Department of APPLIED MATHEMATICS

An algorithm for internal marging of two subsets with small extra storage requirements.

by

Terje O. Espelid

Report No. 50

September 1974



UNIVERSITY OF BERGEN Bergen, Norway



An algorithm for internal merging of two subsets with small extra storage requirements.

by

Terje O. Espelid

Report No. 50

September 1974

Abstract

An algorithm for internal merging of two disjoint linearly ordered subsets into one set is given and analysed. The two subsets are supposed to be given in one array with interlacing elements. The algorithm is based on an interchanging of elements and requires therefore only a fixed amount of extra storage.

366

September 1974

11. 1. 1. 1. A. J.

edana i a anti-arrenda e esta, ai a displate a sector anti-arrende in a an set in given and en a the sector anti-arrende in a anti-arrende and the sector and arrende anti-arrende anti-arr tradicional and ender each arrende anti-arr tradicional arrende and each arrende anti-arrende antitradicional arrende and each arrende anti-arrende antitradicional arrende and each arrende anti-arrende antitradicional arrende and each each arrende anti-arrende

· · · · · · · · · · ·

Introduction

Given a 2-ordered array a[l:n], that is

- 1 -

(1)
$$a[i] \le a[i+2]$$
, $i = l(1)n-2$,

see Knuth [3] p.86.

The array consists of two disjoint linearly ordered subsets. Now the problem is to rearrange the elements in a[l:n] such that the resulting array a'[l:n] is l-ordered.

(2)
$$a'[i] \le a'[i+1]$$
, $i = l(1)n-1$.

This obviously is a merging problem and can be solved in different ways with and without extra storage. In this paper we will only consider methods which work without using a working area. For the purpose of analysis we will assume that $a[i] \neq a[j]$, $\forall i \neq j$. Furthermore the possible permutations of the elements indexes in a' relative to a, $\binom{n}{\lfloor n/2 \rfloor}^*$, are supposed to be equiprobable.

Sifting or straight insertion is one well-known algorithm which can be used. This algorithm is a sorting algorithm, that is, it does not take into account the special form, (1), of the array in question. On the other hand this algorithm is easy to program and analysis in Espelid [2] and [3], shows that the number of comparisons in average will be

 $C_{\rm S}$ (n) \approx .1566643 n \sqrt{n} + n .

[x]([x]) means the greatest (smallest) integer not greater (smaller) than x.

see Knuth (3) p.86.

The array constants of two disjoint linearly ordered subsets. . Now the problem is to rearrange the elements in ailen) such that the securities array a ling is in produced.

(2) a'll] < a'l14] , [142] a > [11] a

This obviously is a marging problem and can be solved in different ways with and without extra storage. In this paper we will only consider methods which work without using a working these. For the purpose of ensiyals we will assume that all is adjin visting a furthermore the possible parauterions of the elements indexes in a relative to a ([a]), are supposed to be equiprobable.

Sifeins on eventshi hearting is a sorting algorithm, which one he wood. This algorithm, is a sorting algorithm, thet is, it does not take into secount the special form, (1), of the areas in question. On the order hand this algorithm is easy to program and analysis in reputed (2) and [3], shown that he program of compactedes in evenues will be

Cg (n) # .1568643 n## (n)

[%] ([x]) means the greatest (smalless) integer no

Merging algorithms which make use of a working area will need maximum n-l comparisons to merge the subsets in a[1:n]. The sifting algorithm therefore makes a lot of superfluous comparisons. It is reasonable however that a method which makes use of a minimum of extra storage has to pay for it in longer running time, compared to usual merging algorithms. Batcher's (parallel) method, see [1], is a sorting algorithm which is based on merging the subsets of 2-ordered arrays taking into account that the arrays really are 2-ordered, and thus reducing the number of comparisons in average compared to sifting. We find that using the main idea in Batcher's method the number of comparisons (independent of a) will be

$$C_{B}(n) \approx \frac{1}{2} n \left[\log_{2} n \right] - \frac{1}{2} n$$
.

