Notes on Merging Networks
(Preliminary Version)

by

Zhu Hong[†] and Robert Sedgewick[‡]

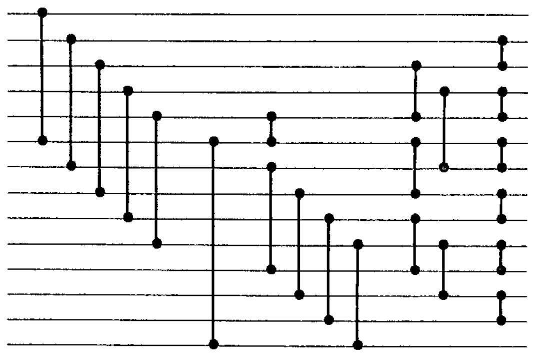
Department of Computer Science
Brown University
Providence, RI 02912
November, 1981

Abstract

Several new results which contribute to the understanding of parallel merging networks are presented. First, a simple new explanation of the operation of Batcher's merging networks is offered. This view leads to the derivation of a modified version of Batcher's odd-even (m, n) network which has delay time $\lceil \log(m+n) \rceil$. This is the same delay time as Batcher's bitonic (m, n) network, but it is achieved with substantially fewer comparators. Second, a correspondence is demonstrated between the number of comparators (and the delay time) for such networks and certain properties of binary number systems which have recently been extensively studied. Third, the $\lceil \log(m+n) \rceil$ delay time is shown to be optimal for a non-degenerate range of values of m and n.

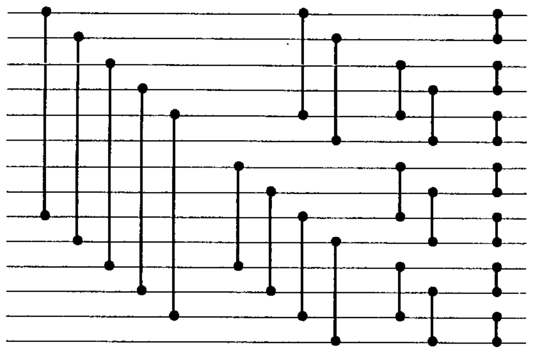
1. Introduction

An (m, n) *merging network* is a model for hardware devices for merging sorted files. Briefly, a merging network consists of a set of $m+n$ horizontal lines interconnected with *comparator modules*. Inputs to the network pass from left to right: when a comparator module is encountered, an exchange is performed if necessary to make the number on the bottom line connected by the comparator greater than the number on the top line. For example, below is drawn a $(5, 9)$ network. This network, when presented with one sorted file on the top 5 input lines and another sorted file on the bottom 9 input lines will produce a sorted file of size 14 as output after the numbers are passed from right to left through the comparator modules.



Such networks are good models for hardware implementation because they are *oblivious*: the sequence of comparisons is independent of the input values. They also are useful models for parallel computation (see [5]); the network structure makes obvious which comparisons are independent and so can be done in parallel. Many more details on merging networks, along with a summary of known results, may be found in [4].

The *delay time* of a merging network is the minimum number of parallel steps required for it to complete the merge. The delay time of the above network is 5, but a better delay time is possible for this example: the following network achieves a delay time of 4.

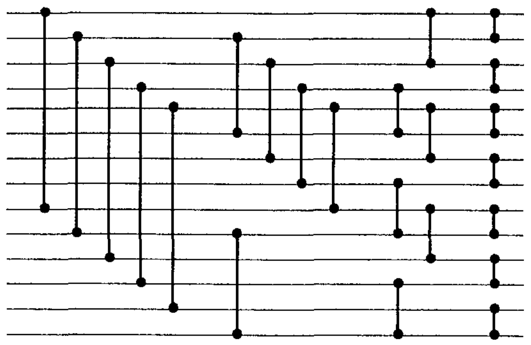


Both networks in the examples above are due to Batcher[1], who gave constructions for both types of networks for arbitrary m and n. The first, called the *odd-even merge*, uses fewer comparators: the number

[†]Currently visiting from Fudan University, China
[‡]Research supported by the NSF, Grant No. MCS80-17579

of comparators used by the odd-even merge is known to be optimal for $m = 2$[6], but for most values of m and n it is not known whether networks with even one less comparator exist, though it is strongly suspected that they do not. The second, called the *bitonic merge*, has a smaller delay time: the delay time is known to be optimal for $m = n$, but for most values of m and n, it is not known whether networks with a shorter delay time exist, though it is strongly suspected that they do not.

