Acta Cybernetica **19** (2010) 661–672.

An Algorithmic Comparison of Three Scientific Impact Indices*

Gerhard J. Woeginger†

Abstract

We use tools from Theoretical Computer Science to analyze the computational complexity of determining the h-index, the g-index, and the w-index in various models of computation. Our results confirm the natural intuition that the h-index is an easier concept than the g-index, which in turn is an easier concept than the w-index.

Keywords: scientific impact measure, efficient algorithm, computational complexity

1 Introduction

Citation analysis as a method for ranking scientific journals, publications, and researchers is an old idea that goes at least back to Gross and Gross ([7]). A simple and natural approach for quantifying the scientific productivity and scientific impact of a researcher with $n \geq 0$ publications is based on the so-called *citation sequence* $\langle x_1, \ldots, x_n \rangle$ of the researcher; here the kth element x_k states the total number of citations to the kth publication. A *scientific impact index* assigns to every such citation sequence a corresponding non-negative integer that concisely expresses the productivity, quality, and visibility of this researcher.

In recent years, the h-index of Jorge Hirsch ([8]) and the g-index of Leo Egghe ([5], [6]) have become particulary popular impact indices in this area. Woeginger ([11],[12]) performed an axiomatic analysis of the h-index and the g-index, and as a by-product these axiomatic investigations lead to the definition of the so-called w-index.

The h-index: A scientist has index h, if h is the largest integer such that at least h of his articles have received at least h citations each.

*This work has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403, and by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society)

†Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands., E-mail: `gwoegi@win.tue.nl`

The g-index: A scientist has index g, if g is the largest integer such that his top g articles have received together at least g^2 citations.

The w-index: A scientist has index w, if w is the largest integer such that w of his articles have received at least $1, 2, 3, 4, \ldots, w$ citations, respectively.

In this paper, we will discuss the algorithmic complexity of computing the h-index, the g-index, and the w-index of a researcher, and we will thereby compare the relative effort needed for determining these three indices. Let us start with an example, and let us consider a researcher X whose 19 publications have attracted the following numbers of citations:

$$10,\ 20,\ 20,\ 12,\ 55,\ 2,\ 2,\ 4,\ 0,\ 14,\ 15,\ 14,\ 0,\ 0,\ 9,\ 10,\ 2,\ 1,\ 3.$$

It is not difficult to see that the h-index of Mr. X is 9 (hint: search for 9 citation numbers that are all at least 9). The task becomes much easier for the human eye, if one first brings the numbers into non-increasing order as follows:

$$55,\ 20,\ 20,\ 15,\ 14,\ 14,\ 12,\ 10,\ 10,\ 9,\ 4,\ 3,\ 2,\ 2,\ 2,\ 1,\ 0,\ 0,\ 0.$$

What about the g-index of Mr. X? Since the sum of the 13 highest numbers in this sequence is $188 \geq 13^2$, and since the sum of the 14 highest numbers is $190 < 14^2$, we see that his g-index is 13. Finally the w-index of this researcher is 14, since the top publication has $55 \geq 14$ citations, since the second-strongest publication has $20 \geq 13$ citations, the third-strongest $20 \geq 12$, the fourth-strongest $15 \geq 11$, and so on down to the publication at rank 14 with $2 \geq 1$ citations.

After building up some more experience and after determining these three impact indices for several dozens of researchers, one inevitably comes to the following conclusion: The h-index is the most primitive index among the three, and usually can be found quickly and easily. The g-index takes somewhat more effort to compute, and in particular involves the summing of a lot of numbers. Finally the w-index seems to be a pain in the neck, and one really has to compare a whole lot of numbers one by one for finding the w-index.

Contributions of this paper. The goal of this paper is to confirm the fuzzy observations that were claimed in the preceding paragraph without much justification. We will reach this goal by applying machinery from algorithms theory, and by analyzing the time complexity for computing the three impact indices under various models of computation: First, we will consider the model where the citation sequence is stored in fast random access memory and has already been sorted into non-increasing order. Secondly, we consider the situation where the citation sequence is in random access memory, but is unordered. Thirdly, we will consider the model where the citation sequence is stored in sequential memory (as for instance on a tape), and where accessing a data element cannot be done instantaneously, but costs some data reading and data processing time.

