ON SORTING NETWORKS

by Marvin Paull and Saul Levy

Department of Computer Science

Livingston College
Rutgers University
The State University of New Jersey
New Brunswick, New Jersey 08903

This paper is an expanded version
of a paper presented at the 1968
Princeton University Conference.

Department of Computer Science Technical
Report #21 - November, 1972

ON SORTING NETWORKS[1]

Many algorithms for internal computer sorting can be modelled by comparator networks. A comparator network is a (non feedback) switching network built of two-input by the two-output gates whose inputs are drawn from some completely ordered set (for convenience and with no loss of generality in our discussion we take that set to be the non-negative integers), and whose outputs correspond to the maximum of the two inputs on one lead and the minimum on the other (Fig. 1). A network consisting exclusively of this type of element can, of course, represent only a subset of the comparator algorithms which could be carried out by a general purpose computer. It cannot, for example, describe algorithms where the comparison tree is altered and pruned as more information becomes available about the ordering of the inputs.

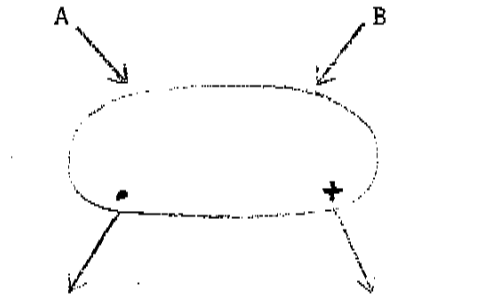


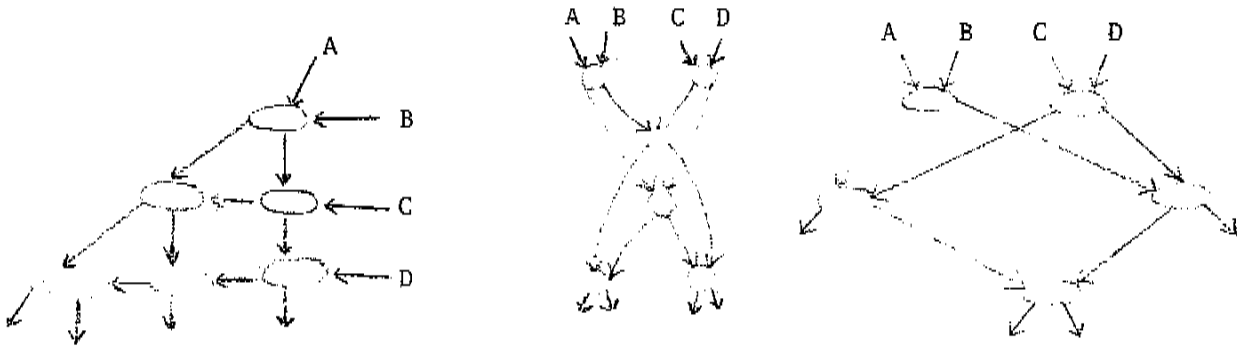
Figure 1

If you examine the 3 4-sort algorithms in figure 2, you'll see that first of all they are substantially different (i.e. not isomorphic) not just simple permutations one of the other, in fact algorithm (c) requires

1 This paper is an expanded version of a paper presented at the 1968 Princeton University Conference.

fewer comparisons than do the others. If you examine algorithm (a) you will see that it sorts by induction -- working down from the top, the outputs of any horizontal row are a sorted sequence and the sole job of the following row is just to insert the next input in its proper position.

The scheme behind algorithms b and c is not so clear -- they do, as we have stated, sort. It became clear early that we needed to have some tool for testing or studying prospective algorithms in order to determine their properties.



(a) (b) (c)

Figure 2

In this paper we will discuss a simple algebra which can be used to describe the workings of these networks. It is a good tool for their analysis and an interesting (if somewhat less effective) tool for their synthesis. It has two operations, minimum (represented by.) and maximum (represented by +). After a brief introduction to some of the formal properties of our operator it will be easy to show that the value of the symmetric function Σ_i (the sum (or maximum) of all the products (or minima) of i distinct variables) is the i^{th} largest element, as well as other results relevant to the use of these operators for sorting.

II. Formal Properties

We have defined a·b to be the minimum of a and b and a+b to be the maximum of a and b. As is the custom in most branches of mathematics the "·" will be left out when no confusion can arise. Max and min are quite obviously:

A) Commutative

$ab = ba$

$a+b = b+a$

B) Idempotent

$aa = a$

$a+a = a$

C) Associative

$a(bc) = (ab)c$

i.e., $\min(a,\min(b,c)) = \min(\min(a,b),c)$

$a+(b+c) = (a+b)+c$

i.e. $\max(a,\max(b,c)) = \max(\max(a,b),c)$

D) Consistent

$x \leq y$ iff $x \cdot y = x$ iff $x+y = y$

i.e., $x \leq y$ iff $\min(x,y) = x$ iff $\max(x,y)=y$

E) Distributive

$x(y+z) = xy+xz$

To show this we consider the six possible orderings of x,y,z:

ordering	x(y+z)	xy+xz
$x \le y \le z$	x	x
$x \le z \le y$	x	x
$y \le x \le z$	x	x
$y \le z \le x$	z	z
$z \le x \le y$	x	x
$z \le y \le x$	y	y

In a like manner we show $x+yz = (x+y)(x+z)$

ordering	x+yz	(x+y)(x+z)
$x \le y \le z$	y	y
$x \le z \le y$	z	z
$y \le x \le z$	x	x
$y \le z \le x$	x	x
$z \le x \le y$	x	x
$z \le y \le x$	x	x

F) Absorptive $x(x+y) = x+xy = x$

ordering	x(x+y)	x+xy
$x \le y$	x	x
$y \le x$	x	x

III. Results

Our first theorem has been discovered independently by Floyd and Knuth (and probably others) as well as ourselves.

