ordering of the keys is reflected in the lexicographic ordering of the code words, i.e.

$$V_j \prec W_i \prec V_{j'}$$

for $j < i \leq j'$ and $\prec$ denoting the lexicographic ordering of strings based on the ordering $a_0 \prec a_1 \prec a_2 \prec \ldots \prec a_{2t}$ of the letters.

Conversely, every prefix code for $\alpha_0, \beta_1, \ldots, \beta_n, \alpha_n$ satisfying 1) and 2) corresponds to a search tree in a natural way. We refer to these codes as alphabetic prefix codes. Codes, which not necessarily satisfy 1) and 2) will be called non-alphabetic prefix codes, or simply prefix codes. In order to stress the distinction in the sequel, we will denote the probability distribution by $(p_1, \ldots, p_n)$, the letters by $b_1, \ldots, b_s$ and their costs by $d_1, \ldots, d_s$ in the non-alphabetic case.

The cost of word $a_{i_1} a_{i_2} \ldots a_{i_k}$ is defined as $c_{i_1} + c_{i_2} + \ldots + c_{i_k}$, i.e. as the sum of the letter costs. The cost of code C is then defined as

$$\text{Cost}(C) = \sum_{i=1}^{n} \beta_i \, \text{Cost}(W_i) + \sum_{j=0}^{n} \alpha_j \cdot \text{Cost}(V_j)$$

The following two problems are of immediate interest.

1) Given letters, their costs and a probability distribution, find an alphabetic code of (nearly) minimal cost.

2) Give good a-priori bounds for the cost of an optimal alphabetic code. (an alphabetic "noiseless coding theorem").

Table I gives a survey of algorithms for the construction of optimal codes. For the general problem the input is a probability distribution and letter costs. No efficient algorithm is known in the non-alphabetic case; however, it is also not known whether the corresponding recognition

	alphabetic	non-alphabetic
general problem	$O(t^2n^2)$ Itai	$\in$ NP
binary, equal cost general	$O(n^2)$ Knuth (a)	unnatural problem
leaf-oriented, $\beta_i = 0$	$O(n \log n)$ Hu/Tucker, Garsia/Wachs	$O(n \log n)$ Huffman

Table 1: Algorithms for the construction of optimal codes

problem is NP-complete. Itai's algorithm is a dynamic programming approach and so is Knuth's. In the binary equal cost case we have $t = 1$ and $c_0 = c_1 = c_2 = 1$. The input is a probability distribution $\alpha_0,\beta_1,\alpha_1,\ldots,\beta_n,\alpha_n$. In the leaf-oriented case we have in addition $\beta_i = 0$ for all i. Huffman's algorithm runs in linear time when the probabilities are sorted [van Leeuwen (b)] and the algorithms of Hu/Tucker and Garis/Wachs are isomorphic [Mehlhorn/Tsagarakis].

In summary, we can state that no efficient algorithm is known in the general case. Note that Itai's and Knuth's algorithm also need space $\Theta(n^2)$. Therefore approximation algorithms were considered early in the game [Bruno/Coffman, Gotlieb/Walker]. Most of these algorithms are based on an "alphabetic noiseless coding theorem".

A plausibility argument: Consider a prefix code over a two letter alphabet of cost d_0,d_1 respectively. In the root of the code tree the set of probabilities is split into two sets of probability p and 1-p, say. The letter of cost d_0 (d_1) is assigned to the first (second) set. Hence the average cost arising in the root of the code tree is $d_0 \cdot p + (1-p) \cdot d_1$ and the average information gain is the binary entropy $H(p,1-p) = -p \log p - (1-p)\log(1-p)$. Information gain per unit cost $(H(p,1-p)/(d_0p+d_1(1-p)))$ is maximized (elementary calculus) with value d for $p = 2^{-dd_0}$ and $1-p = 2^{-dd_1}$ where $d \in \mathbb{R}$ is such that $2^{-dd_0} + 2^{-dd_1} = 1$. Since $H(p_1,\ldots,p_n) = -\Sigma p_i \log p_i$ bits have to be gained in any non-alphabetic prefix code for distribution $(p_1,p_2,\ldots,p_n)$ the cost of such a code has to be at least $H(p_1,\ldots,p_n)/d$. This is made precise in Theorem 2.1. Moreover, the plausibility argument also suggests an

approximation algorithm. Try to split the distribution into two sets of probability about 2^{-dd_0} and 2^{-dd_1} respectively. Then proceed recursively on the two subsets.