The number of comparisons is reduced compared to sifting but unfortunately the amount of bookkeeping needed to control the sequence of comparisons is rather large. Which of the two methods one should choose is not obvious and needs a more thorough analysis. The main power of Batcher's method lies in the possibility of parallel processing.

Still another method on a related problem is given by Kronrod [4]. Kronrod's algorithm seems complicated but he succeeds in forcing the number of comparisons down to

$$C_{K}(n) = O(n) .$$

The number of comparisons when using sifting will at minimum be n-1 and at maximum be $O(n^2)$. We get the maximum when

- 2 -

Cate - Lagoil a f a (a) a

C. Cal = cal .

a[1] is greater than the element with largest even index, a[2[n/2]]. The number of exchanges in average will be of the same magnitude as the number of comparisons for all three methods.

We now consider a new method which uses at maximum n-l comparisons to do the merging. The number of exchanges however turn out to be only slightly better than in sifting.

A merge exchange algorithm

To clarify the idea we will consider an example. Suppose that a[1:9] is given by

i	1	2	3	4	5	6	7	8	9
a[i]	5	l	6	2	7	3	9	4	10

The sifting algorithm will need 17 comparisons and 10 exchanges to sort this array. By comparing a[1] to a[i], i = 2,4,6,8 only 4 comparisons are needed. Now the problem is how to exchange elements without using more than one intermediate record, say w, such that at least one element comes to its final position in each step. We start with

w ← a[1]

all is greater than the element with lungest even index, all all all of the number of exchanges in everages will be of the same asgnitude as the sumbin of comparisons for all threes nethods.

we now consider a new method which thes at maximum n-1 comparisons to do the mangings. The number of exchanges however turn out to be only slightly better than in sifting.

n and a second state of the second second

To claraty the idea was will consider an example ... Suppose that all pl: is given by

		to account to reach a the second of a		

The sect this areas. By comparing all to all it a 2.4.5.8. to sort this areas. By comparing all to all i.i = 2.4.5.8. only 4 comparisons are needed. Now the problem is how to exchange elements without dains more than one intermediate. topoid, say, w, such that at least one element comes to its final position is each stop. We start with

then, the sequence

$$a[1] \leftarrow a[2x1]; a[2] \leftarrow a[2x2]; a[4] \leftarrow a[2x4];$$

Now a[8] is free to use and we move elements in the opposite direction

$$a[8] \leftarrow a[8-2^{\circ}]; a[7] \leftarrow a[7-2^{\perp}]; a[5] \leftarrow w;$$

The last move had to be made because the original $a[5-2^2]$ has been moved to w at the beginning. Now the array is changed to



where an arrow shows the direction in which an element has been moved, connecting the positions involved. We have to finish with

 $w \leftarrow a[3]; a[3] \leftarrow a[2x3]; a[6] \leftarrow w;$

This completes the merging in 8 moves, neglecting the moves from a to w.

One could now extend the problem putting a[10] = 8. 5 comparisons are needed to state that a[8] < a[1] < a[10]. The same procedure now gives in the first step

i	1	2	3	4	5	6	7	8	9	10	1 -	¢	7
a"[i]	1	2	3	4	5	6	7	9	10	8			

We know that the elements a"[1:5] are correctly sorted and are less than the other elements. By two comparisons we find that a"[6] and a"[7] both are less than a"[10] (= a[10]) such that the first 7 elements have found the final position. a[1] - a[2x1] ; a[2] - a[2x2] ; a[4] - a[2x4] ;

Now a[8] is free to use and we move elements in the opposite direction

(6)
$$\leftarrow a(8-2^{0})$$
; $a(7) \leftarrow a(7-2^{1})$; $a(5) \leftarrow w$;