In the sequel we will use the name (or symbol) for a function to stand for its value when no confusion can result.

Definition: A product of literals will be called a phrase.

Theorem 1 The value of Σ_i the sum of all the products of i distinct variables is the i^{th} largest element of the input set.

For example: Consider sorting three elements

a,b,c.

Our theorem states that:

the largest of the three = a+b+c

the next largest = ab+ac+bc

the smallest = abc

Proof Assume we are sorting n numbers $x_1, x_2, \ldots x_n$. Assume further that their order is

$$x_{i_1} < x_{i_2} < \ldots < x_{i_n}$$

$\Sigma_1 = x_{i_1} + x_{i_2} + \ldots + x_{i_n}$ which equals x_{i_n} the largest element.

In general consider the value of Σ_j.

There are $\binom{n}{j}$ phrases of length j.

None of the j-1 largest elements can be the value of any of the $\binom{n}{j}$ phrases in Σ_j because each phrase in Σ_j has j elements one of which must be less than the j-1 largest elements. On the other hand one phrase consists of the product of j largest elements and its value is the j^{th} largest element, $x_{i_{n-j+1}}$, (the value of a phrase is the value of its smallest element). Every other phrase contains some element other than those in the j largest and consequently has value less than $x_{i_{n-j+1}}$.

Therefore the value of Σ_j is $x_{i_{n-j+1}}$.

There exists an analogous result for the case of Π_j, the product of all the i-sums. The value of Π_i is the i^{th} smallest element. The proof is similar to the proof for the Σ_i form.

Proof

(1) $\Pi_1 = \Sigma_n$ = least element

(2) Consider Π_i. The value of Π_i is the minimum value taken on by all the i-sums. One of the i sums consists of the smallest i elements and its value is the i^{th} element. All other sums have at least one element larger than the i^{th} and consequently take a greater value.

Note that all of our subsequent results have an analogous Π-form as well.

Because the number of comparisons required to sort a set increases faster than linearly, in many algorithms for sorting we find that an economical procedure involves breaking our set arbitrarily into equal sized smaller sets and then recombining or merging the smaller sorted sets. The number of such subsets depends generally on the cost of the recombinations. However, theorem 1 yields the following corollary.

Corollary 1: Suppose that we have a set A of n elements to be sorted and that we break the set of n elements into m equal disjoint sets $A_1, A_2, \ldots, A_m$ such that

$$A_1 \cup A_2 \cup \ldots A_m = A \qquad A_i \cap A_j = \phi \quad i \neq j$$

Then using the notation $A^i = \Sigma_i$ with the elements from A, we have the i^{th} largest element

$$= \sum_{j_1 + j_2 + \ldots + j_m = i} A_1^{j_1} A_2^{j_2} \ldots A_m^{j_m}$$

$$= \sum a_{1,\frac{n}{m} - j_1 + 1} \quad a_{2,\frac{n}{m} - j_2 + 1} \ldots a_{m,\frac{n}{m} - j_m + 1}$$

where $a_{i,k}$ is the kth element of A_i, eg., $a_{i,1}$ is the smallest element in A_i.

Corollary 2: If the elements from the sets are combined then:

1) $a_{1,i_1} + a_{2,i_2} + \ldots + a_{n,i_n} \geq a_{i_1 + i_2 + \ldots + i_n}$

2) $a_{1,i_1} \cdot a_{2,i_2} \cdot \ldots \cdot a_{n,i_n} \leq a_{i_1 + i_2 + \ldots + i_n - n + 1}$

Proof: 1) Suppose the largest element of the $a_{i_1} \ldots$ is a_{j,i_j} we observe that

$a_{j,i_j} \geq i_1$ elements of $A_1 + i_2$ elements of A_2

$+ \ldots + i_j$ elements of $A_j + \ldots + i_n$ elements of A_n

QED.

2) Suppose the smallest element out of $a_{1,i_1}, a_{2,i_2}, \ldots$ is a_{j,i_j} we observe that a_{i,i_j} may still be larger than i_1-1 elements of $a_1 + \ldots$

$$= \sum_{k=1, k\neq j}^{n} i_k - 1 + i_j$$

$$= \Sigma i_k - n + 1$$

Before we can apply theorem 1 effectively to synthesis and analysis problems we require a little more formal assistance.

In the following material lower case Greek letters will be used to denote phrases and upper case Greek letters will denote comparator functions.

