Significant Improvements to the Hwang-Lin Merging Algorithm

GLENN K. MANACHER

University of Illinois, Chicago, Illinois

ABSTRACT The Hwang–Lin merging algorithm is the best general-purpose merging algorithm that has been found Many improvements to it have been devised, but these are either for special values of m and n, the number of items being merged, or else improvements by a term less than linear in $n + m$ when the ratio n/m is fixed

A new methodology is developed in which, for fixed ratio n/m, it is possible to decrease the number of comparisons by a factor proportional to m, in fact $m/12$, provided $n/m \geq 8$ and $m \geq 24$ It is shown that the coefficient $\frac{1}{12}$ is not best possible, and a technique for improving it slightly to $\frac{31}{336}$ is sketched

KEY WORDS AND PHRASES merging, Hwang–Lin algorithm

CR CATEGORIES 5 25, 5 31

1. Introduction

In [4], Hwang and Lin present a merging algorithm that combines the virtues of binary insertion and linear merging. It is the best simple general-purpose merging algorithm known for arbitrary list size.

Let m and n be the number of items being merged, with $m \leq n$. This lists "small" and "large," containing, respectively, m and n items, are supposed sorted, so that the least item has the lowest index. As an item is located in the evolving merged list, it is deleted from the list originally containing it. Indexing, as we define it, is a little unusual; we will say that the first *remaining* element in a list has index 1, the next index 2, and so forth.

Linear merging consists in running down both lists, looking for the least element by comparing element 1 of both lists. The process continues until one list is exhausted, whereupon the remaining elements of the other list are merged onto the bottom of the merged list. In the worst case, $m + n - 1$ comparisons are needed.

Binary insertion is the best method for merging one element into n. In the simplest case, $n = 2^k - 1$. The singleton element is first compared with element $2^{k-1} - 1$; if the singleton element is smaller, it belongs in the upper half, so the next comparison is with element $2^{k-2} - 1$, etc. In the worst case k comparisons are required, and this is best possible when merging one element into n, $2^{k-1} \leq n \leq 2^k - 1$.

The Hwang–Lin algorithm (HLA) [4] breaks the big list into *blocks* of size $2^\tau \equiv 2^{\lfloor \log(n/m) \rfloor}$. The first element of the small list, denoted small(1) is compared with large(2^τ). If the first element of the small list is larger than large(2^τ), then 2^τ elements from the large list are *annexed*, i.e., removed from the large list. If the first element of the small list is less than or equal to large(2^τ), small(1) is merged into {large(1), large(2), . . . , large($2^\tau - 1$)} by binary insertion in τ comparisons. (We are using a model in which only comparisons are

Author's address, Department of Information Engineering, University of Illinois at Chicago Circle, Computer Center, Box 4348, Chicago, IL 60680

computationally relevant.) Then element 1 is removed from small, and the elements in large that are smaller than small(1) are annexed. The same process is iterated until one list is exhausted In the companion paper [7] we state the algorithm formally and indicate that we shall use a "static" variant in which τ is computed only once. We also provide the explanation for a diagrammatic convention for the algorithm, in which the iterated core is illustrated as a simple, stylized flowchart. The diagram for the static Hwang–Lin algorithm is shown in Figure 1.

In Figure 1, no annexation is shown in connection with an insertion because in the worst case, no elements will be annexed.

In [7] we analyze the static variant of the algorithm and discover that the number of comparisons in the worst case is identical to the number required [6] by the HLA. The idea is that in the worst case there will be m insertions requiring $\tau + 1$ comparisons each, together with $\lfloor n/2^\tau \rfloor - 1$ annexations requiring 1 comparison each. Therefore, if $\mathrm{HL}(m, n)$ is the number of comparisons required by the HLA or its static variant, then

$$\mathrm{HL}(m, n) = m(\tau + 1) + \lfloor n/2^\tau \rfloor - 1. \tag{1}$$

Definition. Let the HLA operate on lists of size m (the small list) and n (the large list). The operation of the algorithm will be called a *complete run* if at the end, no element of the small list is uninserted and, at most, one block of the large list is unannexed.