Theorem 2.1:

a) [Krause, Ciszar]. Let $C = \{U_1,\ldots,U_n\}$ be a prefix code for distribution $(p_1,\ldots,p_n)$ over alphabet $\Sigma = \{b_1,\ldots,b_s\}$ with costs $d_1,\ldots,d_s$. Then

$$\text{Cost}(C) = \sum_{i=1}^{n} p_i \cdot \text{Cost}(U_i) \geq H(p_1,\ldots,p_n)/d$$

where

$$\sum_{i=1}^{n} 2^{-d_i d} = 1 .$$

b) [Altenkamp/Mehlhorn]. Let $h \in \mathbb{R}$, $h \geq 0$ and

$$L_h = \{i;\ d\cdot\text{Cost}(U_i) \leq -\log p_i - h\}$$

Then

$$\sum_{i\in L_h} p_i \leq 2^{-h}$$ □

Remark: Part b) of Theorem 1 shows that the inequality of a) is almost true for corresponding terms of the two sums. Part a) is a noiseless coding theorem for arbitrary letter costs.

Of course, Theorem 2.1 also gives a lower bound in the alphabetic case. However, a better bound can be proved in that case.

Theorem 2.2 [Altenkamp/Mehlhorn]. Let $C = \{V_0,W_1,\ldots,W_n,V_n\}$ be an alphabetic prefix code for distribution $(\alpha_0,\beta_1,\alpha_1,\ldots\ )$ over letters $a_0,a_1,\ldots,a_{2t}$ with cost $c_0,\ldots,c_{2t}$. Then

$$\text{Cost}(C) \geq \max\{H(\alpha_0,\beta_1,\ldots)/c(x) - (x-1)\cdot(\Sigma\beta_i)\cdot \max_{k \text{ odd}} c_k;\ 1 \leq x \leq \infty\}$$

where $c(x)$ is such that $\sum_{k=0}^{t} 2^{-c(x)c_{2k}} + \sum_{k=0}^{t-1} 2^{-c(x)\cdot x\cdot c_{2k+1}} = 1.$

Proof (sketch): The idea of the proof is to approximate the combinatorial restriction that letters in Σ_{end} can be used only at the end of code words by an artificial increase in the cost of those letters. So define new costs by $\tilde{c}_k = c_k$ for k even and $\tilde{c}_k = x\cdot c_k$ for k odd and $1 \leq x \leq \infty$. Then relate the cost of code C under the new and the old cost function and apply Theorem 2.1 for the new cost function. □

Corollary 2.3 [Altenkamp/Mehlhorn]: Let C be an alphabetic prefix code for distribution $(\alpha_0,\beta_1,\dots)$ with respect to costs $c_0,c_1,\dots,c_{2t}$. Let c,d be such that

$$\sum_{k=0}^{2t} 2^{-cc_k} = 1 \quad \text{and} \quad \sum_{k=0}^{t} 2^{-dc_{2k}} = 1$$

Then there are constants u,v (depending on the c's but not on Cost(C) and the α's and β's) such that

$$H(\alpha_0,\beta_1,\dots) \leq d\cdot Cost(C) + \frac{c\cdot\Sigma\beta_i}{u} \cdot \max_{i \text{ odd}} c_i[1+\ln(u\cdot v\cdot Cost(C))]+ \frac{1}{2u}$$

Remark: Corollary 2.3 shows that the lower bound for the alphabetic code is essentially the lower bound (d·Cost(C)) for the non-alphabetic code over Σ plus a small correction of order $(\Sigma\beta_i\cdot\ln Cost(C))$ which reflects the restricted usage of letters in Σ_{end}.