Lemma 1 $\alpha \cdot \beta \leq \alpha$

Proof Consider the values of α, β. Either $\alpha \leq \beta$ or $\alpha > \beta$.

Case 1 $\alpha \leq \beta$. Then $\alpha \cdot \beta = \alpha$

Case 2 $\alpha > \beta$. Then $\alpha \cdot \beta = \beta < \alpha$

Lemma 2 $\alpha + \beta \geq \alpha$

Proof Consider the values of α and β. Either $\alpha \geq \beta$ or $\alpha < \beta$

Case 1 $\alpha \geq \beta$. Then $\alpha + \beta = \alpha$

Case 2 $\alpha < \beta$. Then $\alpha + \beta = \beta > \alpha$

Definition A phrase α is said to subsume a phrase β if every literal in β is also a literal in α. Thus by Lemma 1 α subsumes β implies $\alpha \leq \beta$.

Definition We take for the normal form of a comparator function, the sum of product form. Our distributive property guarantees us that we can always put our expressions into this form. (Note also that essentially the same results as we obtain could have been derived if we had instead assumed the product of sums form.)

Definition Reduced normal form (RNF) is the form obtained from normal form by eliminating all subsuming phrases.

We shall prove that this reduced normal form is unique for each comparator function. Assume Φ and Ψ are expressions in normal form.

Lemma 3 $\Phi \leq \Psi$ iff each phrase of Φ subsumes a phrase of Ψ.

I. sufficiency

Suppose each phrase of Φ subsumes a phrase of Ψ. Then each phrase in Φ is less or equal a phrase in Ψ. Since the values of Φ, Ψ are determined by the maximum value of their phrases and since to each value taken on by a phrase in Φ there corresponds a value at least as large taken on by a phrase in Ψ, the value of $\Psi \geq$ value of Φ.

II. necessity

Suppose some phrase $\phi_j = a_1 a_2 \cdots a_n \in \Phi$ subsumes no phrase in Ψ. Then every phrase in Ψ contains some literal other than $a_1, a_2, \ldots, a_n$.

Assign the value 2 to each of $a_1, a_2, \ldots, a_n$ and the value 1 to all other literals in Φ and Ψ. The value of $\phi_j = 2$ and the value of all other phrases in Φ and in Ψ equal 1 (since each contains at least one literal with value 1). Therefore the value of $\Phi = 2$ and the value

of Ψ equals 1 and $\Psi < \Phi$ contradicting our hypothesis. Therefore every phrase in Φ must subsume some phrase in Ψ .

Theorem 2 There is precisely one RNF for each comparator function.

Assume $\Phi = \Psi$ and both Φ , Ψ are in reduced normal form. Then each phrase in Φ subsumes some phrase in Ψ and each phrase in Ψ subsumes some phrase in Φ.

Suppose ϕ_i in Φ subsumes ψ_i in Ψ and ψ_i in turn subsumes ϕ_j in Φ $j \neq i$. Then ϕ_i subsumes ϕ_j contradicting our assumption that Φ is in reduced normal form.

Therefore ϕ_i subsumes $\psi_i \leftrightarrow \psi_i$ subsumes ϕ_i, that is ϕ_i is identical to ψ_i. Since each phrase in both Φ and Ψ subsumes a phrase in the other, each must have an identical correspondent in the other and Φ and Ψ being the sum of identical phrases are identical.

The next theorem is a generalization of a theorem by Knuth which states that if a sorting network sorts all n tuples of 0's and 1's it will sort all n-tuples of numbers. Our theorem says that this is true not only for all sorting functions but for any function which can be realized by a network of comparators.

Theorem 3 For each n the number of comparator functions of n variables equals the number of positive Boolean functions of n variables and further each comparator function agrees (under the interpretation which identifies the truth values 0,1 with the corresponding integers) with the corresponding Boolean function on all n-tuples of 0's and 1's.

The theorem says a number of interesting things.

A function constructed entirely of sorting elements is completely determined by what it does on n-tuples of 0's and 1's.

Intuitively one expects that since a comparator network might have to perform up to n! distinct input to output transformations, one would have n! degrees of freedom in determining the function instead of just 2^n.

Proof of Theorem 3

Consider the following mapping between comparator functions and positive Boolean functions

Comparator	Boolean
+	+
·	·
0	0
1	1

Let us give tabular definitions of the two functions on 0-1 inputs.

a b + = "max"	a b + = "or"
0 0 0	0 0 0
0 1 1	0 1 1
1 0 1	1 0 1
1 1 1	1 1 1

a b · = min	a b · = and
0 0 0	0 0 0
0 1 0	0 1 0
1 0 0	1 0 0
1 1 1	1 1 1

A comparison of the function tables on 0-1 inputs shows that the corresponding operators in both classes are identical (that is if 1 > 0 then Boolean "or" is equivalent to comparator "max" and Boolean "and" is equivalent to comparator min).

To each Boolean function there corresponds at least one comparator function namely the one which looks exactly like it. Suppose there were two

comparator functions C_1 and C_2 which agreed on 0's and 1's but disagreed on some other n-tuple say $\vec{x}$. Without loss of generality assume $C_1(\vec{x}) < C_2(\vec{x})$.