Clearly, for every m and n, there exist orderings of the small and large lists that will produce a complete run of the algorithm. This fact animates a simple characterization of worst-case runs, namely:

PROPOSITION 1. *Among the complete runs there is always at least one worst-case run.*

To prove Proposition 1, we require a few simple observations First, we note that insertions of one element of small into $2^\tau - 1$ elements of large may in themselves produce the annexation of between 0 and $2^\tau - 1$ elements of large, depending on where the insertion is made into the $2^\tau - 1$ elements. Clearly, annexing more than 0 elements cannot serve to *increase* the total number of comparisons; it must either reduce it or leave it unaltered. Therefore, if we consider only worst-case runs, we may safely assume that an insertion produces no annexation.

Second, we note that a merge, in order to run to completion, may leave one or more elements "uninserted" from small or unannexed from large, but not both.

We are now ready to prove the proposition. Consider a purportedly worst-case run in which, at the end, there are $k > 1$ elements remaining in small, and none in large. Necessarily, the last step (Figure 1) must have been the annexation of the last block of large (or fragment, if the number of elements in large is not a multiple of 2^τ) by the first (remaining) element of small. There clearly exists an alternative run, with different data, which is identical to the first run up to the insertion of the first $m - k$ elements of small but which, at greater cost, then inserts $k - 1$ elements and finally annexes the last block or fragment of large with the last element of small. Hence the former run is not worst case. An essentially identical argument shows that runs in which no element of small, but more than one block of large, remains, also cannot be worst case. Last, consider the case in which just one element of small is uninserted, and no elements of large are unannexed. The last step was necessarily the annexation of the last block or fragment by the last element of small. If this last block or fragment contains more than one element, then insertion of the last element of small will cost more than annexation. If it contains exactly one element, then insertion and annexation will cost the same, namely, one comparison. This complete the proof. □

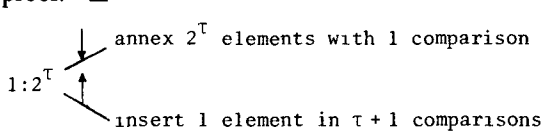

FIG 1

Our main tool for making the analysis of our merging schemes tractable is a variant of cost accounting appropriate only to complete runs. Its relevance to worst-case analysis is guaranteed by Proposition 1. The idea is that since both the number of annexations and insertions in a complete run are known, the total cost of annexations may be averaged over each element of small and "charged" at the time the element is inserted. This is counterbalanced by setting the "cost" of an annexation to 0. Mutatis mutandis, the original charging scheme is transformed into one vastly easier to analyze. We shall call costs reckoned in this way *effective costs.*

Specializing now to the case $n = 2^d m$, the number of annexations in a complete run equals the number of insertions minus 1. Hence assigning an effective cost of 1 to each insertion to cover the aggregate cost of insertion, we shall overcalculate the total number of comparisons by just 1. Our objective is an analysis correct to order m, so this is acceptable.

This effective cost for each step of Figure 1 is then

Step	Cost
"Annex 2^d "	Zero
"Insert one element "	$d + 2$

This method, trivial for Figure 1, will permit easy analysis of the more complex schemes in this paper and its companion [7]. In the sequel and in [7], the term "cost" applied to diagrams like that of Figure 1 will be understood to mean effective cost.

2. *Significant Improvements*

We now introduce the notion of a *significant improvement* over the HLA. Let $M(m, n)$ be the number of comparisons required to merge lists of length m and n. We note that to date, improvements in the HLA, in the sense that they yield smaller results than (1), have been presented either for special values of m or n (or both), or are similar to Hwang and Lin's demonstration [5] that $M(m, 2m) \leq 3m - 2$, or else [2] have been achieved for a broad spectrum of values of n and m without achieving *significant improvement* in the sense of this paper. By a significant improvement, we mean an improvement that for fixed n/m increases linearly with m. Such schemes have not been demonstrated, it is the purpose of this paper to show that they exist.

For constant n/m, if an algorithm can be found that requires $\delta(n/m)m$ comparisons (to order m), then easy information-theoretic arguments show that [6]

$$\mathrm{HL}(m, n) - \delta(n/m)m < m, \tag{2}$$

so that if $\mathrm{HL}(m, n) = h(n/m)m - c$, for some constant c, then $\delta(n/m)$ can differ from $h(n/m)$ only by some number less than one. Thus the margins for improvement are not very great.

