Theoretical Computer Science 103 (1992) 137–141
Elsevier

Sorting the sums (x_i+y_j) in $O(n^2)$ comparisons*

Jean-Luc Lambert

Laboratoire d'informatique de Paris-Nord, Département de Mathématiques et Informatique, Centre scientifique et polytechnique, Université de Paris-Nord, Avenue Jean-Baptiste Clément, 93 430 Villetaneuse, France

Abstract

Lambert, J.-L., Sorting the sums (x_i+y_j) in $O(n^2)$ comparisons, Theoretical Computer Science 103 (1992) 137–141.

Let $(x_i)_{1\leqslant i\leqslant n}$ and $(y_j)_{1\leqslant j\leqslant n}$ be two sequences of numbers. It was proved by Fredman (1976) that the n^2 sums $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ can be sorted in $O(n^2)$ comparisons, but until now, no explicit algorithm was known to do it. We present such an algorithm and generalize it to sort $(x_{1,i_1}+\cdots+x_{k,i_k})_{1\leqslant i_1,\ldots,i_k\leqslant n}$ in $O(n^k)$ comparisons. Unfortunately, none of these algorithms is efficient in practice.

0. Introduction

Let $(x_i)_{1\leqslant i\leqslant n}$ and $(y_j)_{1\leqslant j\leqslant n}$ be two sequences of numbers. The problem of sorting the n^2 sums $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ in the smallest number of comparisons was posed by E. Berlekamp and first studied by Harper et al. [4], who proved that $n^2\log_2 n$ comparisons are sufficient and even necessary if we just sort $(x_i)_{1\leqslant i\leqslant n}$ and then $(y_j)_{1\leqslant j\leqslant n}$ and use only the fact that $x_i\leqslant x_i'$, $y_j\leqslant y_j' \Rightarrow x_i+y_j\leqslant x_i'+y_j'$.

Fredman [3] proved a result that seems to contradict the previous one: there exists a decision tree for this problem whose depth is $O(n^2)$. This implies that $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ can be sorted in $O(n^2)$ comparisons. But his result is nonconstructive and until now no explicit algorithm was known to generate the comparisons sequence corresponding to such a decision tree. Dietzfelbinger [2] recently proved that this upperbound is optimal.

We present such an algorithm. More precisely, we exhibit an algorithm which, given $2n$ numbers $(x_i)_{1\leqslant i\leqslant n}$ and $(y_j)_{1\leqslant j\leqslant n}$, sorts $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ in $O(n^2)$ *comparisons.*

*This work was partially realized at Université de Paris-Sud, Centre d'Orsay, Laboratoire de recherche en informatique, CNRS UA 410, 91405 Orsay France.

0304-3975/92/$05.00 © 1992—Elsevier Science Publishers B.V. All rights reserved

Then we generalize the algorithm to sort $(x_{1,i_1}+\cdots+x_{k,i_k})_{1\leqslant i_1,\ldots,i_k\leqslant n}$ in $O(n^k)$ comparisons. Unfortunately, the additional complexity due to the replacement of comparisons by other calculations makes this algorithm not efficient in practice.

The first version of this paper was presented at the colloquium STACS 90 [5]. Arnold [1] noticed that a recursive version of our algorithm exists. We present here such a version and this is the reason why this article differs appreciably from the original one.

1. Sorting $(x_i+y_j)_{1\leqslant i,j\leqslant n}$

By the classical algorithms, the sorting of $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ needs $O(n^2\log_2 n)$ comparisons. To diminish the number of comparisons, we need to use the underlying arithmetical structure. The main idea is given by the following lemma.