In Section 2 we exhibit a network, a variant of the odd-even merge, which achieves the same delay time as the bitonic merge, but using only about the same (asymptotically) number of comparators as the odd-even merge. Thus the best known bounds for delay time and number of comparators can be achieved simultaneously, at least asymptotically. The following is the (5, 9) version of this network, which has a delay time of 4, but uses two less comparators than the bitonic network.



The number of comparators in the networks for general m and n has proven to be a difficult quantity to analyze: in Section 3 we show how this quantity can be studied through a correspondence with certain properties of binary number systems. These properties have only recently been fully analyzed: this analysis yields results of direct applicability to merging networks.

Minimal delay time and minimal number of comparators for merging networks are thought to be extremely difficult questions to resolve. In Section 4 we give some evidence that the modified odd-even merge of Section 2 (and the bitonic merge) have optimal delay time by proving a lower bound that holds for m and n satisfying certain conditions.

Section 5 offers some concluding remarks.

2. Merging Tableaux

Batcher's odd-even merging method can be very simply described and proven correct in terms of operations on two-dimensional tableaux of numbers, as follows: Suppose that the following two files of eight numbers are to be merged:

```
0 0 0 4 6 9 9 9        0 1 2 2 3 3 8 8
```

If we write one file on top of the other, then compare-exchange the vertical pairs to put the smaller one on top, then we get a 2×8 tableau with the property that both rows and columns are sorted:

```
0 0 0 4 6 9 9 9        0 0 0 2 3 3 8 8
| | | | | | | |
0 1 2 2 3 3 8 8        0 1 2 4 6 9 9 9
```

Now, this tableau can be compressed into a 4×4 tableau with the same property by exactly the same operation: divide each row in half, interleave the rows, then perform compare-exchanges on vertical pairs:

```
0 0 0 2        0 0 0 2
3 3 8 8        0 1 2 4
| | | |
0 1 2 4        3 3 8 8
6 9 9 9        6 9 9 9
```

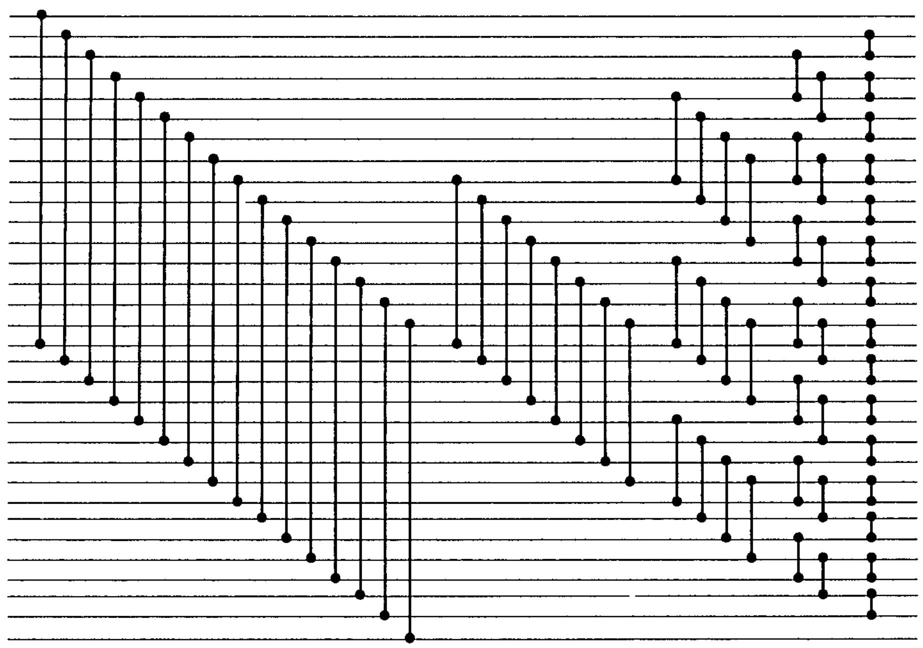
Again, both rows and columns are sorted. Clearly, compare-exchanges are needed only between elements in the middle two rows: all elements in the first row are smaller than all elements in the second row because they came from the same row in the previous tableau, and they are smaller than corresponding elements in the third row because they were involved in compare-exchanges on the previous step.