A significant improvement for $n/m = 8$ was presented in [7]. Its diagram is shown in Figure 2. This diagram depends on the fact that $M(2, 8) = 6$ and $M(3, 8) = 8$. $M(2, 8)$ comes from the explicit formula

$$M(2, n) = \lceil \log(\tfrac{14}{17})(n + 1) \rceil + \lceil \log(\tfrac{7}{12})(n + 1) \rceil \tag{3}$$

derived first by Graham [1] and then by Hwang and Lin [3]. $M(3, 8)$ is easily derivable from $M(3, 6) = 7$, which was derived by Hwang and Lin (see [6] for their proof). The diagram indicates that the first *three* elements of the small list are compared, respectively, to the eighth, ninth, and ninth elements of the large list. The costs are now calculated by assuming that an inserted element should annex eight elements in order to be at "par."[1]

[1] "Par" means that no annexation cost is factored in. If the second step read "insert 1 in 5 and annex 8," its cost would be 5. "Insert 1 in 5" would cost 6, the extra comparison compensating for the unperformed annexation that must be averaged in.

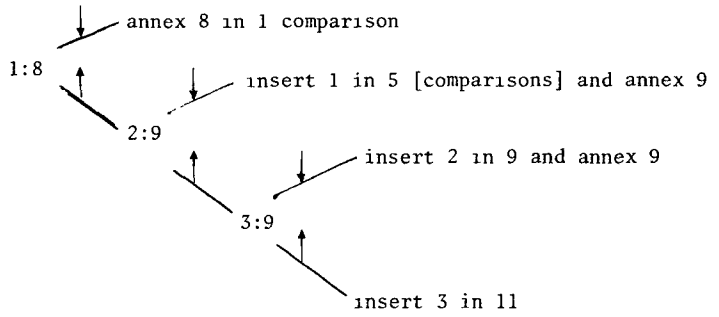

FIG 2

The step "insert 1 in 5 comparisons and annex 9" now has cost of only $4\frac{7}{8}$ because it annexes *not only* its "par" but *also* one-eighth of the next group of eight. The costs (per element) of Figure 2 are thus

Step	Cost
"Annex 8 "	Zero
"Insert 1 "	$4\frac{7}{8}$
"Insert 2 "	$4\frac{15}{16}$
"Insert 3 "	$4\frac{2}{3}$

The highest cost is that of "insert 2 ... ." It is clear that for $n/m = 8$ and m sufficiently large, we will obtain for $H(m, n)$ $4\frac{15}{16}$ comparisons per element rather than the 5 required by the HLA where $H(m, n)$ is the number of comparisons required by the new algorithm.[2]

We would now like to generalize this construction in four ways:

(1) To find analogs to Figure 2 for $n/m = 2^d$ for $d > 3$ (none are known to exist for $d < 3$)

(2) To generalize further to n/m not a power of 2.

(3) To find just how big m must be to make $H(m, n)$ smaller than $\mathrm{HL}(m, n)$.

(4) To discover whether for sufficiently large n/m there exists a single algorithm for which $\delta(n/m)$ is larger than some fixed constant, and if so to find the constant.

The main result is contained in our

THEOREM. *For $n/m \geq 8$, there exists a merging algorithm, which we present explicitly, requiring $H(m, n)$ comparisons, where*

$$HL(m, n) - H(m, n) = m/12 - e \tag{4}$$

where $-\frac{13}{12} < e < 2$.

We now develop the proof of the theorem.

LEMMA 1. $M(2, 2^d) = 2d$ *for* $d \geq 3$.

PROOF. Equation (3). □

LEMMA 2 $M(3, 2^d) \leq 3d - 1$ *for* $d \geq 3$.

Note. $\mathrm{HL}(3, 2^d) = 3d$ for $d \geq 2$.

PROOF. Figure 3, together with the fact that $M(3, 8) = 8$. □

LEMMA 3. *For* $d \geq 3$,

(*a*) $M(3, \lfloor(\frac{31}{28})2^d\rfloor - 1) \leq 3d - 1$,

[2] This is achieved by combining "insert 2 " and annexation in just the right proportion and sequence to produce a complete run Clearly this is possible and will produce a run with the largest possible coefficient of m We should also mention that the problem of maximizing the number of comparisons for Figure 2 has an alternative formulation in terms of integer programming to find values of integer variables x_1, x_2, x_3 and x_4 that will maximize the value of $x_1 + 5x_2 + 9x_3 + 11x_4$ subject to the constraints $m = x_2 + 2x_3 + 3x_4$ and $8m = 8x_1 + 9x_2 + 9x_3$

(*b*) $M(3, \lfloor(\frac{5}{4})2^d\rfloor - 1) \leq 3d$,

(*c*) $M(3, \lfloor(\frac{3}{2})2^d\rfloor - 1) \leq 3d + 1$,

(*d*) $M(3, \lfloor q2^d\rfloor - 1) \leq 3d + 2$ *for* $\frac{3}{2} \leq q \leq 2$.