The last move had to be made because the original $a[5-2^4]$ has been moved to w at the beginning. Now the array is changed to



where an arrow shows the direction in which an element has been moved, connecting the positions involved. We have to finish with

One could now extend the problem putting all0] = 8 -5 comparisons are needed to state that a[8] < a[1] < a[10] -The same procedure now gives in the first step

We know that the elements a"[1:5] are correctly sorted and are less than the other elements. By two comparisons we find that e"[6] and a"[7] both are less than a"[10] (* a[10]) - such that the first 7 elements have "ound the final position. The sorting problem left is not exactly of the original form. We still have two subsets but the interlacing character starts with the second element. We therefore need to generalize the problem slightly. Let us now leave the example.

Given an array a[1:n] where the b first elements have found their final position, the elements a[b+1], ..., a[n]represent the merging problem remaining. These elements consist of two disjoint linearly-ordered subsets. The first l elements, $1 \le l \le n-b$, all belong to one of the subsets, element l+1belongs to the other subset and from this point the array is 2-ordered, that is

 $a[b+1] \le a[b+2] \le \dots \le a[b+l] \le a[b+l+2] \le a[b+l+4] \le \dots$ (6)

 $a[b+l+1] \le a[b+l+3] \le \dots$

The following flowchart gives the main points in the algorithm:

AL ANTINE SCORE SET IN THE SECOND SECOND CONSTRUCTS AN AND NOT TO PURCE BE IN INTERACTE CONSERVE SCORE AND IN THE SECOND SECOND SECOND IN PERSION ON A SECOND SECOND ADDITION SET AND SECOND SECOND IN A SECOND SECO

alb+\$+11 ≤ alb+\$+3] ≤ ...

the following flowohang. gres the main points, in the states in the second seco





We have removed from the flowchart the details in connection with moving the elements around successively. When one starts to move elements around, it is clear that exactly t of the elements from the other subset are less than a[b+1]. The actual indexes relative to b are

where all the circled indexes belong to the same subset. After the move of elements phase, we shall have (relative to b)

new index 1 2 ... t t+1 t+2 t+l t+2+1 t+2+2+2t-1 old index l+1 l+3 l+2t-1 1 2 ... l l+2 l+4 ...l+2t-2

This gives the following connection between the new index, b+new, and the old one, b+old,

 $\begin{cases} 1 \le \text{new} \le t & : \text{ old } \leftarrow \ell + 2\text{new} - 1 ; \\ t \le \text{new} \le \ell + 2t - 1 : \text{ old } \leftarrow \underline{\text{if new}} \le t + \ell \underline{\text{then new}} - t \\ \underline{\text{else}} 2(\text{new} - t) - \ell ; \end{cases}$

Here b+new is supposed to be the index of an element in a[l:n] which is free to use. The problem is therefore to compute the index (old) of the element which shall be moved to a[b+new] and so on in a cyclic manner. To start the process one moves a[b+1] to w and then defines i $\leftarrow 1$. Some additional administration of the moves to and from w is also needed. The details are found in the algol program at the end of this paper, where this problem is solved in a self-explanatory manner.

We have removed from the flowohart the details in connection with moving the elements around successively: when one starts to move elements around, it is clear that exactly t of the elements from the other subset are less that affills The elements indexes, relative to b are

Where all the circled indexer belong to the same subset. After the move of elements phase, we shall have (relative to b)

Denges the following connection between the new index, '

1 S new S.t. : · 01d <- Stlewell; ·

t s new s \$+2t-1 : old & if new s the then new-t

Here, benew is supposed to beacher index of an element in all of which is free to see. The problem is therefore to compute the index, (old) of the slement which shall be moved to "s(break]" and so on in a cyclicizanner. To start the precess one noves alphilitto, w and chemidefines i and the index is similaritonal edministration of the proves to and from w is simo needed. The details are found in the algol program if the end of this paper.