It turns out that in all three models the g-index can be computed within a time complexity that is proportional to the length n of the citation sequence. The

h-index behaves similarly as the g-index, but in the first of the above models it is (provably!) much easier to compute. The w-index can also be computed with linear effort in the first and second model, but in the third model it is (provably!) harder to compute than the h-index and the g-index. Our results are summarized in Table 1.

We stress that our main contribution does not consist of the algorithms derived in this papers: These algorithms are purely theoretical constructions that have been tailored to work under certain simplified models of computation and that most probably have no practical relevance. In fact the calculation of all three indices is relatively easy and could be done quickly even for millions of data records on any modern PC by the most primitive and direct implementations. Our main contribution is purely conceptual: We use these tools from Theoretical Computer Science to provide mathematical evidence that the h-index is a more primitive concept than the g-index and that the g-index is a more primitive concept than the w-index. And actually, we are not aware of any other scientific tools that would be able to yield such a result.

The paper is organized as follows. Section 2 gives a soft introduction into the analysis of algorithms, and also specifies the three considered models of computation more precisely. Sections 3, 4, 5 respectively discuss how to compute the considered impact indices in the three models of computation. Section 6 gives the conclusion.

Table 1: Asymptotic worst case time complexities for computing various impact indices (of citation sequences with n elements) in various models of computation. All nine results are asymptotically best possible.

	h-index	g-index	w-index
Sorted data in random access memory	$\log n$	n	n
Unsorted data in random access memory	n	n	n
Unsorted data in sequential memory	n	n	$n \log n$

2 Preliminaries on algorithms and computation

In this section we summarize some very basic facts on the analysis of algorithms and on models of computation. For more information on these concepts, we refer the reader to the books of Aho, Hopcroft, and Ullman ([1]), Cormen, Leiserson, and Rivest ([4]), and Papadimitriou ([10]).

Algorithms can be evaluated by a variety of criteria. The most common criterion is the growth of the running time required to solve larger and larger instances of a problem: Every instance of a problem has a certain *size*, which measures the quantity of input data. For example the size of a citation sequence $\langle x_1, \ldots, x_n \rangle$ is the number n of elements in the sequence. The *time complexity* of an algorithm is a worst case measure and denotes the maximum number of elementary steps needed by the algorithm as a function of the input size; in other words if an algorithm has time complexity $T(n)$ then it can solve all instances of size n within $T(n)$ elementary steps. The *asymptotic time complexity* of an algorithm is the limiting behavior of the worst case time complexity as size increases. If an algorithm can process all inputs of size n in time $T(n) \le cn^2$ for some constant c, then we say that its time complexity is $O(n^2)$. More precisely, a time complexity function $f(n)$ is said to be in $O(g(n))$, if there exists a constant c such that $f(n) \le cg(n)$ holds for all positive n.

In the preceding paragraph we have specified the running time of an algorithm as the number of elementary steps. The definition of an elementary step heavily depends on the underlying model of computation. The most common model of computation is the random access machine (RAM) model. Every storage cell of a RAM can hold an integer. The integers in two storage cells can be added, subtracted, multiplied, divided, or compared against each other in one elementary step. The contents of every storage cell can be accessed instantaneously, and the results of additions, subtractions, multiplications, and divisions can be stored instantaneously into new storage cells. If a citation sequence $\langle x_1, \ldots, x_n \rangle$ is stored in the cells of a RAM, then each element x_k can be accessed through its index k within a single elementary step. The random access machine is the underlying model of computation for Sections 3 and 4: In Section 3 we will additionally assume that the input citation sequence has already been sorted into non-increasing order $x_1 \ge x_2 \ge \cdots \ge x_n$. In Section 4 we discuss the case of unordered citation sequences.

Another fundamental model of computation stores the data in *sequential memory*. Also in this model every storage cell can hold an integer, but now accessing the storage cells is more expensive: The data is stored and processed on a fixed number of tapes (or lists), and the data on every tape is accessed through a read-write head for this tape. In this model the contents of a storage cell can not be accessed instantaneously: For instance, if we want to access the contents of storage cell #1 and afterwards the contents of storage cell #1.000, then the read-write head must inbetween move from cell #1 to cell #1.000, which takes 999 elementary steps. Hence moving one cell along the tape, reading a cell, and writing into a cell form elementary steps in the sequential memory model. The model also assumes that there are a small fixed number of register cells (in fast memory) available that can be used for temporarily storing data. Addition, subtraction, multiplication, division, and comparison of the integers in the register cells are elementary steps.