PROOF. (d) is almost trivial, if $q = 2$ then by Lemma 2, $M(3, 2^{d+1}) \leq 3(d + 1) - 1 = 3d + 2$. The inequality obviously holds for lesser q. To prove (c) from (d), we use Figure 4. We then prove (b) from (c) using virtually the same scheme as that of Figure 5. To prove (a) from (b), we use Figure 5. □

LEMMA 4. $M(4, \lfloor(\frac{5}{4})2^d\rfloor - 1) \leq 4d - 1$.

PROOF. The HLA. □

It turns out that for the case $n/m = 8$, a highly efficient algorithm—considerably more efficient than that of Figure 2—involves four elements from the small list rather than three. (The reader should consult [7] for details.) We now generalize this four-element diagram, using also the fact that $M(2, \lfloor(\frac{17}{14})2^d\rfloor - 1) = 2d$, as can be derived directly from (3). The highest cost is for the "insert 3 ..." step; the cost is $d + \frac{23}{12}$. Following similar analysis in [7], it can be shown that when $m \bmod 3 = 0$, then the worst case is realized when *all* the elements in the small list are inserted by means of this step. Let $H_B(m, n)$ be the number of comparisons required. Then

$$H_B(m, n) = (d + \tfrac{4}{3})m + \lfloor(n - \lfloor(\tfrac{5}{4})2^d\rfloor(m/3))/2^d\rfloor. \tag{5}$$

By using $\lfloor a\rfloor = a - \theta$, where $0 \leq \theta < 1$, we obtain

$$H_B(m, n) = (d + \tfrac{11}{12})m + n/2^d - \theta_1 \tag{5a}$$

which reduces to $(d + \frac{23}{12})m - \theta_1$ when $n = 2^d m$.

We now consider n/m not a power of 2. Suppose we consider n in the range $n \in \sigma_d = \{2^d m, 2^d m + 1, \ldots, 2^{d+1}m - 1\}$. Clearly, $\tau = \lfloor\log n/m\rfloor$ will be just d for all n in this range.