Continuing, we divide each row in half, interleave the rows, and perform compare-exchanges on vertical pairs to get a 8×2 tableau with both rows and columns sorted:

```
0 0        0 0
0 2        0 1
| |
0 1        0 2
2 4        2 3
| |
3 3        3 4
8 8        6 8
| |
6 9        8 9
9 9        9 9
```

It is possible to prove that compare-exchanges are only needed between (even, odd) row pairs, though this proof is not as trivial as it first seems.

One more step completes the sort, creating a 16×1 sorted tableau:



Odd-Even (16,16) Merging Network

```
0     0
0     0
|
0     0
1     0
|
0     1
2     2
|
2     2
3     3
|
3     3
4     4
|
6     6
8     8
|
8     8
9     9
|
9     9
9     9
```

Again, only compare-exchanges between (even, odd) pairs are needed.

The generalization of the description above to larger files is obvious: to merge together two files of size 2^n organized in a 2×2^n tableau, compare-exchange vertically adjacent elements, then perform the following steps n times: divide each row in half, interleave the row halves, and perform compare-exchanges between vertically adjacent elements in (even, odd) row pairs. To prove that this is valid, it is necessary to prove that compare-exchanges between two sorted row pairs leaves both rows sorted (this is easily proven by contradiction), and that the specified compare-exchanges are sufficient to sort the columns (this is proven by showing that, after interleaving, elements in odd numbered rows could not have any smaller elements in the column below them, while elements in even numbered rows could have one smaller element immediately below).

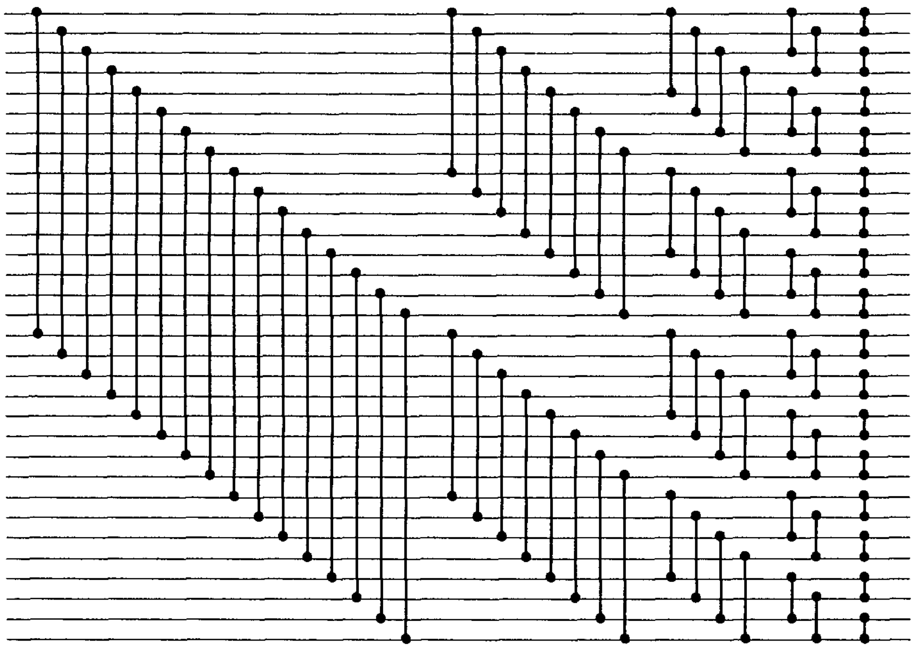
The comparisons performed in this method are oblivious, so it can be translated directly to a merging network. The network which results is Batcher's odd-even merging network. For example, the network for a 16×16 merge is given above. (The 8×8 network corresponding directly to the above example may be found by considering only those comparators with both ends on the middle 16 lines.)