Lemma 1.1. *Let R and S be two finite sets of numbers; then if $R+R$ and $S+S$ are sorted then $R+S$ can be sorted in at most* $(\text{card } R)^2+(\text{card } S)^2-1$ *comparisons.*

Proof. We use the trivial equivalence

$$r+s\leqslant r'+s' \Leftrightarrow r-r'\leqslant s'-s.$$

We can then deduce with no comparison the orderings of

$$R-R=\{r-r',\ (r,r')\in R^2\}$$

and of

$$S-S=\{s-s',\ (s,s')\in S^2\}$$

from the orderings of $R+R$ and $S+S$. Merging these two sets in at most $(\text{card } R)^2+(\text{card } S)^2-1$ comparisons, we get the ordering of the set $(R-R)\cup(S-S)$. But using the same identity, we deduce from it, with no comparison, the ordering of the set $R+S$. □

The interest of that lemma is the clearest when $\text{card } S=\text{card } R=n$; then we can sort $R+S$ in $O(n^2)$ comparisons, which is just our problem.

Before sorting $R+S$ we must sort $R+R$ and $S+S$, but this can be done in the same manner by using in a recursive way the result of Lemma 1.1.

Lemma 1.2. *Let X be a finite set of n numbers; we can sort $X+X$ in $O(n^2)$ comparisons.*

Proof. Let card $X=n$ and $X=\{x_1,\ldots,x_n\}$. We set

$$R=\{x_1,\ldots,x_{\lceil\frac{n}{2}\rceil}\},\qquad S=\{x_{\lceil\frac{n}{2}\rceil+1},\ldots,x_n\}.$$

Then $X+X=(R+R)\cup(S+S)\cup(R+S)$.

We sort by recursion $R+R$ and $S+S$; then by Lemma 1.1, $R+S$ may be sorted in $\mathrm{O}(n^2)$ comparisons. It only remains to merge $R+R$, $S+S$ and $R+S$ to get $X+X$ sorted, and this can be done in $\mathrm{O}(n^2)$ comparisons.

If $C(n)$ is the number of comparisons needed to sort $X+X$ by this algorithm, we have the recursive equation

$$C(n)=C(\lceil \tfrac{n}{2}\rceil)+C(\lfloor \tfrac{n}{2}\rfloor)+\mathrm{O}(n^2), \quad C(1)=0.$$

Solving this equation yields

$$C(n)=\mathrm{O}(n^2). \qquad \square$$

Using Lemmas 1.1 and 1.2, we conclude easily the following theorem.

Theorem 1.3. *There exists an algorithm which sorts $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ in $\mathrm{O}(n^2)$ comparisons.*

We sum up here our algorithm to sort $X+Y$.

```
Procedure Sorting($X+Y$)
    Sort($X+X$)
    Sort($Y+Y$)
    Sort $X-X$ and $Y-Y$ with no comparison
    Merge($X-X$, $Y-Y$)
    Sort $X+Y$ with no comparison
```

```
Procedure Sort($X+X$)
    $R:=\{x_1,\ldots,x_{\lceil n/2\rceil}\}$
    $S:=\{x_{\lceil n/2\rceil+1},\ldots,x_n\}$
    Sort($R+R$)
    Sort($S+S$)
    Sort $R-R$ and $S-S$ with no comparison
    Merge($R-R$, $S-S$)
    Sort $R+S$ with no comparison
    Merge($R+R$, $S+S$, $R+S$)
```

This algorithm sorts $(x_i+y_j)_{1\leqslant i,j\leqslant n}$ in $\mathrm{O}(n^2)$ comparisons, but the comparisons are replaced by a procedure which, given an ordered set $R+R$, sorts $R-R$ (or the contrary) with no comparison. The only strategy we know for doing that is to use a classical sorting algorithm on $R-R$ but instead of making a comparison between $r-r'$ and $s-s'$, we just look at the result of the comparison between $r+s'$ and $s+r'$, which is known in $R+R$.

The complexity of this procedure is $\mathrm{O}((\operatorname{card} R)^2\log_2(\operatorname{card} R))$, which gives a complexity of $\mathrm{O}(n^2\log_2 n)$ elementary operations for the original problem. To get an algorithm in $O(n^2)$ operations we would need a procedure which runs in linear time (i.e. $O((\operatorname{card} R)^2)$ operations).

2. A generalization: sorting $(x_{1,i_1}+\cdots+x_{k,i_k})_{1\leqslant i_1,\ldots,i_k\leqslant n}$ in $O(n^k)$ comparisons

The generalization is based on the following generalization of Lemma 1.1.