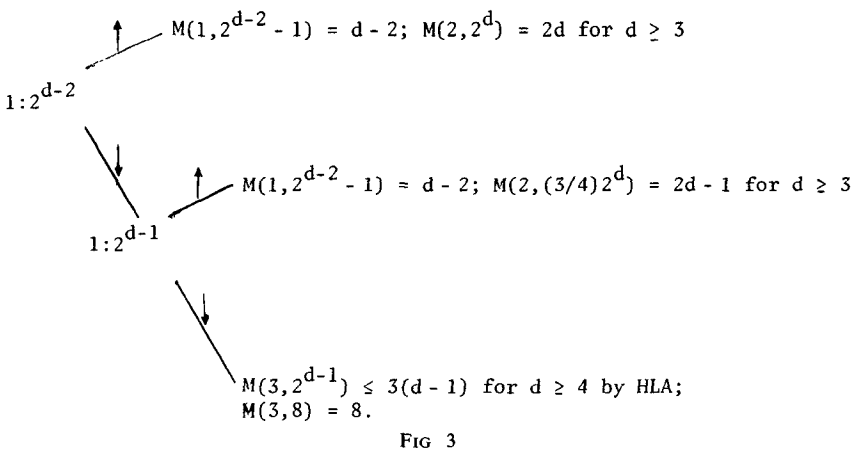

FIG 3

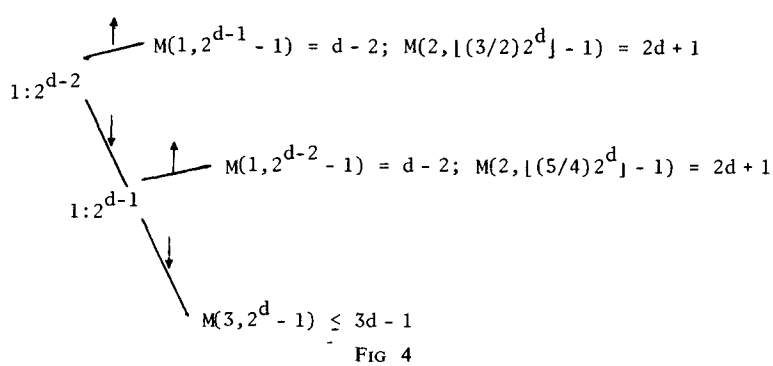

FIG 4

Now the HLA can be reexpressed as

$$\mathrm{HL}(m, n) = m(d + 1) + n/2^d - 1 - \theta_2 \tag{6}$$

in the same range. We need to have a formula for $H_B(m, n)$ when m is not a multiple of 3. If m is congruent to 1 mod 3, then the worst case for Algorithm B is realized when all but one of the elements are inserted by means of "insert 3 ..." and one is inserted by means of "insert 1 ... ." If m is congruent to 2 mod 3, then all but 2 are inserted by means of "insert 3 ... ," and 2 are inserted by means of "insert 2 ... ." The net effect is to reduce expression (5a) for Algorithm B by a small constant, which is zero when $m \equiv 0 \bmod 3$ and nonzero but quite a bit smaller than 1 when $m \not\equiv 0 \bmod 3$. We will denote this constant ϵ; it proves easy to show that $\epsilon \leq \frac{1}{12}$. We then have

$$\mathrm{HL}(m, n) - H_B(m, n) = m/12 - 1 + \theta_1 + \epsilon - \theta_2, \tag{7}$$

so that in this range, the difference is just $m/12 - c$, where $-2 < c < 1 + \epsilon$.

Consider now n in *any* range σ_d, $d \geq 3$. The analysis just given is still exact, since $\lfloor(\frac{5}{4})2^d\rfloor = (\frac{5}{4})2^d$ when $d \geq 3$. Consequently, (7) still holds in *every* range σ_d for $d \geq 3$.

Finally, we observe that (7) must yield a positive result whenever $m \geq 24$. This completes the proof of the theorem.

3. *A Still Better Improvement*

It is possible to improve (7) so that the term $m/12$ may be improved to $(\frac{31}{336})m$. The proof

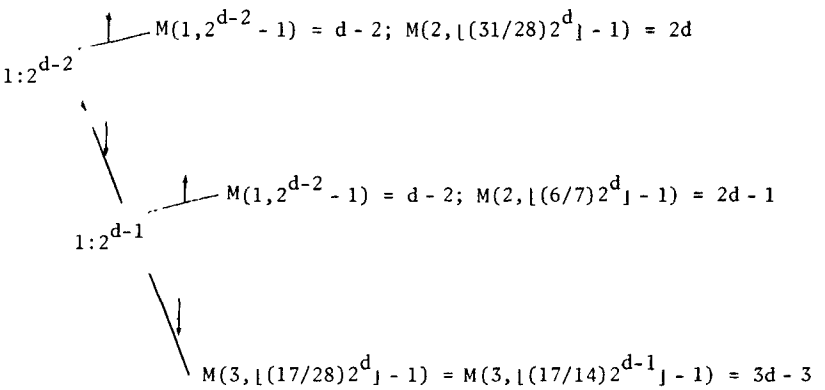

FIG. 5

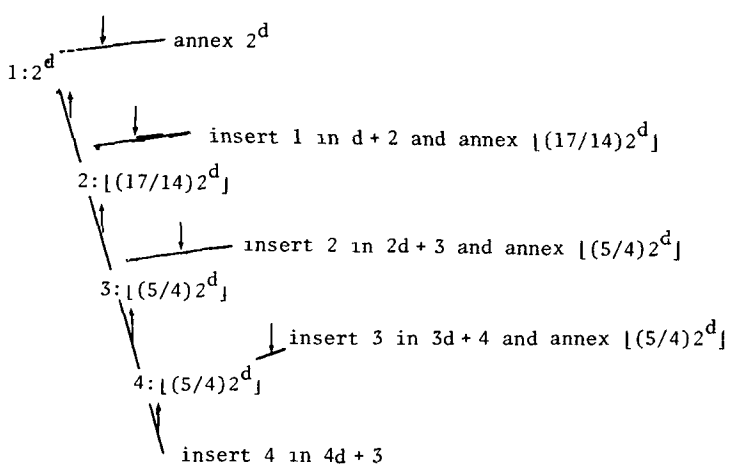

FIG 6. Algorithm B

will only be sketched, since this improvement is marginal.