All of the compare-exchanges during each divide-interleave-compare-exchange step can be done in parallel, so the delay time of the network is simply the number of such steps.

The same idea can be used to describe Batcher's bitonic method, as follows: To merge to sorted files of size 2^n, organize them in a 2×2^n tableau as before, but put the second line in reverse order. Then compare-exchange the two rows and perform the divide-interleave-compare-exchange cycle as before, except using vertically adjacent elements in *(odd, even)* row pairs. The following table shows the first three steps of this method on the same sample file as above:

```
0 0 0 4 6 9 9 9        8 8 3 3 2 2 1 0


0 0 0 4 6 9 9 9        0 0 0 3 2 2 1 0
| | | | | | | |
8 8 3 3 2 2 1 0        8 8 3 4 6 9 9 9
```


Bitonic (16,16) Merging Network

```
0 0 0 3      0 0 0 0
| | | |
2 2 1 0      2 2 1 3
8 8 3 4      6 8 3 4
| | | |
6 9 9 9      8 9 9 9
    0 0      0 0
    | |
    0 0      0 0
    2 2      1 2
    | |
    1 3      2 3
    6 8      3 4
    | |
    3 4      6 8
    8 9      8 9
    | |
    9 9      9 9
```

Continuing, the sort will be completed in one more step. This method obviously requires more comparisons than the odd-even merge, but it has other benefits (descibed below). The proof that this method is correct is somewhat more complicated than for the odd-even merge, because the rows are not completely sorted. (Instead, they form *bitonic sequences*, which go up, then down, then possibly up again.) The bitonic merging network which derives directly from this method is complementary to the odd-even network in an obvious way, as shown in the above diagram.

Tableaux provide a simple derivation and description of Batcher's methods: the benefits of this view are apparent when we consider the problem of deriving a method for (m, n) merging when m and n are not equal to the same power of two. The idea is to use the method above for a power of two bigger than m and n, supplementing the numbers to be sorted with conveniently placed "dummy" keys. In the standard odd-even merge, the dummy keys have values higher than all other keys and are placed at the ends of the arrays. For example, to do a 5 by 9 merge, we would begin with the tableau

$$\begin{matrix} 2\;4\;4\;8\;9\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty \\ 1\;3\;3\;4\;5\;5\;6\;6\;7\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty \end{matrix}$$

Comparisons between two dummy ∞ keys are ignored, since the outcome is predetermined and cannot affect the result. The delay time for this case is obviously $\lceil \log_2(\max(m, n)) \rceil + 1$.

This method can be improved as follows: put dummy keys with value smaller than every other key (o) at the beginning of the first array, dummy keys with value larger than every other key (∞) at the end of the second array. Thus, for the example above, we would begin with the tableau

$$\begin{matrix} o\;o\;o\;o\;o\;o\;o\;o\;o\;o\;o\;2\;4\;4\;8\;9 \\ 1\;3\;3\;4\;5\;5\;6\;6\;7\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty\;\infty \end{matrix}$$

Now, the result of all the compare-exchanges is predetermined for this case, so the first stage can be skipped entirely. The delay time for this modified version of the odd-even merge is $\lceil \log_2(m + n) \rceil$. We'll refer to this method as the *even-odd* merge. This same delay time can be achieved in the bitonic merge, for example by beginning with the tableau

$$\begin{matrix} o\;o\;o\;o\;o\;o\;o\;o\;o\;o\;o\;2\;4\;4\;8\;9 \\ 7\;6\;6\;5\;5\;4\;3\;3\;1\;o\;o\;o\;o\;o\;o\;o \end{matrix}$$

Actually, since the network can sort bitonic sequences, a different method is normally used, as demonstrated by the following initial tableau.

```
o o o o o o o o o o o o o o o o
o o 2 4 4 8 9 7 6 6 5 5 4 3 3 1
```

To derive (m,n) merging networks from these considerations, we start with the network for a power of two bigger than both m and n and then delete superfluous lines and redundant comparators. The situation is more complicated than it might seem because lines may need to be relabeled to account for the effect of exchanges involving dummy keys. (The reader may appreciate this point by deriving the $(5,9)$ network given in Section 1 from the comparators touching the 1st through the 5th and the 26th through the 32nd lines in the large odd-even network above.) Though this relabeling involves no real cost, it makes particular networks difficult to comprehend and analyze.

