MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS


Jesper Sindahl Nielsen
Aarhus University

Danmarks Grundforskningsfond
Danish National Research Foundation

Integer Sorting

Problem definition

We are given an array, A, of n integers using w bits each where w is the word size of the Random Access Machine (RAM) and we want to sort them.

There are two parameters for this problem: the input size and the word size. The problem has been extensively studied but there are still many venues to be explored. The following table summarizes the state of art.

Word size	Space usage	Running time	Algorithm
O(log n)	O(n) words	O(n)	Radix sort, deterministic
Any	$O(2^w) = O(n)$ if w = log n	$O(n \log w + 2^w) = O(n \log \log n)$ If w = log n	van Emde Boas tree, deterministic
$\geq O(\log^{2+\varepsilon} n)$	O(n)	O(n)	[1] randomized
Any	O(n)	O(n log log n)	[4] deterministic
Any	O(n)	O(n $\sqrt{\log \log n}$) expected	[5] randomized
$\geq O(\log^2 n \log \log n)$	O(n)	O(n) expected	Our contribution

Question

The primary question is, can we sort in linear time for all word sizes? Our contribution is a step towards this goal.

One may ask whether sorting integers is too restrictive. However any sorting algorithm on integers will also work for floating point numbers due to the way they are represented in memory [1].

Idea

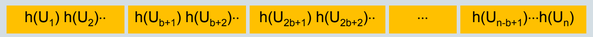
Our idea is to sort the integers using a compressed trie of the input where letters are w/2 bits long. In a compressed trie we only care about the branching nodes, so we will find these by hashing the w/2 most significant bits. With high probability if two elements hash to the same value, then they share the same w/2 most significant bits – in which case we have a branching node.

Note, that to determine the order between two elements that share the same w/2 most significant bits, we need only look at their w/2 least significant bits, thus we can essentially shrink the original integers.

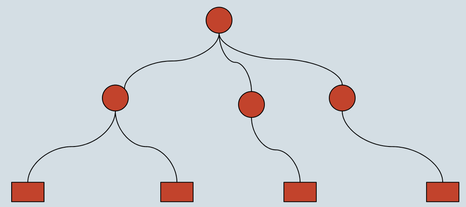
Finding branching nodes

We start with our input, and we take out the w/2 most significant bits of each integer. Let A_i be the i'th input and U_i be its w/2 most significant bits.

Next we hash each element in this array to log n bits and pack all the hashes tightly into n·log n/w words. Let b be the number of hashes pr word.



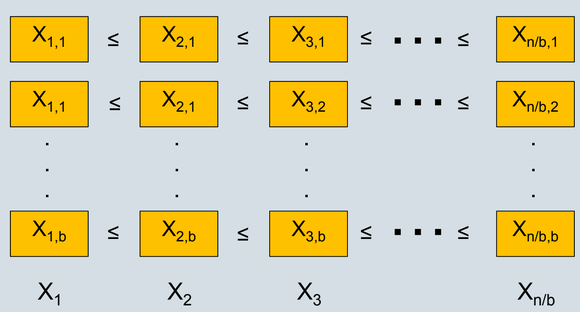
Note that we now use n/b words – and it turns out it that we can sort n/b words packed with integers fast as explained below. We sort the hashes to find out which elements had the same w/2 most significant bits.



We know from the sorted hashes that the first child of the root is branching. The second and third child will have a unique hash with high probability, so their order is totally given by the w/2 most significant bits therefore We can throw away the least significant bits for those two elements. The procedure can be applied recursively. Now we just need to sort log n bit integers packed in words.

Sorting packed integers

In the first phase of our algorithm we want to permute elements such that the first element in the first word is less than the first element in the second word which is less than the first element in the third word and so on. Equivalently we want this for the second element in each word, third element and so on:



Sorting packed integers (continued)

To do this we implement a sorting network that works in parallel on the b tracks. A sorting network is a fixed sequence of comparisons and swaps where the comparison determines whether or not to swap the two compared elements. Our input is N=n/b words where there are b integers pr word.

- Construct a network C for input size N
- When C needs to compare index i and j, we use bit parallelism to compare $x_{i,k}$ with $x_{j,k}$ for $1 \leq k \leq b$ in constant time.
- Based on comparison output we can swap individual elements

Implementing sorting networks in RAM

Suppose each element has an extra most significant bit. If we want to compare word i and j, then set all these extra bits to 1 in word i and subtract the two words:

Then $z_k = 0$ if and only if $x_{i,k} < x_{i,k}$. Based on the result word we can mask out the elements that are larger and swap the smaller elements and vice versa.

Considering the image on the bottom to the left: we do not have any order inside words. Consider b consecutive words as a b×b matrix. If we transpose this matrix, we get b words that are internally sorted. Do this transposition on every consecutive block of b words. To get b sorted lists where each word is sorted, we concatenate the first word of every block in order, then the second word of every block in order and so on.

We have b sorted sequences and the last step is to merge them. We merge by applying bitonic sorting and merging. This takes $\log^2 b$ time pr word. There are initially b lists thus there will be O(log b) rounds of merging.

References

[1] Arne Andersson, Torben Hagerup, Stefan Nilsson, Rajeev Raman. *Sorting in Linear Time?* J. Comput. Syst. Sci. 57(1), 1998.

[2] Mikkel Thorup. *Randomized sorting in* O(*n* log log *n*) *time and linear space using addition, shift, and bit-wise boolean operations. J. Algorithms* . 42(2), 2002.

[3] Michael T. Goodrich. *Randomized Shellsort: A Simple Data-Oblivious Sorting Algorithm*. J. ACM 58(6), 2011.

[4] Yijie Han. *Deterministic sorting in* O(*n* log log *n*) *time and linear space*. STOC 2002.

[5] Yijie Han, Mikkel Thorup. *Integer Sorting in* 0(*n* √log log *n*) *Expected Time and Linear Space*. FOCS 2002.