LEMMA 5. *For* $d \geq 3$,

(*a*) $M(3, \lfloor(\frac{31}{28})2^d\rfloor - 1) \leq 3d - 1$,

(*b*) $M(3, \lfloor((\frac{5}{4}) + (\frac{3}{56}))2^d\rfloor - 1) \leq 3d$,

(*c*) $M(3, \lfloor((\frac{3}{2}) + (\frac{3}{28}))2^d\rfloor - 1) \leq 3d + 1$,

(*d*) $M(3, \lfloor q2^d\rfloor - 1) \leq 3d + 2$ *for* $(\frac{3}{2}) \leq q \leq (\frac{31}{14})$.

PROOF (sketch). Instead of proving (d) from Lemma 2, we observe that it is just a restatement of Lemma 3(a). We then prove (c) from (d) and (b) from (c), using the same diagrams as those for the corresponding steps in Lemma 3. Attempting the same thing for (a) from (b), we discover that we cannot improve (a). □

LEMMA 6

(*a*) $M(4, \lfloor(1 + \frac{3}{56})2^d\rfloor - 1) \leq 4d - 2$.

(*b*) $M(4, \lfloor((\frac{5}{4}) + (\frac{3}{112}))2^d\rfloor - 1) \leq 4d - 1$.

PROOF (sketch). (a) may be proved by means of essentially the same diagrams as those of Figures 4, 5, etc.; one needs to put in 3's for 2's and 4's for 3's. By using Lemma 3(a) and 3(c) carefully and noting that $M(4, \lfloor((\frac{1}{2}) + (\frac{3}{56}))2^d\rfloor - 1) = M(4, \lfloor(1 + \frac{3}{28})2^{d-1}\rfloor - 1) \leq M(4, \lfloor(\frac{5}{4})2^{d-1}\rfloor - 1) \leq 4d - 5$ by the HLA, one completes the proof. (b) follows from (a) in the same fashion that Lemma 5(c) is derived from 5(d) □

One now uses Lemma 6(b) to substitute 4: $\lfloor((\frac{5}{4}) + (\frac{3}{112}))2^d\rfloor$ for 4: $\lfloor(\frac{5}{4})2^d\rfloor$ in Algorithm B. The result is to decrease the cost per element inserted by $\frac{1}{112}$. The coefficient of m in (7) is then improved to $\frac{1}{12} + \frac{1}{112} = \frac{31}{336}$.

4. *Concluding Remarks on Another Class of Algorithms*

Another algorithm in the literature dominates the HLA for probably more values of m and n than the present algorithm. This is the algorithm of Hwang and Deutsch [2], which is optimal over a class of Algorithms R, all of which iteratively perform the following steps:

1 Determine dynamically which array is smallest, call it small and the other large

2 Compare small [1] large [y], or large [1] small [x]

3 If small [1] were compared with large [y] in step 2, and small [1] < large [y], then insert small [1] into {large [1], , large [$y - 1$]}, if small [1] ≥ large [y], then annex y elements A symmetrical operation is performed if large [1] is compared with small [x] in step 2.

4 Perform steps 1 through 3 until one of the lists is exhausted

Hwang and Deutsch show (a) that the optimal algorithm in R *always* performs small [1]: large [y], and (b) that for this optimal algorithm, it is possible to specify y inductively in terms of m and n.

The domain of dominance over the HLA appears to be as robust as ours, if not more so. However, the improvement for fixed n/m over the HLA increases more slowly than linearly in m [8].

REFERENCES

1 GRAHAM, R L On sorting by comparisons Proc Second Atlas Conf, 1971

2 HWANG, F K, AND DEUTSCH, D N A class of merging algorithms *J ACM 20*, 1 (Jan 1973), 148–159

3 HWANG, F K, AND LIN, S Optimal merging of two elements with n elements *Acta Informatica 1* (1971), 145–158

4 HWANG, F K, AND LIN, S A simple algorithm for merging two disjoint linearly-ordered sets *SIAM J Comptng 1*, 1 (1972), 31–39

5 HWANG, F K, AND LIN, S Some optimality results in merging two disjoint linearly-ordered sets Internal Memo, Bell Laboratories, Murray Hill, N J, 1972

6 KNUTH, D *The Art of Computer Programming*, Vol 3 *Sorting and Searching* Addison-Wesley, Reading, Mass, 1973, pp 181–207

7 MANACHER, G The Ford-Johnson sorting algorithm is not optimal *J ACM 26*, 3 (July 1979), 441–456

8 HWANG, F K, Private Communication

RECEIVED AUGUST 1977, REVISED AUGUST 1978