In Section 5 we will discuss the computation of impact indices under the sequential memory model. As a main tool we will use a celebrated algorithm for selecting the m-largest element among n numbers.

Proposition 1. *(Blum, Floyd, Pratt, Rivest, and Tarjan, [3])*
Consider n numbers that are stored in random access memory or in sequential memory. Then the m-largest element among these n numbers can be determined in linear time $O(n)$.

3 The case of sorted data in random access memory

Throughout this section we consider citation sequences $\langle x_1, \ldots, x_n \rangle$ in which the elements are in non-increasing order $x_1 \geq x_2 \geq \cdots \geq x_n$. We assume that the data is stored in fast random access memory, so that each element x_k can be accessed immediately through its index k.

We start our investigations with the h-index. In this case our main tool is the classical *Binary Search* procedure, as discussed for instance in the textbook of Aho, Hopcroft, and Ullman ([1]). We will apply Binary Search to the following auxiliary problem: Given a sorted sequence $y_1 \geq y_2 \geq \cdots \geq y_n$ of integers with $y_1 \geq 0$, determine the largest index h for which y_h is non-negative. The main idea of Binary Search is to narrow down the search space to smaller and smaller intervals $[\ell, u]$. In the beginning the search space is the entire interval $[1, n]$ so that $\ell = 1$ and $u = n$. Then Binary Search looks at the value of the middle element y_m with $m := \lfloor (\ell + u)/2 \rfloor$. If y_m is negative, then the new search space becomes $[1, m-1]$. If y_m is non-negative, then the new search space becomes $[m, n]$. This is repeated until the search space has been narrowed down to at most two elements, which are then checked separately. Since every search step removes half of the search interval, the time complexity $T(n)$ satisfies

$$T(n) \ \leq \ T(\lceil n/2 \rceil) + c,$$

where c is the time needed for computing m and for querying the mth element. Now an easy induction yields $T(n) \leq d \lceil \log_2 n \rceil$ for some appropriate constant d.

Theorem 1. *The h-index of a sorted citation sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ can be determined in $O(\log_2 n)$ time. No algorithm can have a worst case time complexity that is asymptotically better than $O(\log_2 n)$.*

Proof. The h-index of a sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ is the largest integer h for which the value $y_h = x_h - h$ is non-negative. Hence the h-index can be computed by solving the auxiliary problem discussed above for the auxiliary sequence defined by $y_i = x_i - i$ for $1 \leq i \leq n$. This yields the $O(\log_2 n)$ time complexity claimed in the positive part of the theorem.

For the negative part we use an information-theoretic argument. For $1 \leq k \leq n$ consider the sorted citation sequence $S_{n,k}$ that consists of k elements of value n followed by $n-k$ elements of value 0. Clearly the h-index of $S_{n,k}$ equals k. Now consider an arbitrary algorithm A for computing the h-index. Whenever A queries one element in such a sequence $S_{n,k}$, it only gains a single bit of information: The queried element is either equal to n, or it is not (in which case it is 0). Since the algorithm needs at least $\log_2 n$ bits to distinguish between the n possible outcomes, it must query $\log_2 n$ elements in the worst case. $\square$

Theorem 2. *The g-index of a sorted citation sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ can be determined in $O(n)$ time. No algorithm can have a worst case time complexity that is asymptotically better than $O(n)$.*

Proof. First we compute the sum $s[k] = \sum_{i=1}^{k} x_i$ for $k = 1, \ldots, n$. This can be done in overall linear time $O(n)$, since $s[1] = x_1$ and since $s[k] = s[k-1] + x_k$ holds for $k = 2, \ldots, n$. The g-index is then the largest index k with $s[k] \geq k^2$. This yields the $O(n)$ time algorithm for the positive part of the theorem.

For the negative part we consider the following sorted citation sequence S'_n: The first element of sequence S'_n is $\frac{1}{2}(n^2 + n)$. The remaining $n - 1$ elements in S'_n are the values $1, 2, 3, \ldots, n - 1$ in decreasing order. Since the sum of all elements in sequence S'_n is n^2, its g-index is n. Furthermore for $1 \leq k \leq n$ we define a sequence $S'_{n,k}$ that results from sequence S'_n by decreasing the kth element by 1. The resulting sequence $S'_{n,k}$ is still sorted, but its g-index has dropped down to $n - 1$.