Let $C_2(\vec{x}) = m$. Then there must exist a phrase in $C_2(\vec{x})$, say P, each of whose literals is greater than or equal to m in value. Define a new input tuple x^B (for Boolean) in the following way. Set each x_i in P = 1 and all other $x_i = 0$. Then $C_2(x^B) = 1$ because phrase P = 1 and all other phrases are zero. But since C_1 and C_2 agree on all 0-1 inputs $C_1(x^B) = 1$. Therefore there must be some term in C_1 all of whose literals have value 1. But the only literals with value 1 in x^B are those with value $\geq m$ in $\vec{x}$. Therefore, the value of $C_1(x) \geq m$ contradicting our hypothesis that $C_1(x) < C_2(x) = m$. Therefore if two Comparator functions differ on some x they must differ on some x of 0's and 1's. Therefore the number of comparator functions equals the number of positive Boolean functions and corresponding functions agree on input tuples of 0's and 1's.

IV. Examples

Example 1 Analysis

Prove that the following network sorts

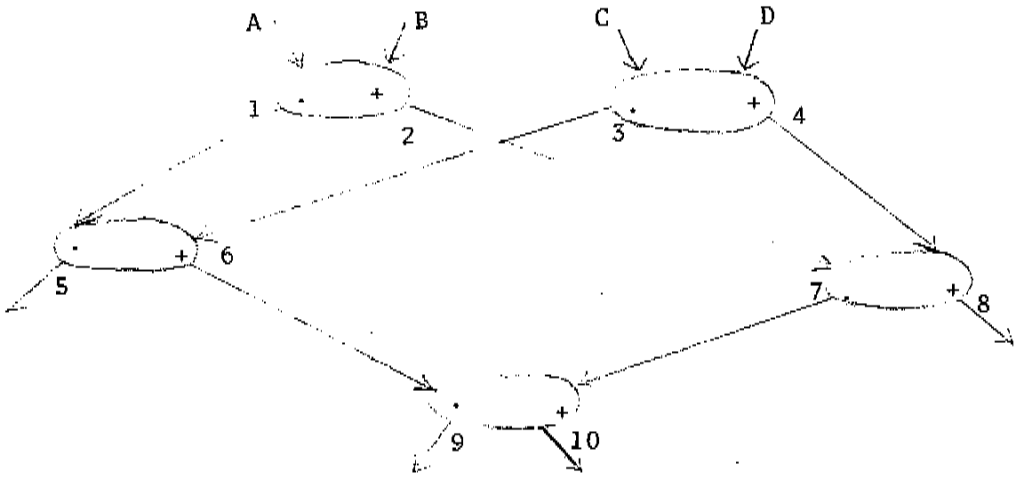


FIGURE 3 4 -SORTER

Calculate values at all points.

1 ab 3 cd

2 a+b 4 c+d 6 ab +cd

5 abcd = Σ_4 7 (a+b)(c+d) = ac+ad+bc+bd

8 a+b+c+d = Σ_1 = maximum

9 abc + abd + acd + bcd = Σ_3 = 2^{nd}

10 ab+ac+ad+bc+bd+cd = Σ_2 = next to largest

and we have shown the network sorts. An attempt to prove this by using maximum and minimum and logic will demonstrate the power of the algebra.

Example 2 Synthesis

Suppose we wish to design a sorter for eight numbers which works by

first dividing the eight inputs into two groups and sorting each of the groups and then merges the two sorted groups. Call the two sorted groups $A = a_1 \leq a_2 \leq a_3 \leq a_4$ $B = b_1 \leq b_2 \leq b_3 \leq b_4$.

Terminology Let A^i = the Σ_1 from elements in A e.g., $A^3 = a_1a_2a_3 + a_1a_3a_4 + a_1a_2a_4 + a_2a_3a_4 = a_2$

To derive the expression for each Σ_i we will take the "sum" of all the ways in which the appropriate length product can be formed. For example, a phrase in Σ_3 can come about in any of four ways:

1) 3 literals from A, 2) 2 literals from A and 1 from B, 3) 1 literal from A and 2 from B, 4) 3 literals from B

Therefore, $\Sigma_3 = A^3 + A^2B^1 + A^1B^2 + B^3$
$= a_2 + a_3b_4 + a_4b_3 + b_2$

$\Sigma_1 = A^1 + B^1 = a_4 + b_4$

$\Sigma_2 = A^2 + A^1B^1 + B^2 = a_3 + a_4b_4 + b_3$

$\Sigma_3 = A^3 + A^2B^1 + A^1B^2 + B^3 = a_2 + a_3b_4 + a_4b_3 + a_2$

$\Sigma_4 = A^4 + A^3B^1 + A^2B^2 + A^1B^3 + B^4$
$= a_1 + a_2b_4 + a_4b_3 + a_4b_2 + b_1$

$\Sigma_5 = a^4B^1 + A^3B^2 + A^2B^3 + A^1B^4 =$
$= a_1b_4 + a_1b_4 + a_2b_3 + a_3b_2 + a_4b_1$

$\Sigma_6 = A^4B^2 + A^3B^3 + A^2B^4$
$= a_1b_3 + a_2b_2 + a_3b_1$

$\Sigma_7 = A^4B^3 + A^3B^4 = a_1b_2 + a_2b_1$

$\Sigma_8 = A^4B^4 = a_1b_1$

As before, if you attempt to verify that the network appearing on the following page (Fig. 4.) sorts using max's, min's and logic you will get a feel for the power of the algebra.

