Comment on *Self-Indexed Sort*

Arne Andersson* Andrej Brodnik†

August 8, 1996

In a recent paper [5] Wang presented *Self-Indexed Sort.* Roughly, the algorithm sorts n keys taken from the universe of m elements. We claim that the paper is based on some severe misunderstandings.

- The analysis is wrong.
- The algorithm is not new. It can be found (with correct analysis) in many basic textbooks.
- The presented "testing results" are misleading and they have very little to do with real execution time.

1 Analysis. The author claims the algorithm has $O(n)$ and $O(m+n)$ time and space complexities respectively. The space complexity is indeed $O(m+n)$, but the time complexity is in fact $\Theta(n+m)$. To see this, consider the algorithm's three phases (cf. [5, Algorithm 1]):

1. initialization;
2. self-indexed arrangements;
3. order-preserved compression.

The first phase initializes the m-element array `SS` to 0. This takes time $\Theta(m)$. The second phase contains a loop running through the n-element array `X` containing keys which are to be sorted. This takes time $\Theta(n)$. The last phase again contains a loop running through the whole array `SS` while placing keys back into `X`. This takes time $\Theta(n+m)$.

The first phase could be avoided by using back pointers ([1, exercise 2.12]). However, this would not help in the third phase. Hence, the cost of *Self-Indexed Sort* is $\Theta(n+m)$.

2 A Textbook algorithm. The author claims that the algorithm is new. However, the same algorithm (but with correct analysis) can be found in many basic text books. Aho, Hopcroft and Ullman call the algorithm *bucket sorting* ([1, pp. 77–78]) or *bin sorting* ([2, pp. 274–277]). Cormen, Leiserson, and Rivest call the algorithm *counting sort* ([3, pp. 176–177]). They also mention that it is stable. Sedgewick calls the algorithm *distribution*

*Lund University, Lund, Sweden

†Luleå University, Luleå, Sweden, and IMFM, Ljubljana, Slovenia

counting ([4, pp. 111–113]). He also mentions that the algorithm is stable. Weiss calls the algorithm *bucket sort* ([6, p. 243]).

There are minor differences in implementation between the textbook algorithms. The large array of length m can be used in two ways: either it contains linked lists of the corresponding records or it contains counters of how many keys of each kind occurs. The first alternative is used by Aho, Hopcroft and Ullman. The second alternative is used by Cormen, Leiserson, and Rivest, by Sedgewick, and by Weiss.

Wang in Algorithm 1 uses both counters and linked lists at the same time. This gives no advantage. In fact, by simplifying Wang's algorithm – removing the use of one of the two methods and an appropriate change in the code – we would end up with a textbook algorithm.

Indeed, in Algorithm 2 Wang drops linked lists but for a different reason: the author assumes the elements contain only a key and the elements with the same key are indistinguishable. In this case, there is no need to keep any other information than the number of keys of each kind. This possibility is folklore, as an example it is described by Weiss [6, p. 243]. And, the complexity is still $\Theta(n+m)$.

3 "Testing results". The author gives the impression that the algorithm has been implemented and tested. However, the measured quantity is nonsignificant. The "cost" of his own algorithm is claimed to be exactly $2n$. It seems like the measured cost is either the number of moves back and forth from the array `SS` or the author assumes $m = n$. In reality we are dealing with 16-, 24-, 32-, or even 64-bit keys (and thus also this large integers) and the latter assumption is incorrect. Therefore, if m is large comparing to n (which is true for Wang's testing cases – cf. [5, Table 2]) the cost of scanning the entries in `SS` will be much higher than that of moving records. Each entry in `SS` is visited at least twice so if we add these visits to our measure the cost would be $2n + 2m$. To see the difference this makes, consider sorting 24-bit integers. Then, m would be $2^{24} = 16777216$ and the tables and diagrams would look very different. For $n = 100$ the "cost" of Self-Indexed Sort would be $2n + 2m = 33554632$ compared to the cost 200 claimed by Wang.[1] In fact, the $2m$ term would dominate the cost even for n as large as hundreds of thousands.

In summary, the "testing results" do not reflect real running time.

Final comment. It is true that in many applications (maybe even in most applications), algorithms based on indirect addressing are faster than comparison-based algorithms. This is known by many researchers but not always made fully clear in textbooks. Indeed, the majority of algorithms described in the textbooks are comparison based and thus people quite often overlook these indirect addressing based algorithms. In any case, we recommend people interested in developing fast practical algorithms to take a careful look into a couple of textbooks on algorithms and data structures. We cite Mark Weiss ([6, p. 243]):

> *Although bucket sort seems like much too trivial an algorithm to be useful, it turns out that there are many cases where the input is only small integers, so that using a method like quicksort is overkill.*

[1] We do not claim that our numbers can be translated directly into CPU time, but it would be closer to the truth than Wang's.

As a matter of fact, even when the range of the input is large, there are algorithms based on indirect addressing which runs significantly faster than quicksort, both with respect to asymptotic complexity and CPU time. However, an overview of this area is beyond the scope of this note.

References

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

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, Reading, Massachusetts, 1987.

[3] Th.H. Cormen, Ch.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[4] R. Sedgewick. *Algorithms.* Addison-Wesley, Reading, Massachusetts, 1988.

[5] S.Y. Wang. A new sort algorithm: Self-indexed sort. *ACM SIGPLAN Notices*, 31(3), March 1996.

[6] M.A. Weiss. *Data Structures and Algorithm Analysis.* The Benjamin/Cummings Publishing Company, Redwood City, California, 1992.