Now consider an arbitrary algorithm A for computing the g-index, and feed the input sequence S'_n into algorithm A. We claim that A must inspect all n elements of sequence S'_n: If the algorithm does not inspect the kth element, then it could not distinguish sequence S'_n (with g-index n) from sequence $S'_{n,k}$ (with g-index $n - 1$). $\square$

Theorem 3. *The w-index of a sorted citation sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ can be determined in $O(n)$ time. No algorithm can have a worst case time complexity that is asymptotically better than $O(n)$.*

Proof. The w-index of the sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ is the largest integer k such that $x_k \geq 1$ and $x_{k-1} \geq 2$ and $x_{k-2} \geq 3$ and so on down to $x_1 \geq k$. Equivalently, we may write the w-index as

$$\begin{aligned} w &= \max\{k : \ x_m \geq k - m + 1 \text{ holds for } m = 1, \ldots, k\} \\ &= \max\{k : \ x_m \geq k - m + 1 \text{ holds for } m = 1, \ldots, n\} \\ &= \arg\min\{x_m + m - 1 : \ 1 \leq m \leq n\}. \end{aligned}$$

The minimum value of $x_m + m - 1$ over the domain $1 \leq m \leq n$ can be determined by a single pass over the citation sequence. This yields the desired $O(n)$ time algorithm.

For the negative part we once again consider the sorted sequences S'_n and $S'_{n,k}$ that have been introduced in the proof of Theorem 2. We note that the w-index of sequence S'_n is n, whereas the w-index of every sequence $S'_{n,k}$ with $2 \leq k \leq n$ is $n - 1$. Similarly as in the proof of Theorem 2 we argue that any algorithm for computing the w-index must inspect a linear number of elements in sequence S'_n. □

4 The case of unordered data in random access memory

Throughout this section we consider unordered citation sequences $\langle x_1, \ldots, x_n \rangle$ that are stored in fast memory. As a first step we apply the following algorithm to this unordered sequence. The algorithm essentially emulates sorting through counting; see also Cormen, Leiserson, and Rivest ([4]).

> Phase 1: Initialize a data array $C[0, \ldots, n]$ by setting $C[k] := 0$ for $k = 0, \ldots, n$.
> Initialize a variable BigSum:= 0.
>
> Phase 2: Work through the elements of the citation sequence for $k = 1, \ldots, n$.
> If $0 \leq x_k < n$ holds, then set $C[x_k] := C[x_k] + 1$.
> If $x_k \geq n$ holds, then set $C[n] := C[n] + 1$ and BigSum:=BigSum$+x_k$.
>
> Phase 3: Output the following sorted citation sequence:
> If $C[n] > 0$ holds, then output an element of value BigSum$-(C[n]-1)n$, followed by $C[n]-1$ elements of value n. Furthermore for $k = n-1, n-2, \ldots, 0$ output $C[k]$ elements of value k.

What does this algorithm do to a sequence $\langle x_1, \ldots, x_n \rangle$? Let us first explain the meaning of the variables: The array element $C[k]$ with $0 \leq k \leq n-1$ counts the number of elements of value k in the sequence. The last array element $C[n]$ counts the number of elements of value at least n in the sequence; these elements are called *big* elements. Finally, the variable BigSum contains the total size of all big elements in the sequence.

The values of the counters and of BigSum are determined in Phase 1 and Phase 2. In Phase 3 the citation sequence is output again, but this time in non-increasing order and with some slight changes in the values of the big elements: The total size BigSum of the big elements remains unchanged, but now only a single big element can be strictly larger than n. The reader may want to verify that for $n = 5$ the input sequence $\langle 0, 10, 4, 2, 20 \rangle$ will be transformed into the output sequence $\langle 25, 5, 4, 2, 0 \rangle$.

Lemma 1. *Let $x = \langle x_1, \ldots, x_n \rangle$ be an input sequence for the above algorithm, and let $y = \langle y_1, \ldots, y_n \rangle$ be the corresponding sorted output sequence. Then sequences x and y have the same h-index, the same g-index, and the same w-index.* □

Theorem 4. *If an unordered citation sequence $\langle x_1, \ldots, x_n \rangle$ with n elements is stored in random access memory, then*

(a) its h-index,

(b) its g-index, and

(c) its w-index

can all be determined in linear time $O(n)$. Under the random access model of computation, no algorithm for these three indices can have a worst case time complexity that is asymptotically better than $O(n)$.