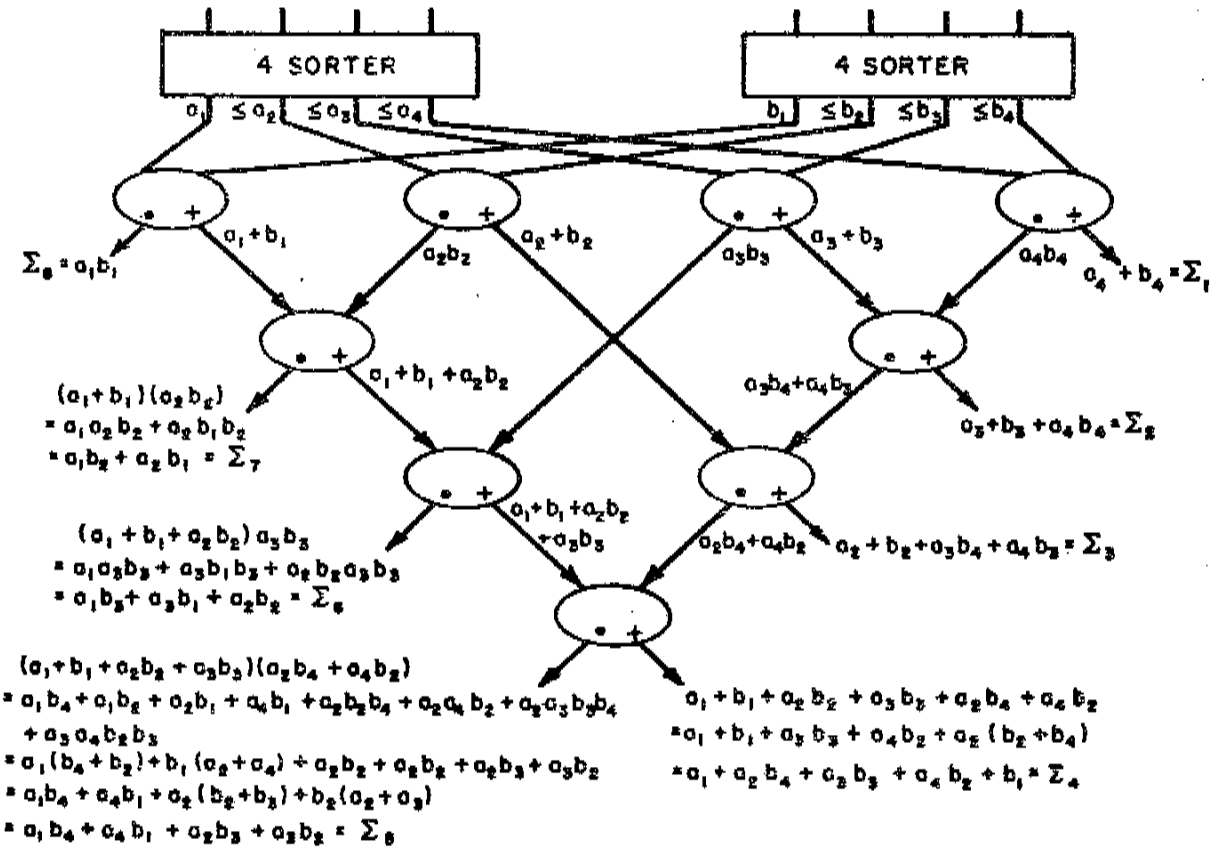

4 SORTER
4 SORTER
$a_1 \le a_2 \le a_3 \le a_4$
$b_1 \le b_2 \le b_3 \le b_4$
$\Sigma_8 = a_1 b_1$
$a_1 + b_1$
$a_2 b_2$
$a_2 + b_2$
$a_3 b_3$
$a_3 + b_3$
$a_4 b_4$
$a_4 + b_4 = \Sigma_1$
$a_1 + b_1 + a_2 b_2$
$(a_1 + b_1)(a_2 b_2)$
$= a_1 a_2 b_2 + a_2 b_1 b_2$
$= a_1 b_2 + a_2 b_1 = \Sigma_7$
$a_3 b_4 + a_4 b_3$
$a_3 + b_3 + a_4 b_4 = \Sigma_2$
$a_1 + b_1 + a_2 b_2 + a_3 b_3$
$(a_1 + b_1 + a_2 b_2) a_3 b_3$
$= a_1 a_3 b_3 + a_3 b_1 b_2 + a_2 b_2 a_3 b_3$
$= a_1 b_3 + a_3 b_1 + a_2 b_2 = \Sigma_6$
$a_2 b_4 + a_4 b_2$
$a_2 + b_2 + a_3 b_4 + a_4 b_3 = \Sigma_3$
$(a_1 + b_1 + a_2 b_2 + a_3 b_3)(a_2 b_4 + a_4 b_2)$
$= a_1 b_4 + a_1 b_2 + a_2 b_1 + a_4 b_1 + a_2 b_2 b_4 + a_2 a_4 b_2 + a_2 a_3 b_3 b_4$
$+ a_3 a_4 b_2 b_3$
$= a_1 (b_4 + b_2) + b_1 (a_2 + a_4) + a_2 b_2 + a_2 b_2 + a_2 b_3 + a_3 b_2$
$= a_1 b_4 + a_4 b_1 + a_2 (b_2 + b_3) + b_2 (a_2 + a_3)$
$= a_1 b_4 + a_4 b_1 + a_2 b_3 + a_3 b_2 = \Sigma_5$
$a_1 + b_1 + a_2 b_2 + a_3 b_3 + a_2 b_4 + a_4 b_2$
$= a_1 + b_1 + a_3 b_3 + a_4 b_2 + a_2 (b_2 + b_4)$
$= a_1 + a_2 b_4 + a_3 b_3 + a_4 b_2 + b_1 = \Sigma_4$