Analysis of the merge exchange algorithm

We will now concentrate on the analysis of the merging problem given in (6). As in the flowchart we define $r \equiv n-b$, that is the number of elements left to merge. Let s be the number of elements in the subset which contains a[b+l+1], $s \equiv \lfloor (r+1-l)/2 \rfloor$. This means that we are going to compare a[b+1] successively with from 1 up to s elements before the final position of a[b+1] is found. Now we have supposed that all the possible different final permutations, $\binom{r}{s}$, of this merging problem are equiprobable. This means that we can give the probability for each possible number of comparisons or equivalent for each possible number of exchanges (or moves).

Define

(7)
$$\begin{cases} P_{1} = Prob \{a[b+1] < a[b+l+1]\} \\ P_{j} = Prob \{a[b+l+2j-3] < a[b+1] < a[b+l+2j-1]\}, \\ j = 2(1)s. \end{cases}$$
$$P_{s+1} = Prob \{a[b+l+2s-1] < a[b+1]\} \end{cases}$$

We find by our assumption that

(8)
$$\begin{cases} P_{1} = \frac{\binom{r-1}{s}}{\binom{r}{s}} = (r-s)/r \\ P_{j+1} = \frac{\binom{r-j-1}{s-j}}{\binom{r}{s}} = P_{j}(s-j+1)/(r-j) , j = 1(1)s . \end{cases}$$

- 8 ---

Indiate of the seven seven set to ersyland

are and a second and a second and a second and a second a

Define

We fand by our assumption that

Now let $c_{r,l}$ and $e_{r,l}$ mean the average number of comparisons and exchanges (or moves) needed to merge a[b+1:n] by the algorithm. Note that $c_{r,r} = e_{r,r} = 0$. We find the following recurrence relations

(9)
$$c_{r,l} = p_1(c_{r-1,max(1,l-1)} + 1) + p_{s+1} \cdot s$$

C

$$\frac{\sum_{j=2}^{p} p_{j}(c_{r-j}, l+j-2 + j)}{and s = \lfloor (r+l-1)/2 \rfloor}$$

(10)

$$e_{r,l} = P_{l}(e_{r-l,max(l,l-l)} + P_{s+l}(l+2s-l) + \sum_{j=2}^{s} P_{j}(e_{r-j,l+j-2} + l+2j-3), l = l(l)r-l$$
and $s = \lfloor (r+l-l)/2 \rfloor$

We note that the comparisons in our algorithm behave just like an ordinary merge algorithm on two disjoint linearly ordered subsets. This means that $c_{r,s}$ is given by

(11)
$$c_{r,l} = s(r-s)/(s+1) + s(r-s)/(r-s+1)$$

see for example [3].

This gives

(12)
$$c_{r,1} = (1/(1+\lfloor r/2 \rfloor) + 1/(1+\lceil r/2 \rceil)) \cdot \lfloor r/2 \rfloor \cdot \lceil r/2 \rceil$$

To solve (10) seems considerably more difficult. Using (10) we have computed the first values of $e_{r,l}$, l = l(l)r-l in table 1.

- 6

- 10 -

r

2

2	T.0000					
3	1.3333	1.6667				
4	2.3333	1.7500	2.2500			
5	2.8000	3.3000	2.2000	2.8000		
6	3.8500	4.4667	4.2667	2.6667	3.3333	
7	4.4000	5.0286	4.2381	5.2381	3.1428	3.8571

 $e_{r,l}$, l = l(1)r-1

Table 1.

We are now interested in finding an approximate expression for $e_{r,1}$. Therefore we tabulate, using (10), $e_{r,1}$ for $r = 2^{j}$, j = 1(1)8 in table 2.

r	e _r ,	1
2	1.00000	00000
4	2.33333	33333
8	5.50000	00000
16	13.04607	61461
32	31.21811	86596
64	75.73013	95663
28	187.05514	13464
56	471.62559	05589

Table 2.