Proof. The above algorithm takes an arbitrary input sequence, and transforms it in linear time into a sorted output sequence. By Lemma 1 the sorted sequence has the same h-index (respectively g-index and w-index) as the output sequence. Hence we may apply the fast algorithms for sorted data from the preceding section to the output sequence. This yields the positive part of the theorem.

For the negative part, we consider a citation sequence that consists of $n-1$ elements of value 0 and a single element of value 1 (that is maliciously hidden somewhere between the other $n-1$ elements). Note that the h-index, g-index, and w-index of this sequence are 1. However if an algorithm fails to inspect the element of value 1, then is cannot distinguish the sequence from the all-zero sequence whose h-index, g-index, and w-index are 0. Hence in the worst case the algorithm must query all n elements. □

5 The case of unordered data in sequential memory

Throughout this section we consider unordered citation sequences $\langle x_1, \ldots, x_n \rangle$ whose elements are stored in sequential memory.

We start our discussion with the h-index. We consider the following (purely technically motivated) generalization of the h-index that is built around an integer parameter $p \geq 0$: The $h(p)$-index of a citation sequence $\langle x_1, \ldots, x_n \rangle$ is the largest integer h such that the sequence contains at least h elements that all have value at least $h+p$.

Lemma 2. *Let $y = \langle y_1, \ldots, y_r \rangle$ and $z = \langle z_1, \ldots, z_s \rangle$ be two citation sequences with $y_i \leq z_j$ for all i and j, and let $y_{\max}$ denote the largest value in sequence y. Let $x = \langle x_1, \ldots, x_{r+s} \rangle$ denote the union of sequences y and z, and let $p \geq 0$ be an integer.*

(a) If $y_{\max} \leq s+p$ holds, then the $h(p)$-index of sequence x coincides with the $h(p)$-index of sequence z.

(b) If $y_{\max} > s+p$ holds, then the $h(p)$-index of sequence x equals the $h(p+s)$-index of sequence y incremented by value s.

Proof. Suppose $y_{\max} \leq s+p$. Since the $s+1$ largest elements in sequence x are the s elements $z_1, \ldots, z_s$ and $y_{\max}$, in this case the $h(p)$-index of sequence x is at most s. Hence only the elements in sequence z are relevant. This yields (a).

Next suppose $y_{\max} > s + p$. In this case the s elements $z_1, \dots, z_s$ and element $y_{\max}$ all have value at least $s + p + 1$, and hence the $h(p)$-index of sequence x is at least $s + 1$. Then the $h(p)$-index of x is determined by all s elements in z together with a subset of t elements in y that all have value at least $s + t + p$. Hence we are looking for the largest t such that y contains t elements that all have value at least $s + t + p$; this largest t is precisely the $h(p + s)$-index of y. □

Lemma 2 suggests the following recursive approach for computing the $h(p)$-index of a given sequence $x = \langle x_1, \dots, x_n \rangle$.

> Phase 1: If $n \leq 2$ then determine the $h(p)$-index directly and stop.
> Otherwise set $s := \lfloor n/2 \rfloor$, and determine the value v of the s-largest element in sequence x.
> Split sequence x into a sequence z of length s that contains elements of value $\geq v$, and into a sequence y of length $n - s$ that contains elements of value $\leq v$.
>
> Phase 2: If $v \leq s + p$ holds, then throw away sequence y. Recursively compute and then output the $h(p)$-index of sequence z.
> If $v > s + p$ holds, then throw away sequence z. Recursively compute the $h(p + s)$-index of sequence y, increment it by s, and output the resulting value.

In the beginning sequence x is stored in sequential memory. Proposition 1 allows us to find the s-largest element in $O(n)$ time. The sequences y and z are easily determined and stored in sequential memory in $O(n)$ time (we run through the tape containing x, and split its contents appropriately into two other tapes; afterwards we may reuse the tape that contained sequence x). In Phase 2 we recurse on a sequence of length at most $\lceil n/2 \rceil$. Since every search step removes half of the search interval, the time complexity $T(n)$ of this procedure satisfies

$$T(n) \;\leq\; T(\lceil n/2 \rceil) + O(n).$$

A straightforward induction yields $T(n) \leq cn$ for some appropriate constant c. Finally we note that the h-index coincides with the $h(0)$-index.