Proof (sketch): Let c(x) = d+δ(x) where c(x) is as in Theorem 2.2 and c,d are as in Theorem 2.3. Then $0 \leq \delta(x) \leq c-d$ and $\delta(x) \leq v\cdot e^{-u(x-1)}$ for some constants u and v. This can be verified by substituting into the definitions of c(x),c,d and δ(x). Next substitute the upper bound for c(x) = d+δ(x) into Theorem 2.2 and apply differential calculus to find x which gives the best bound.

Table 2 summarizes the lower bounds for the costs of alphabetic and non-alphabetic prefix codes. Bayer's lower bound is a special case of Theorem 2.2.

	alphabetic	non-alphabetic
general problem	2.2 and 2.3	2.1
binary, equal cost general	$\max\{(H-d\Sigma\beta_i)/\log(2+2^{-d}); d \in \mathbb{R}\}$ Bayer	
leaf-oriented		

Table 2: Lower bound for the costs of prefix codes

Next we turn to the construction of nearly optimal alphabetic prefix codes. The clue is the plausibility argument preceeding Thereorem 2.1. We illustrate the methods by way of example.

Example: Let $c_0 = c_1 = c_2 = 1$ and $(\alpha_0,\beta_1,\alpha_1,\ldots,\beta_4,\alpha_4) = (1/6,1/24,0,1/8,0,1/8,1/8,0,5/12)$. Then $c = 1$ and $d = \log 3$ as defined in Corollary 2.3.

Figure 2.1 shows the probability distribution drawn on the unit line. The plausibility argument suggests that we should split the distribution into two subsets of probability about 2^{-cc1} and 2^{-cc2} respectively,

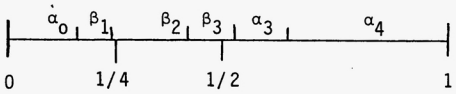


Figure 2.1:The distribution (1/6, 1/24, 0, 1/8, 0, 1/8, 1/8, 0, 5/12).

here 1/2 and 1/2. In our example 1/2 lies in the left half of α_3. Hence we should assign letter a_0 to $\alpha_0,\beta_1,\alpha_1,\beta_2,\alpha_2$, letter a_1 to β_3 and letter a_2 to $\alpha_3,\beta_4,\alpha_4$. Next we work on the subproblem $\alpha_0,\beta_1,\alpha_1,\beta_2,\alpha_2$.

Method 1: Apply the same strategy recursively, i.e. try to split the distribution $(\alpha_0,\beta_1,\alpha_1,\beta_2,\alpha_2)$ in the relation 1/2 : 1/2. This would assign letter a_0 to α_0, letter a_1 to β_1 and letter a_2 to $\alpha_1,\beta_2,\alpha_2$.

Theorem 2.4 [Mehlhorn(d)]. Method 1 constructs an alphabetic prefix code with $\text{Cost}(C) \leq H(\alpha_0,\beta_1,\ldots)/d + (\Sigma\alpha_j)[1/d + \max_{k \text{ even}} c_k] + \Sigma\beta_i[\max_{k \text{ odd}} c_k]$

where d is defined as in Corollary 2.3 . □

In the binary case, a slightly better bound is due to Horibe.

Method 2: In order to solve the subproblem $(\alpha_0,\ldots,\alpha_2)$ method 1 splits this distribution in the relation $2^{-cc1}:2^{-cc2}$. Method 2 splits the interval $[0:2^{-cc1}]$ in the relation $2^{-cc1}:2^{-cc2}$; this yields 1/4 in our case. Hence letter a_0 is assigned to $\alpha_0,\beta_1,\alpha_1$, letter a_1 to β_2 and letter a_2 to α_2. Note that the two methods are identical if exact splits are always possible.

Theorem 2.5 [Csizar, Krause, Altenkamp/Mehlhorn]:

a) Method 2 constructs a prefix code C with

$$Cost(W_i) \leq [-\log \beta_i]/d + \max_{k \text{ odd}} c_k$$

$$Cost(V_i) \leq [-\log \alpha_j + 1]/d + \max_{k \text{ even}} c_k$$

b) Theorem 2.4 with Method 1 replaced by Method 2. □

In the binary, equal cost case results of type a) of 2.5 are also known for method 1 and for optimal trees [Katona/Nemetz, Güttler et al]. For both methods efficient implementations are known.