Using the results in [2,3] we make the guess that

(13)

1 2

 $g_r = e_{r,1}/r = \alpha r^{\frac{1}{2}} + \beta + \gamma r^{-\frac{1}{2}} + \delta r^{-1} + \dots$

2 1.0020 4 1.3332; 1.6667 2.3332; 1.7500 2.2600 2.3332; 1.7500 2.2600 3.3332 4.3500 4.5667 4.2667 2.8667 3.3333 4.41000 5.0286 4.2381 5.2867 3.3338

er, 1 + 4 = 1(1)r-1

Table 1.

We due now interested in finding an approximate expression form $e_{r,1}$. Therefore we takidiste, using (10): $e_{r,1}$ for $r = 2^{\frac{1}{2}}$. $j_{j} = 1(4)$ in table 2.

a) () () ()	0000040	
45	. 1886	
0.000	60.8003.00	
	18.04607	
D ai g B L	187.05514	

Table 2.

Using the results in [2,3] we make the guess that

1.8.1

Using extrapolations on the values in table 2 and taking into. account the form (13) we get the results in table 3 defining

$$h_{p} = (g_{2p} - g_{p})(\sqrt{2} + 1)/\sqrt{r}$$

This extrapolation turns out to be rather successful and gives

Using the same method to estimate β and γ one finds

$$\begin{cases} \beta \approx .6250 \\ \gamma \approx -.607 \end{cases}$$

Note that

2a ≈ .1566643

which seems to be exactly the same constant as given in [2] for the sifting algorithm used on the same problem. Knuth shows in his analysis [3] that this constant is

20 × VT/128.

It is remarkable however that the numerical procedure used in [2] comes out with 7 significant digits. Maing extragolations on the values in thirs 2 and texing form

inte statepolation tento ave se presentation austressive

Ranis and i the d steations of particle dis all all all

Note that

2a + .1566643

who effect to be exactly the are constant to gaven in (*) for the effect of original used on the same prophety. Knuth shows to big anotypic (*) view this equations in

20 % / F/ 12

it is a marking hower that the preprint) propodure weed to be come with 7 second company.

8/2-extra-	polation	•						.07833215											
8-extra-	uotiptod						.07833413		.07833233										
4/2-extra-	ротантоп					.07831123		.07833127		.07833219									- + + + · · · · · · · · · · · · · · · ·
4-extra-	POTA CTOIL				.07836994		.07832161		.07832956		.07833173								he numbers :
2V2-extra-				.08345379		.07364090		.07865143		.07841003		*07835130							ecimals and +
2-extra- Dolation			.10922158		.09256411		.08420995		.08051666		·07919019		.07864789						ne with 10 d
ц ц		.14225890		.12574029		.10915220		,09668107		,08864887	ica:	.08391953		.08128371					have been do
50			. 583333333		• 6875		.81537976		.97556621		1.18328343		I.46136829		1.84228746				SUCTIPINATIONS
ы.	2	1	4		00		16		32		64		128		256			Tho o	TILC

- 12 -

e numbers in this table are given correctly rounded to 8 decimals.

.

Table 3

atomicae i of beba

								61250							stry out	573X:
								ANG.								
							1014633533		· 01933473						Bagartyou	8. estimates
						578-97J3		1028381SA		76:10%7¢.						-SITXO-NA
					0.633713		A-9-12870.		*CUSSADIET		C.V. 8.2.2.1.2.				ັນດູ່ເປະບານ	
				1016351350-		STOL AND STOL						5.0.8 Shirt S. 18			nnoidsioq	SALLEXX. TAR
			* 614 8. 01 12 80 C		419110T0.		10 63 61 6 64		15601480.		.0882647.		・10月末月19日		the curst of the set	S S Marine .
		· Q.G. J. S. 8 3 & F		30010380.				19 [x3369].		116412820		11581+058				
	1.20.2502A8		T. 4613		四十二年月四年日, 日		1979297976		5797978.		97.83.		2233			and Ge
	0		2.0		41 ()				30		62			35		