For the even-odd and the bitonic merges, such relabeling is avoided because of the placement of the dummy keys. An (m,n) even-odd network is derived by choosing m lines above the center and n lines below the center from a larger network, keeping all comparators with both endpoints on such lines; an (m,n) bitonic network is derived from the $(m+n)$ bottom lines in a larger network in the same way. (The reader may wish to check the use of these constructions for the $(5,9)$ networks in Section 1 from the 32-line networks in this section.) The placement of the dummy keys in these networks is such that no "virtual exchanges" are required: (0) keys never move down and (∞) keys never move up. If $\lceil \log_2(m+n)\rceil < \lceil \log_2(\max(m,n))\rceil + 1$ then no comparators from the first stage survive: this is the savings in the delay time.

3. Networks and Number Systems

The delay time for the odd-even and the even-odd networks is trivial to compute from the tableau representation; the bitonic merge has a recursive structure which also yields the delay time immediately. Counting the number of comparators for a general (m,n) network, however, is significantly more difficult: no closed formula has previously been available for any of the methods. In this section, we see the reason for this by exhibiting correspondences between the numbers of comparators in these networks and simple quantities related to binary numbers. Closed formulae have recently been derived for these quantities, but only after extensive analysis (see [2,3]).

The number of comparators for each of the three types of networks that we have discussed can be described by relatively simple recurrence relations, shown in the table below.

Each formula gives the number of comparators used in a (m,n) network. All of the quantities are 1 for $m = n = 1$ and 0 for $m+n < 2$; the number of comparators for the odd-even and even-odd merges is also 0 if m or n is 0. Derivations of the recurrences for the bitonic and odd-even methods may be found in [4]; the argument for the even-odd method is similar.

Knuth notes that $C_o(m,n)$ is "not an especially simple" function of m and n, and this comment certainly applies to the other functions. By working with $C_o(m+1,n+1) - C_o(m,n)$, Knuth is able to show that

$$C_o(m,n) = \frac{1}{2}n(\lceil \log_2 m\rceil + \delta(m)) + O(1)$$

(where $\delta(m) = m/2^{\lceil \log_2 m\rceil}$) as n grows, for m fixed. By working with $C_e(m+1,n-1) - C_e(m,n)$, we are able to show that the same expression holds for $C_e(m,n)$, so that

$$C_e(m,n) = C_o(m,n) + O(1)$$

as n grows, for fixed m.

Knuth does not give a solution for the recurrence for $C_b(m+n)$; but it turns out that this is the easiest of the functions to deal with. Consider the function

$$\nu(i) = \{\#\text{ of 1's in the binary representation of } i\}.$$

We are primarily interested in the function

$$\beta_1(n) = \sum_{0\le i<n} \nu(i).$$

There are $\lfloor n/2\rfloor$ odd numbers less than n and $\lceil n/2\rceil$ even numbers less than n, so we are immediately led to the recurrence

$$\beta_1(n) = \beta_1(\lceil (n/2)\rceil) + \beta_1(\lfloor (n/2)\rfloor) + \lfloor (n/2)\rfloor.$$