Theorem 2.6 [Fredman, Altenkamp/Mehlhorn]. It is possible to construct an alphabetic prefix code according to methods 1 and 2 in time $O(t \cdot n)$.

Method 3: There is a third way to interpret the plausibility argument. Try to maximize the information gain per unit cost at every step. A partial analysis of method 3 is available in the binary, equal cost case [Horibe/Nemetz, Güttler et al]. Recent results of Horibe/Nemetz suggest that a linear average time implementation is possible.

Finally we want to mention that yet another approach was followed by Cot. His method is based on Kraft's inequality.

II.2 Tries

In this section we briefly consider the case that the keys are tuples over some set Δ and that only comparisons between components of the keys are possible. One popular search strategy is to iteratively determine the components of the keys (TRIE).

Let $S \subseteq \Delta^m$ be a set of m-tuples over Δ. For $w \in \Delta^*$ let $p_w = |\{x \in S;$ w is a prefix of $x\}|$. Now, let $y = (y_1, y_2, \ldots, y_m)$ be an arbitrary element of S. In order to identify y we proceed as follows.

$i \leftarrow 1$; $w \leftarrow \varepsilon$ (the empty string)

while $i \leq m$ do

begin identify y_i in a binary search tree for Δ and probability distribution $(p_{wa}/p_w; a \in \Delta)$;

$i \leftarrow i+1$; $w \leftarrow wy_i$

end

If the search trees are constructed according to method 2 above then $-\log(p_{y_1 \ldots y_i}/p_{y_1 \ldots y_{i-1}}) + 2$ comparisons suffice to identify y_i

according to theorem 2.5 a. Hence $2m + \log p_\varepsilon/p_y = 2m + \log |S|$ comparisons suffice to identify y.

Theorem 2.7 [Fredman (a),(b), v.Leeuwen (a), Güttler et al.]

Let $S \subseteq \Delta^m$. Then searching in S requires no more than $2m + \log |S|$ comparisons between elements of Δ.

II.3 Dynamic Search Trees

In many applications the probability distribution varies over time. Recently, some methods were proposed to keep a search tree nearly optimal as probabilities change [Allen/Munro, Unterauer, Baer, Mehlhorn (a)].

III. Sorting and Information Theory

In this section we want to apply the knowledge of the previous section to sorting. We will only consider sorting algorithms which sort by means of comparisons with binary outcome. Such algorithms correspond to prefix codes, usually called decision trees in this context, in a natural way.

Let Γ be a subset of the set of permutations of n elements. For instance, Γ could be all permutations with $\leq$ F inversions (for a definition see below). We want to consider algorithms which sort under the assumption that only permutations in Γ are legitimate input sequences.

Theorem 3.1: Let A be a sorting algorithm for sequences in Γ. Then A uses at least $\log|\Gamma|$ comparisons on the average.

Proof: A is a prefix code for probability distribution $(p_1,\ldots,p_{|\Gamma|})$ with $p_i = 1/|\Gamma|$ for all i. □

Theorem 3.1 is usually referred to as the information theoretic lower bound in sorting. Fredman has shown that this lower bound is sharp up to a linear factor of O(n). This is most easily seen be describing a permutation by its inversion table.

Let $x_1x_2\ldots x_n$ be a sequence of distinct elements from an ordered universe. For $1 \leq i \leq n$ let $f_i = |\{j;j > i \text{ and } x_j < x_i\}|$ be the number of elements to the right of i yet smaller than x_i. The tuple $(f_1,\ldots,f_n)$ is called the inversion table and Σf_i is called the total number of inversions of the sequence. Knowledge of the inversion table permits a simple insertion sort of the sequence: Start with an empty sequence and then iteratively insert x_i at the f_i-th position of the sorted version of $x_{i+1},\ldots,x_n$. Hence determination of the inversion table is tantamount to sorting.