Figure 4

Example 3

There is a class of sorting networks called odd-even merge networks invented by Batcher (2). The general form of a Batcher network appears in (Fig. 5.)

The inputs are divided into two equal groups A and B and each group is sorted (by recursive application of Batcher network). Then the odd-indexed terms from both groups are merged, and the even-indexed groups are merged. The smallest member of the odd merged group $a_1 b_1$ is the smallest element. The largest of the even group $a_n + b_n$ is the largest element and the intermediate elements are derived from a single row of comparators each of which compares the i^{th} output of the odd merge with the $i+1^{st}$ output of the even merge. We shall use the algebra to verify that this network indeed sorts.

Consider the even merge:

It has inputs $a_2 a_4 \cdots a_n$

$b_2 b_4 \cdots b_n$

output $c_1 c_2 \cdots\cdots\cdots\cdots c_n$,

the odd merge:

inputs $a_1 a_3 \cdots a_{n-1}$

$b_1 b_3 \cdots b_{n-1}$

output $d_1 d_2 \cdots\cdots\cdots\cdots d_n$

the outputs of the complete merge are

$e_1 e_2 e_3 \cdots e_{2n}$.

In Batcher's network $e_{2n} = c_n$

$$e_{2n-1} = c_{n-1} + d_n$$

$$\vdots$$

$$e_3 = c_1 + d_2$$

$$e_2 = c_1 . d_2$$

$$e_1 = d_1$$

The extreme terms are obviously correct

$e_{2n} = a_n + b_n = c_n$

$e_1 = a_1 . b_1 = d_1$

Now, $e_{2k} = c_k . d_{k+1}$ We will prove that this

$e_{2k+1} = c_k + d_{k+1}$ is correct.

let $k = n-j$, then $c_k = B^{j+1} + B^j A^1 + B^{j-1} A^2 + \ldots + B^1 A^{j+1}$

$$= b_{n-2j} + b_{n-2j+2} a_n + b_{n-2j+4} a_{n-2} + \ldots + b_n a_{n-2j+2} + a_{n-2j}$$

$[c_k = \Sigma_{j+1} (A_{even} \cup B_{even})]$

$k+1 = n-j+1 = n(j-1)$

$$d_{k+1} = b_{n-2j+1} + b_{n-2j+3} a_{n-1} + b_{n-2j+5} a_{n-3} + \ldots + b_{n-1} a_{n-2j+3} + a_{n-2j+1}$$

$[d_{k+1} = \Sigma_j (A_{odd} \cup B_{odd})]$

$e_{2k+1} = c_k + d_{k+1}$

$$= b_{n-2j} + b_{n-2j+1} + b_{n-2j+2} a_n + b_{n-2j+3} a_{n-1} + \ldots + b_{n-1} a_{n-2j+3} + b_n a_{n-2j+2} + a_{n-2j+1} + a_{n-2j}$$

$$e_{2k} = c_k \, d_{k+1}$$

$$= b_{n-2j} + b_{n-2j+1}a_n + b_{n-2j+2}a_{n-1} + \cdots$$
$$+ b_{n-1}a_{n-2j+2} + b_n a_{n-2j+1} + a_{n-2j}$$

Remember in general $R^k = r_{n-k+1}$

General term for e_{2k+1} (derived independently of network).

$$\text{odd } e_{2k+1} = e_{2n-2j+1} = e_{2n-(2j-1)}$$

$$= \text{sum of } 2j \text{ at a time products} = \sum_{i=0}^{2j} A^i B^{2j-i}$$

$$= B^{2j} + A^1 B^{2j-1} + \ldots + A^{2j}$$

$$= b_{n-2j+1} + b_{n-2j+1}a_n + b_{n-2j+3}a_{n-1} + \cdots$$
$$+ a_{n-2j+1}$$

$$\text{even } e_{2k} = e_{2n-2j} = 2j+1 \text{ at a time products}$$

$$= \sum_{i=0}^{2j+1} A^i B^{2j-i+1}$$

$$= A^o B^{2j+1} + A^1 B^{2j} + A^2 B^{2j-1} + \ldots + A^{2j-1} B^2$$
$$+ A^{2j} B^1 + A^{2j+1} B^o$$

$$= b_{n-2j} + b_{n-2j+1}a_n + b_{n-2j+2}a_{n-1} + \cdots$$
$$+ b_{n-1}a_{n-2j+2} + b_n a_{n-2j+1} + a_{n-2j}$$

and this checks with the outputs of the Batcher network.