Theorem 5. *The h-index of an unordered citation sequence $\langle x_1, \dots, x_n \rangle$ in sequential memory can be determined in $O(n)$ time. Under this model of computation, no algorithm can have a worst case time complexity that is asymptotically better than $O(n)$.*

Proof. The $O(n)$ time algorithm in the positive part follows from the above discussion. The negative result follows along the lines of the negative result in Theorem 4. □

Now let us turn to the g-index. Similarly as for the h-index, we introduce a purely technical generalization that is defined around two non-negative integer parameters $p, q \geq 0$: The $g(p,q)$-index of a citation sequence $\langle x_1, \ldots, x_n \rangle$ is the largest integer g such that the sequence contains g elements that have sum at least $(g+p)^2 - q$. Note that the classical g-index coincides with the $g(0,0)$-index.

Lemma 3. *Let $y = \langle y_1, \ldots, y_r \rangle$ and $z = \langle z_1, \ldots, z_s \rangle$ be two citation sequences with $y_i \leq z_j$ for all i and j, and let Z denote the sum of all elements in sequence z. Let $x = \langle x_1, \ldots, x_{r+s} \rangle$ denote the union of sequences y and z, and let p and q be non-negative integers.*

(a) If $Z \leq (s+p)^2 - q$, then the $g(p,q)$-index of sequence x coincides with the $g(p,q)$-index of sequence z.

(b) If $Z > (s+p)^2 - q$, then the $g(p,q)$-index of sequence x equals the $g(p+s, q+Z)$-index of sequence y incremented by value s.

Proof. Suppose $Z \leq (s+p)^2 - q$. Then the $g(p,q)$-index of sequence x is at most s, and only the elements in sequence z are relevant for it. This yields (a).

Next suppose $Z > (s+p)^2 - q$, in which case the $g(p,q)$-index of x is at least s. Then the $g(p,q)$-index of x is determined by all s elements in z, together with a subset of t elements in y. Let Y' denote the sum of these t elements, and observe that $Z + Y' \geq (s+t+p)^2 - q$ must hold true. These t elements in sequence y whose sum is at least $(s+t+p)^2 - q - Z$ precisely yield the $g(p+s, q+Z)$-index of y. □

Lemma 2 leads to the following recursive approach for computing the $g(p,q)$-index of a given sequence $x = \langle x_1, \ldots, x_n \rangle$.

> Phase 1: If $n \leq 2$ then determine the $g(p,q)$-index directly and stop.
> Otherwise set $s := \lfloor n/2 \rfloor$, and determine the value v of the s-largest element in sequence x.
> Split sequence x into a sequence z of length s that contains elements of value $\geq v$, and into a sequence y of length $n-s$ that contains elements of value $\leq v$.
> Determine the sum Z of all elements in sequence z.
>
> Phase 2: If $Z \leq (s+p)^2 - q$ holds, then throw away sequence y. Recursively compute and then output the $g(p,q)$-index of sequence z.
> If $Z > (s+p)^2 - q$ holds, then throw away sequence z. Recursively compute the $g(p+s, q+Z)$-index of sequence y, increment it by s, and output the result.

Similarly as in the computation of the $h(p)$-index, this algorithm for the $g(p,q)$-index can be implemented to run in $O(n)$ time if the sequence x is stored in sequential memory.

Theorem 6. *The g-index of an unordered citation sequence $\langle x_1, \ldots, x_n \rangle$ in sequential memory can be determined in $O(n)$ time. Under this model of computation, no algorithm can have a worst case time complexity that is asymptotically better than $O(n)$.* □

Finally let us discuss the w-index. It is easy to determine the w-index of an unordered citation sequence $\langle x_1, \ldots, x_n \rangle$ in $O(n \log n)$ time: First sort the elements in $O(n \log n)$ time (a classical sorting algorithm like MergeSort will do this also for data in sequential memory). Then apply the $O(n)$ algorithm from Theorem 3. Interestingly this is already the best asymptotic time complexity one can reach in sequential memory.