This is precisely the same as the recurrence for $C_b(m+n)$, including initial conditions, so this constitutes a proof that

$$C_b(m+n) = \beta_1(m+n).$$

Bitonic	$C_b(m+n) = C_b(\lceil \frac{1}{2}(m+n)\rceil) + C_b(\lfloor \frac{1}{2}(m+n)\rfloor) + \lfloor \frac{1}{2}(m+n)\rfloor$
Odd-even	$C_o(m,n) = C_o(\lceil m/2\rceil, \lceil n/2\rceil) + C_o(\lfloor m/2\rfloor, \lfloor n/2\rfloor) + \lceil \frac{1}{2}(m+n)\rceil) - 1$
Even-odd	$C_e(m,n) = C_e(\lceil m/2\rceil, \lfloor n/2\rfloor) + C_e(\lfloor m/2\rfloor, \lceil n/2\rceil) + \lceil m/2\rceil + \lceil n/2\rceil - 1$

Recurrences to count comparators

This function has been recently analyzed in detail (see [2,3]) with the result that

$$\beta_1(n) = \frac{1}{2}n(\log_2(n) + B(\log_2 n)) + O(1)$$

where $B(n)$ is a periodic function with period 1.

Similar, though more complicated results are available for the other recurrences. First, consider a complementary function to β_1:

$$\beta_0(n) = \sum_{0\le i<n} \nu(2^{\lceil \log_2 n\rceil} - 1 - i).$$

This is the same as the number of 0 bits in the rightmost $\lceil \log_2 n\rceil$ bits of the binary representation of the numbers less than n, so that

$$\beta_0(n) + \beta_1(n) = n\lceil \log_2 n\rceil.$$

As we know from the analysis of the bitonic networks, there is a strong relationship between the network recurrences and these β-functions. For example, it turns out that $\beta_0(m+n) - 1$ satisfies the same recurrence as does the number of comparators for the odd-even merge, and $\beta_0(n) + \beta_0(m)$ satifies the same recurrence as does the number of comparators for the even-odd merge. However, unlike the bitonic case, the initial conditions do not match. Close examination of this discrepency uncovers some interesting facts about the networks, described below.

The problem is that the recurrences for the odd-even and even-odd methods assume (naturally) that no comparators are required for a $(0,n)$ merge, while this initial condition is not imposed on the recurrence for the bitonic network. The reason for this is that an n-line bitonic network can be used not only for a $(0,n)$ merge but also for a $(1,n-1)$ merge, a $(2,n-2)$ merge, etc. Thus, obviously, more than 0 comparators are needed, which is reflected in the analysis only by the absence of an initial condition for $m = 0$ or $n = 0$. Now, such a condition can be imposed on bitonic networks, resulting in merging networks with fewer comparators and the same delay time. (An alert reader may have noticed a "missing" comparator in the $(5,9)$ bitonic network in Section 1, which was removed according to this principle.) The recurrence describing the number of comparators is the recurrence for the even-odd case, with the additive term modified to be the additive term for the bitonic case, which is one greater if both m and n are odd. This recurrence seems to be as difficult to solve as the others.

Conversely, this suggests that if we remove the initial conditions for $m = 0$ and $n = 0$, then the recurrences for the odd-even and the even-odd cases my be easier to solve. This in fact turns out to be the case: the number of comparators for the odd-even merge under these assumptions is $\beta_0(m+n) - 1$; the number of comparators for the even-odd merge is $\beta_0(m) + \beta_0(n)$. These solutions are proven directly from the recurrences and simple properties of binary numbers, as was done above for the bitonic case. The networks described by these recurrences have extra comparators: it is not clear whether they give the networks any particular special properties. The expressions involving β_0 are quite accurate estimates of the number of comparators in the networks for m close to n.

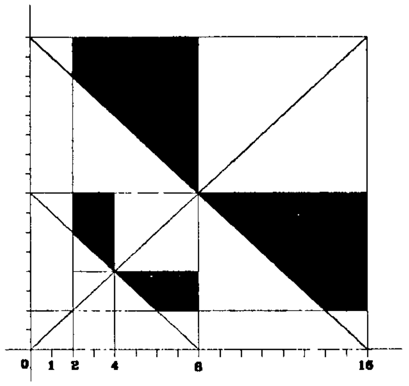
4. Lower Bounds

The delay time for any (m,n) merging network is known to be less than or equal to $\lceil \log_2(m+n)\rceil$ for $m = 1$ or $m = n$ (see [4]); thus both the bitonic and the even-odd method are optimal for these values. In this section we extend the range of values for which these networks are known to be optimal.