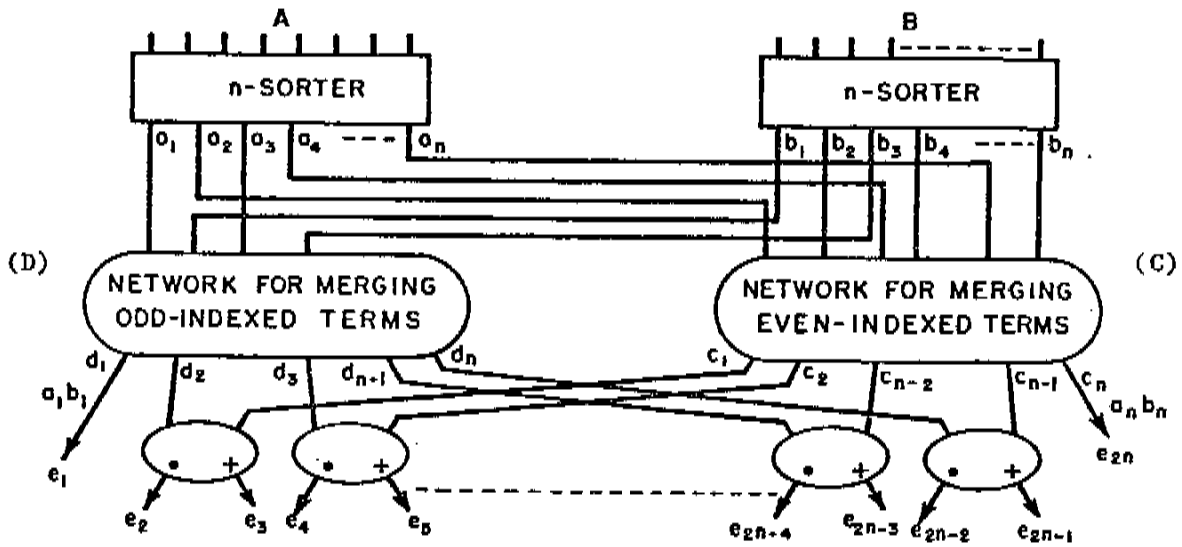


FIGURE 5 BATCHER NETWORK

Example 4: An alternate proof that the Batcher network sorts

As a consequence of theorem 3, we need only show that it sorts for all 0-1 inputs, that is for any input containing z zeroes and 2n-z ones the outputs $e_1,\ldots,e_z$ are zeroes and $e_{z+1}\ldots e_{2n}$ are ones.

(Refer to Figure 5)

Lemma: Let the number of zeroes out of Box C be Z_C and the number of zeroes out of Box D be Z_D. Then $Z_C \le Z_D \le Z_C + 2$.

Proof: Suppose there are k zeroes among the a's and ℓ zeroes among the b's.

Then $a_1,a_2,\ldots,a_k$ all equal 0

and $b_1,b_2,\ldots,b_\ell$ all equal 0.

If k is even then since alternate a's go to C and then D, the number of zeroes from a's to each is $\frac{k}{2}$. If k is odd then since D gets the first zero, one more zero goes from a to D then to C. In a like manner considering the parity of ℓ; if it is odd then D gets one more zero from b than does C and if even both D and C get the same number of zeroes.

Therefore, we have the following result:

1) If there is an odd number of zeroes among the a's and an even number among the b's then D has one more zero input than C.

2) Precisely the same result holds for an even number of zeroes among the a's and an odd number among the b's.

3) If there is an even number of zeroes among the a's and among the b's, then D and C have the same number of zero inputs.

4) If there is an odd number of zeroes among the a's and the b's then D has two more zero inputs than C.

We now return to the theorem from the lemma, there are three cases:

a) there are k zeroes in C and in D.

b) there are k+1 zeroes in D and k in C.

c) there are k+2 zeroes in D and k in C.

The resulting inputs to the last stage of the network are shown in the following table.

CASE	GATE INPUTS																	
	d_1	d_2	c_1	d_{k-1}	c_{k-2}	d_k	c_{k-1}	d_{k+1}	c_k	d_{k+2}	c_{k+1}	d_n	c_{n-1}	$1^{\dagger}$	c_n
a	0	0	0	0	0	0	0	1	0	1	1	1	1		1
b	0	0	0	0	0	0	0	0	0	1	1	1	1		1
c	0	0	0	0	0	0	0	0	0	0	1	1	1		1

Therefore the corresponding network outputs in each case would be of the form e_1 through $e_z=0$, $e_z=0$, e_{z+1} through $e_{2n}=1$ where z is the number of 0's input to the entire network.

The key point is that because of the above lemma no two successive output comparators can have <u>both</u> a 0 and 1 input.