Theorem 7. *The w-index of an unordered citation sequence $\langle x_1, \ldots, x_n \rangle$ in sequential memory can be determined in $O(n \log n)$ time. Under this model of computation, no algorithm can have a worst case time complexity that is asymptotically better than $O(n \log n)$.*

Proof. The proof of the negative statement is based on an auxiliary problem called Permutation-Recognition: Given n integers $u_1, \ldots, u_n$, decide whether these integers form a permutation of the numbers $1, 2, \ldots, n$. It is known that this problem cannot be solved with a worst case time complexity better than $O(n \log n)$, if the data is stored in sequential memory; see for instance Ben-Or ([2]).

Consider an arbitrary instance $u_1, \ldots, u_n$ of Permutation-Recognition. In a first step compute the sum U of all elements in this instance in linear time $O(n)$. If $U \neq \frac{1}{2}n(n+1)$, we stop right away with the answer NO. Otherwise we move on, and feed the sequence $u_1, \ldots, u_n$ into an algorithm A for computing the w-index. If algorithm A finds that the w-index is at most $n-1$, we stop with answer NO. If algorithm A finds that the w-index is n, we stop with the answer YES.

If there was an algorithm A for the w-index with worst case time complexity better than $O(n \log n)$, this approach would yield an algorithm for Permutation-Recognition with worst case time complexity better than $O(n \log n)$. $\square$

6 Conclusion

In this paper we have discussed the algorithmic complexity of computing the h-index, the g-index, and the w-index in various (standard) models of computation. Our results suggest that the h-index is computationally the easiest index to compute, that the g-index needs some more effort, and that the w-index is the hardest.

We note that our techniques can easily be adapted to yield similar results for other scientific impact indices. Consider for instance the so-called *Kosmulski-index* of a citation sequence (see Kosmulski, [9]): A scientist has Kosmulski-index k, if k is the largest integer such that at least k of his articles have received at least k^2 citations each. An equivalent definition of the Kosmulski-index of a citation sequence $x = \langle x_1, \ldots, x_n \rangle$ is as follows: Define an auxiliary citation sequence $y = \langle y_1, \ldots, y_n \rangle$ by setting $y_i = \lfloor \sqrt{x_i} \rfloor$ for $1 \leq i \leq n$. Then the Kosmulski-index of sequence x coincides with the h-index of sequence y. Now our results on the h-index imply that

- the Kosmulski-index of a sorted citation sequence $x_1 \geq x_2 \geq \cdots \geq x_n$ can be determined in $O(\log_2 n)$ time,

- the Kosmulski-index of an unordered citation sequence in random access memory can be determined in $O(n)$ time,
- the Kosmulski-index of an unordered citation sequence in sequential memory can be determined in $O(n)$ time.

Furthermore, these time complexities are best possible in the respective models of computation. This once again confirms our intuition that the Kosmulski-index is very closely related to the h-index, and that these two indices behave in more or less the same way.

References

[1] Aho, A. V., Hopcroft, J. E. and Ullman J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Ben-Or M., Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing (STOC'1983)*, pages 80–86, 1983 .

[3] Blum, M., Floyd, R. V., Pratt, V. R., Rivest, R. L. and Tarjan, R. E. Time bounds for selection. *Journal of Computer and System Sciences* 7:448–461, 1973.

[4] Cormen, T. H., Leiserson, C. E. and Rivest R. L. *Introduction to Algorithms.* MIT Press, 1990.

[5] Egghe, L. An improvement of the h-index: The g-index. *ISSI Newsletter* 2:8–9, 2006.

[6] Egghe, L. Theory and practice of the g-index. *Scientometrics* 69:131–152, 2006.

[7] Gross, P. L. K. and Gross, E. M. College libraries and chemical education. *Science* 66(1713):385–389, 1927.

[8] Hirsch, J. E. An index to quantify an individual's scientific research output. *Proceedings of the National Academy of Sciences* 102(46):16569–16572, 2005.

[9] Kosmulski M. A new Hirsch-type index saves time and works equally well as the original h-index. *ISSI Newsletter* 2:4–6, 2006.

[10] Papadimitriou C. H. *Computational Complexity.* Addison-Wesley, 1994.

[11] Woeginger G. J. An axiomatic characterization of the Hirsch-index. *Mathematical Social Sciences* 56:224–232, 2008.

[12] Woeginger G. J. (2008b). An axiomatic analysis of Egghe's g-index. *Journal of Informetrics* 2:364–368, 2008.

Received 6th November 2009