Lemma 2.1. *Let R, S and T be three finite sets of numbers, then if $R+R+T$ and $S+S+T$ are sorted then $R+S+T$ can be sorted in at most* $(\operatorname{card} R)^2 \operatorname{card} T+(\operatorname{card} S)^2 \operatorname{card} T-1$ *comparisons.*

Proof. We use the following equivalence:

$$r+s+t\leqslant r'+s'+t' \Leftrightarrow r-r'+t\leqslant s'-s+t'.$$

Thus, we can sort

$$R-R+T=\{r-r'+t,\ (r,r',t)\in R\times R\times T\},$$

$$S-S+T=\{s-s'+t,\ (s,s',t)\in S\times S\times T\}$$

with no comparison, and we merge the two sets in $(\operatorname{card} R)^2 \operatorname{card} T+(\operatorname{card} S)^2 \operatorname{card} T-1$ comparisons. At last we deduce from that the ordering of $R+S+T$ with no additional comparison. □

As in the previous section, we deduce from Lemma 2.1 the sorting of the set $X+X+T$.

Lemma 2.2. *Let X be a finite set of numbers, then we can sort $X+X+T$ in* $O(\max((\operatorname{card} X)^2 \operatorname{card} T, \operatorname{card} T\log_2(\operatorname{card} T)))$ *comparisons.*

Proof. Let $\operatorname{card} X=n$, $X=\{x_1,\ldots,x_n\}$. We set

$$R=\{x_1,\ldots,x_{\lceil\frac{n}{2}\rceil}\},\qquad S=\{x_{\lceil\frac{n}{2}\rceil+1},\ldots,x_n\}.$$

We use the same algorithm as in Lemma 1.2 just replacing $R+R, S+S$ and $R+S$ by $R+R+T, S+S+T$ and $R+S+T$. Let $C(n)$ be the number of required comparisons; we get the recursive equation

$$C(n)=C(\lceil\tfrac{n}{2}\rceil)+C(\lfloor\tfrac{n}{2}\rfloor)+O(n^2 \operatorname{card} T).$$

But this algorithm would sort T n times (when sorting the sets $\{x\}+\{x\}+T$).

Since T does not need to be sorted n times, we suppose that T was first sorted in $\operatorname{card} T\log_2(\operatorname{card} T)$ comparisons and then $C(1)=0$, which yields now

$$C(n)=O(n^2 \operatorname{card} T).\qquad \square$$

Using these two lemmas and choosing $T = \{(x_{3,i_3} + \cdots + x_{k,i_k})_{1 \leqslant i_3, \ldots, i_k \leqslant n}\}$ so that $\operatorname{card} T = n^{k-2}$, we get the following theorem.

Theorem 2.3 *There exists an algorithm which sorts* $(x_{1,i_1} + \cdots + x_{k,i_k})_{1 \leqslant i_1, \ldots, i_k \leqslant n}$ *in* $O(n^k)$ *comparisons.*

3. Conclusion

Our algorithms are inefficient because of the transformations of sets of sums into sets of subtractions and vice versa. These transformations can be practically suppressed in the algorithms (we can restrict ourselves to make only comparisons between sums), but we always need to compute an equivalent information to determine which comparisons have to be done. An open problem is to find, if it exists, an algorithm which effectively sorts the sums in $O(n^2)$ operations.

References

[1] A. Arnold, private communication.

[2] M. Dietzfelbinger, Lower bounds for sorting sums, *Theoret. Comput. Sci.* **66** (1989) 137–155.

[3] M.L. Fredman, How good is the information theory about sorting?, *Theoret. Comput. Sci.* **1** (1976) 355–361.

[4] L.H. Harper, T.H. Payne, J.E. Savage and E. Straus, Sorting $X + Y$, *Comm. ACM* **18** (6) (1975) 347–349.

[5] J.L. Lambert, Sorting the sums $(x_i + y_j)$ in $O(n^2)$ comparisons, in: *Proc. STACS 90*, Lecture Notes in Computer Science, Vol. 415 (Springer, Berlin, 1990) 195–206.