The min-max relations we have developed thus far allow the analysis of particular sorting networks of fixed size. They are not of too much help in analysing sorting schemes. We have used them to verify the Batcher scheme, as well as particular networks, but in doing so certain more general principles provided the key to the proof. These general principles allow us to partition the sorting scheme in such a way that each part can be separately analyzed. For synthesizing also, the min-max relations are not enough. For example, the relations in Theorem 1 can be used directly to separately synthesize each output, of a sorting network but this would result in a very inefficient network. What is required is schemes for combining the use of min-max gates so that the same gate can effect more than one output.

We can describe methods for the analysis and synthesis of sorting schemes which go beyond the min-max equations and are more generally useful in analysing sorting schemes. We do this by giving a number of recursive definitions of functions whose arguments are sets and whose result is an ordered set.

If A is a sequence, $a_1\ a_2 \ldots a_n$

SORT (A) is a sequence, $s_1\ s_2 \ldots s_n$ consisting of the elements of A arranged so that $s_i \leq s_{i+1}\quad i < n$.

(1) SORT (A) = $\prod_{i=1}^{n} a_i$ * SORT (X) where * is concatenation

X is the sequence: $(a_1 + a_2) * (a_1 \cdot a_2 + a_3) * \cdots * (a_1 \cdot a_2 \cdot \ \ldots \ \cdot a_{n-1} + a_n)$

Proof: The elements of X are easily seen to be all those elements of A excepting $\prod_{i=1}^{n} a_i$.

If $A = a_1\ a_2 \ldots a_n$ and $B = b_1\ b_2 \ldots b_n$ are ordered sequences (i.e. $b_i \leq b_{i+1}$, $a_i \leq a_{i+1}$ $i < n$), Merge (A, B) is an ordered sequence $s_1 \ldots s_n$ consisting of all the elements from both A and B.

(2) Merge (A, B) = $(a_1 \cdot b_1) * \text{Merge } (X, Y) * (a_n + b_n)$

with X = the sequence $(a_2 \cdot b_2) * (a_3 \cdot b_3) * \cdots * (a_n \cdot b_n)$

Y = the sequence $(a_1 + b_1) * (a_2 + b_2) * \cdots * (a_{n-1} + b_{n-1})$

Proof: It is clear that $a_1 \cdot b_1$ is the lowest and $a_n + b_n$ is the highest of the elements in A and B. It is further easy to verify that X and Y as defined are ordered sequences as required by the definition of the Merge function, and they contain the remaining elements of A and B.

Consider any two successive elements in the sequence X, $a_i \cdot b_i$ and $a_{i+1} \cdot b_{i+1}$, without loss of generality, assume $a_i \leq b_i$ then $a_i \cdot b_i = a_i$ which $\leq a_{i+1}$ by hypothesis and is $\leq b_{i+1}$ because it's less than b_i; therefore $a_i \cdot b_i \leq a_{i+1} \cdot b_{i+1}$.

Similar reasoning works for Y. It is also clear that the set of elements in X is disjoint from the set of elements of X.

Notation: If $Z = z_1\ z_2 \ldots z_{2n}$ is an ordered sequence

$\underline{Z} = z_1\ z_2 \ldots z_n$ and $\overline{Z} = z_{n+1} z_{n+2} \ldots z_{2n}$

Let $A = a_1\ a_2 \ldots a_n$, $B = b_1\ b_2 \ldots b_n$

(3) Merge (A, B) = $\underline{\text{Merge}}\ (\underline{A}, \underline{B}) * \text{Merge } (X, Y) * \overline{\text{Merge}}\ (\overline{A}, \overline{B})$

$X = \overline{\text{Merge}}\ (\underline{A}, \underline{B})$

$Y = \underline{\text{Merge}}\ (\overline{A}, \overline{B})$

Proof: The key to the proof is the fact that the lowest $\frac{n}{2}$ elements from A and B are found in the subsets $\underline{A}$ or $\underline{B}$ and similarly that the highest $\frac{n}{2}$ elements from A and B are found in the subsets $\overline{A}$ or $\overline{B}$.

In a similar manner a more general result can be shown. If we define $\underline{A}$, $\underline{B}$ to be the first (lowest) m elements of A, B and $\overline{A}$, $\overline{B}$ to be the last (highest) n-m elements of A, B then the above equation (3) still holds for $m \leq n$.

Each of the above 3 equations corresponds to general schemes for analysis and/or synthesis of sorting networks. For example, the network of Figure 2a corresponds to equation (1), the network of Figure 2c corresponds to equation (2), and the network of Figure 3 corresponds to equation (3). In fact quite simple proofs of the validity of these networks could be obtained using the corresponding equations.

References

1. Birkhoff, G., Lattice Theory, American Mathematical Society Colloquium Publications, Providence, R. I., Vol. XXV, 1948, p.145.

2. Batcher, K., Sorting Networks and Their Applications, AFIPS Conference Proceedings, Vol. 32, Spring, 1968, pp. 307-314.

3. Levy, S. Y., Paull, M. C., An Algebra for the Analysis of Sorting Algorithms, Proceedings of the Third Princeton University Conference, 1968.