Theorem 3.2 [Fredman (a),(b)]. There is a sorting algorithm A_Γ for Γ which never uses more than $\log|\Gamma| + 2n$ comparisons.

Proof: Let $\tilde{\Gamma}$ be the set of inversion tables corresponding to permutations in Γ. By theorem 2.7 searching in $\tilde{\Gamma}$ can be done with at most $\log|\tilde{\Gamma}| + 2n$ comparisons. □

Remark: An explicite construction of algorithm A_Γ requires complete knowledge of $\tilde{\Gamma}$ which one does not have in general.

Examples:

a) Sorting X+Y. Let X and Y be sets of m distinct reals each. Then $X+Y = \{x+y;\ x \in X,\ y \in Y\}$ has m^2 elements. Only $2^{O(m \log m)}$ out of $(m^2)!$ permutations are possible for a set of form X+Y [Harper et al] and hence X+Y may be sorted with $O(m^2)$ comparisons [Fredman]. Note that $n = m^2$ in theorem 3.2.

b) Let Γ be the set of permutations of n elements with $\leq$ F inversions. Then $\log|\Gamma| = O(n \log \frac{F}{n})$ and hence sequences with $\leq$ F inversions can be sorted with $O(n \log \frac{F}{n})$ comparisons. In other words, presorted files ($\log F/n = o(\log n)$) can be sorted with strictly less than n log n com-

parisons. Note however, that Fredman's result does not say anything about the possible running time of an algorithm for sorting presorted files. Recently, algorithms with run time $O(n(1+\log F/n))$ were described by Guibas et al, Brown/Tarjan and Mehlhorn. Mehlhorn's solution is the most efficient. We describe a blend of Mehlhorn's and Brown/Tarjan's algorithm here.

Definition [Adelson-Velskii/Landis]. A binary tree T (every node of T has either 2 or no sons) is an AVL-tree if for every node v of T the difference between the heights of the left and right subtree of v is at most one. The difference is denoted hb(v). An ordered set S is stored in an AVL-tree by labelling the nodes of the tree from left to right with the elements of S. (cf. figure 3.1).

Suppose now that we want to use an AVL-tree for an insertion sort. Say $x_{i+1},\ldots,x_n$ are stored in sorted order in AVL-tree T and x_i is to be inserted next. By the definition of f_i, x_i will be inserted at the f_i-th position of the sorted version of $x_{i+1},\ldots,x_n$. If the file is presorted, i.e. Σf_i is small, the elements x_i will tend to be inserted near the beginning of the sorted sequence. Hence the standard insertion algorithm will on the average deviate from the left spine (= the path from the root to the left-most leaf) near the left-most leaf. Thus it is much cheaper to look for that point by starting at the left-most leaf y_1 and walking towards the root until a node y_{k_i} with $x_i < y_{k_i}$ is found. (cf. figure 3.2). Next x_i is inserted into the right subtree of y_{k-1} as usual. Since the height of node y_{k_i} is at most $2k_i$ it takes $O(k_i)$ units of time to find the position where x_i is to be inserted. After the correct position is found and a new leaf with label x_i is inserted, the AVL-tree needs to be rebalanced. Rebalancing is restricted to the path from the new leaf to the first node z with balance $hb(z)\neq 0$ on the path from the new leaf to the root. [see Knuth b, pp. 451-463 for details]. Let ℓ_i be the length of that path. Then rebalancing after the insertion of x_i takes time $O(\ell_i)$.

In summary, it takes $O(k_i+\ell_i)$ units of time to process x_i and hence the complete sort takes time $\sum_{i=1}^{n} O(k_i+\ell_i)$.

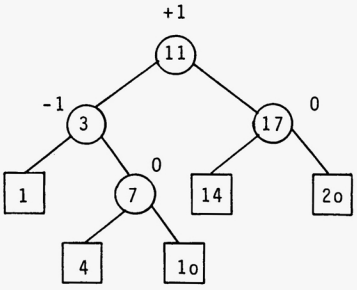


Figure 3.1: An AVL-tree for $S = \{1,3,4,7,1o,11,14,17,2o\}$. Height-balances are shown above nodes.