In particular, we are able to prove that, for $m < n$, any (m,n) merging network must have a delay time of at least $\lceil \log_2(m+n)\rceil$ except possibly for

$$2 < m < 2^{\lfloor \log_2 n\rfloor} < n < 2^{\lceil \log_2 n\rceil} < m+n.$$

The obvious symmetric result holds for $n < m$. The shaded areas in the following diagram show the values of $m, n < 16$ for which the question is unresolved.



The proofs consist of case analyses covering the various gaps in the inequality list above.

If $m = 1$, then $\lceil \log_2(n+1)$ levels of delay are required because the first input could end up on any one of the $n+1$ output lines, and each level of delay can at most double the number of lines that the first input could be switched to. (See [4].)

If $m + n \le 2^{\lceil \log_2 n\rceil}$, then we must have

$$\lceil \log_2(m+n)\rceil \le \lceil \log_2 n\rceil \le \lceil \log_2(n+1)\rceil$$

which we know is no bigger than the delay time required to do a $(1,n)$ merge, which certainly is no bigger than the delay time required to do an (m,n) merge.

If n is a power of 2, then we must have

$$\lceil \log_2(m+n) \rceil \leq \lceil \log_2 n \rceil + 1 \leq \lceil \log_2(n+1) \rceil$$

which is no bigger than the delay time required to do an (m,n) merge, as above.

If $m \geq 2^{\lfloor \log_2 n \rfloor}$, then we must have

$$\lceil \log_2(m+n) \rceil \leq \lceil \log_2(n+1) \rceil + 1 \leq \lceil \log_2(2m+1) \rceil.$$

The last inequality holds because

$$\begin{aligned} \lceil \log_2(n+1) \rceil + 1 &= \lceil \log_2(2^{\lceil \log_2(n+1) \rceil} + 1) \rceil \\ &\leq \lceil \log_2(2^{\lfloor \log_2 n \rfloor + 1} + 1) \rceil. \end{aligned}$$

Now, $\lceil \log_2(2m+1) \rceil$ is no greater than then delay time required to do an (m,n) merge by the same kind of argument used for the $m = 1$ case above: any one of the first m or the last $m+1$ inputs could end up on the last output line, and each level of delay can at most halve the number of input lines switchable to any particular output line.

Although these proofs show that the bitonic and even-odd networks have optimal delay time for a large range of values of m and n, the general question of proving optimality for all m and n still appears to be quite difficult. It is certainly reasonable to conjecture that they are optimal, but the remaining cases seem to require much more sophisticated arguments.

5. Conclusion

Networks for merging and sorting have been the subject of intensive study since the "Bose-Nelson" problem first came to light before 1960, culminating in the development of Batcher's odd-even and bitonic sorting and merging networks. As appropriate models for VLSI implementations of sorting and merging machines, there has been renewed interest in the properties of Batcher's and related networks. The simplified explanation of Batcher's odd-even network which is given in Section 2 may make easier the development of variants appropriate for new technologies, as is evidenced by the development in Section 3 of a natural way to improve the delay time of the odd-even merge for (m,n) networks. The correspondence with binary number systems which is described in Section 5 provides a new approach to analyzing the networks, which has been useful in understanding known networks and in studying new ones.

References

1. K. E. Batcher, "Sorting networks and their applications", *AFIPS SJCC* **32** 1968.
2. H. Delange, "Sur la fonction sommatoire de la fonction somme des chiffres", *Enseignement Math.* **21**, (1975).
3. P. Flajolet and L. Ramshaw, "A note on Gray code and odd-even merge", *SIAM J. on Computing* **9**, 1 (1980).
4. D. E. Knuth, *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, Mass. (1973).
5. H. S. Stone, "Parallel processing with the perfect shuffle", *IEEE Trans. on Computing* **C-20**, 2 (1971).
6. A. C. Yao and F. F. Yao, "Lower bounds for merging networks", *J. of the ACM* **23**, 3 (1975).