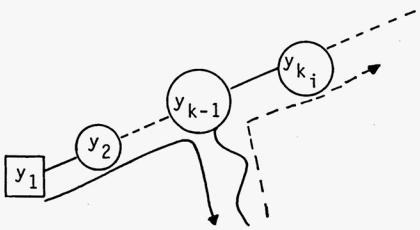


Figure 3.2: The insertion (———) and rebalancing (------) paths.

Lemma 3.3:

a) $k_i = O(\log f_i)$

b) $\Sigma k_i = O(n(1+\log \frac{\Sigma f_i}{n}))$

c) $\Sigma \ell_i = 5n$

Remark: Part c) of Lemma 3.3 is interesting in its own right. It shows that the total number of rebalancing operations (= balance changes + rotations and double rotations) is linear in the number of insertions. A more careful analysis shows $\Sigma \ell_i \leq 4n$.

Proof:

a) x_i is larger than all elements in the subtree with root y_{k_i-2}. Since the height of y_{k_i-2} is at least k_i, this subtree contains at least $Fib(k_i-2+2)-1$ nodes [see Knuth, page 453], where $Fib(m)$ is the m-th Fibonacci number. This shows a).

b) From a) we conclude

$$\Sigma k_i = \Sigma O(\log f_i) = O(n + \log \Pi f_i)$$

$$= O(n + \log(\frac{\Sigma f_i}{n})^n) = O(n(1+\frac{\Sigma f_i}{n}))$$

The third inequality follows from the fact that Πf_i is maximal for

$f_i = f_j$ for all i,j.

c) Rebalancing after the insertion of x_i is tantamount to changing the balance of all nodes between the new leaf and node z from 0 to ± 1 followed by either a change of balance at node z (from ± 1 to 0) or a rotation or double rotation about z. A rotation (double rotation) changes the balance of at most 3 nodes. In summary, rebalancing after the insertion of x_i generates at most 4 nodes with balance 0 (the father of the new leaf and the 3 nodes affected by the rotation) and changes the balance of at least $\ell_i - 1$ nodes from 0 to ± 1. Hence $\Sigma(\ell_i - 1) \leq 4n$ since only zeroes which were created before can be changed. □

As an immediate consequence of Lemma 3.3 we obtain

Theorem 3.4: It is possible to sort sequences with $\leq$ F inversions in time $O(n(1+\log F/n))$ where n is the length of the sequence.

The algorithm behind 3.4 may be dubbed an adaptive sorting method. Hard sorting problems ($F \approx n^2/2$) are solved in time $O(n \log n)$ and simple sorting problems ($F << n^2$) are solved in time strictly less than $O(n \log n)$. In [Mehlhorn] the algorithm is compared with Quicksort and it is shown to be superior for $F \leq 0.25\, n^{1.8}$.

Bibliography:

Adel'son-Velskii, Landis : An algorithm for the organization of information, Soviet Math. Dokl, 3, 1259-1262.

Allan, B., Munro, I. : Self-Organizing Binary Search Trees, JACM, Vol. 25 (1978), pp. 526-535.

Altenkamp, D., Mehlhorn, K. : Codes : Unequal Letter Costs, Unequal Probabilities, Technical Report A 78/18, FB 1o, Universität des Saarlandes (preliminary version: 5th Colloquium on Automata, Languages and Programming, Udine, 1978, Lecture Notes in Computer Science 62, pp. 15-25.

Baer, J. L. : Weight-balanced trees, Proc. AFIPS, Vol. 44, 1975, pp. 467-472.

Bayer, P. J. : Improved Lower Bounds on the Cost of Optimal and Balanced Binary Search Trees, Technical Report, Dept. of Computer Science, MIT.

Brown, M.R., Tarjan, R.E. : A Representation for Linear Lists with Movable Fingers, 1oth ACM Symposium on Theory of Comouting, pp. 19-29, 1978.

Bruno, J., Coffman, E.G.: Nearly Optimal Binary Search Trees (IFIP 1971, North Holland 1972, pp. 99-1o3).

Cot, H. : Characterization and Design of Optimal Prefix Codes, Ph. D. Thesis, Stanford University, June 1977.

Csiszar : Simple Proofs of some theorems on noiseless channels, Inf. and Control 14, pp. 285-298, 1969.

Fredman, M.L. (a) : Two Applications of a Probabilistic Search Technique : Sorting X+Y and Building Balanced Search Trees, 7th ACM Symposium on Theory of Computing, 1975, pp. 24o-244.

Fredman, M.L. (b) : How good is the information theory bound in sorting, TCS 1, (4), pp. 355-362, 1976.

Garsia, A.M., Wachs, M.L. : A new algorithm for minimum cost binary trees, SICOMP, No 4, 1977, pp. 622-642.

Gotlieb, Walker : A Top-Down Algorithm for Constructing Nearly Optima Lexicographical Trees, Graph Theory and Computing, Academic Press, 1972.

Guibas, L.J., McCreight, E.M., Plass, M.F., Roberts, J.R.: A new representation for linear lists, 9th ACM Symposium on Theory of Computing, 1977, pp. 49-6o.

Güttler, R., Mehlhorn, K., Schneider, W.: Binary Search Trees : Average and Worst Case Behavior, GI-Jahrestagung 1976, Informatik Fachberichte 5, pp. 3o1-317.

Harper, Pague, Savage, Straus : Sorting X+Y, CACM 18, 6 (1975), pp. 347-35o.

Horibe, Y. : An improved bound for weight balanced trees, Inf. and Control 34, 1977.

Horibe, Y., Nemetz, T. : On the Max-Entropy Rule for a Binary Search Tree, Technical Report, Mathematical Institute, Hungarian Academy of Sciences.

Hu, Tucker : Optimum Computer Search Trees, SIAM J. of Applied Math. 21, 1971, pp. 514-532.

Huffman : A method for the Construction of Minimum-Redundancy Codes, Proc. IRE 4o, 1o98-11o1, 1952.

Itai, A. : Optimal Alphabetic Trees, SIAM J. on Computing, Vol. 5, No. 1, March 1976, pp. 9-18 .

Katona, G., Nemetz, T. : Huffman Codes and Self Information, IEEE Transactions on Information Theory, May 1976, pp. 337-34o.

Knuth, D.E. (a) : Optimum Binary Search Trees, Acta Informatica 1, 1971, pp. 14-25 .

Knuth, D.E. : The Art of Computer Programming, Vol 3 : Sorting and Searching, Addison Wesley, 1973 .

Krause : Channels which transmit letters of unequal duration, Inf. and Control 5, pp. 13-24, 1962 .

Mehlhorn, K. (a) : Dynamic Binary Search, 4th Colloquium on Automata, Languages and Programming, Turku, 1977, Springer Lecture Notes 52, pp. 323-336 (to appear SIAM J. on Computing).

Mehlhorn, K. (b) : Sorting Presorted Files, 4th GI-Conference on Theoretical Computer Science, Aachen, 1979 .

Mehlhorn, K. (c) : Effiziente Algorithmen, Teubner Verlag, Studienbücher Informatik, 1977 .

Mehlhorn, K. (d) : An Efficient Algorithm for the Construction of Nearly Optimal Prefix Codes, Technical Report A 78/13, FB 1o, Universität des Saarlandes, submitted for publication.

Mehlhorn, K., Tsagarakis, M. : On the isomorphism of two algorithms : Hu/Tucker and Garsia/Wachs, 4ième Colloque de Lille "Les Arbes en Algebre et en Programmation", Lille 1979 .

van Leeuwen, J. (a) : The complexity of data organization, 1976, Mathematical Centre Tract 81, pp. 37-147.

van Leeuwen, J. (b) : On the construction of Huffmann Trees, 3rd ICALP (1976), pp. 382-41b, Ed. S. Michaelson and R. Milner, Edinburgh University Press.

Unterauer, K. : Optimierung gewichteter Binärbäume zur Organisation geordneter dynamischer Dateien, Doktorarbeit, TU München, 1977.