- 21

Conclusions

We have found that

$$C_{M}(n) = (1/(\lfloor n/2 \rfloor + 1) + 1/(\lceil n/2 \rceil + 1)) \lceil n/2 \rceil \lfloor n/2 \rfloor \approx n - 2$$

and

$$E_{M}(n) \approx .078332 n^{3/2} + .6250 n - .607 n^{1/2}$$

for this new merge exchange algorithm. This makes the algorithm considerably better than sifting when only comparisons and moves are taken into account. Asymptotically we will have $C_{M}(n) + E_{M}(n) \sim 1/4(C_{S}(n) + E_{S}(n))$. In table 4 these expressions are compared for

n	$C_{S}(n) + E_{S}(n)$	$C_{M}(n) + E_{M}(n)$	
16	35	27	. 77
32	88	61	.69
64	224	138	.62
128	581	313	.54
256	1539	726	.471

Table 4

smaller values of n.

Unfortunately the amount of book-keeping in this new method, just as for Batcher's parallel method, is rather large. One way to improve the method might be to increase the working area. The author has not studied how this might influence the bookkeeping problem, and the question how much this would reduce the amount of work therefore remains open.

Tadd bruch even eW

E_M(n) # .078332 n^{3/2} + .6250 n -.607 n^{1/2}

for this new marge exchange algorithm. This nakes the algorithm considerably better than sifting when only comparisons and moves are taken into account. Asymptotically we will have $C_{\rm H}(n) + E_{\rm M}(n) \sim 1/4(C_{\rm S}(n)) + E_{\rm S}(n))$. In table 4 these expressions are compared for

	$C_{M}(n) + E_{M}(n)$	$c_{g}(a) + E_{g}(a)$	
77.		28	- J.F
₽ð.		- 89	
.62	3. F	224	$\frac{1}{2}$ $a = \frac{1}{2}$
₽ð.	313		691
			803

Unfortupately the amount of book-keeping in this new nethod, just as for Batcher's parallal nathod, is rather large. One way to improve the meteod night be to increase the working area. The author has not studied bor this anget influence the hoden to keeping problem, and the question how much this would reduce the anount of work tharefree remains open.

The Algol program

The algol program given has been written only to show how this merge exchange algorithm works. As is seen by a first glance, the program is not optimal. One has tried to use the same notation in this program as in the text. Introducing too many new helpvariables has been avoided although this would have speeded up the program. The author hopes that this fact combined with reading the text makes the program selfexplanatory without too many comments.

procedure Merge_exchange (a,n);

value n; integer n; integer array a;

- <u>comment</u> This procedure transforms the 2-ordered array a[1:n] to a 1-ordered array a[1:n] with small extra storage requirements;
- integer b, j, l, r, t, w, new, old, count, windex; b < 0; l < 1; comment b: basis pointer, l: see text;

start : $r \leftarrow n-b$; <u>comment</u> r: number of elements left; <u>if $l \ge r$ then go to</u> fin; $w \leftarrow a[b+1]; j \leftarrow b+l+1;$

if w < a[j] then

begin b ← b+1; l ← if l=1 then l else l-1; go to start; end w has found its final place; t ← 1;

for $j \leftarrow j+2$ while $j \leq n$ do

if $w \leq a[j]$ then go to move else t \leftarrow t+1;

Mergora LogIA 9HV

The algel Divgrah given has been written only to show how this merge exchange algorithm works. As is seen by a first glance, the program is not Optimal. One has tried to use the same notation in this program as in the text. Introducing too many hew helpvariables has been avoided although this would have speeded up the program. The author hopes that this featdefibined with reading the text makes the program solfexplanatory without too many comments.

> <u>cocedure</u> (torge_exchange_(a.n); value n: integer n: integer acrev a:

<u>comment</u> This procedure transforms the 2-ordered array a[1:n] to a 1-ordered array a[1:n] with small extra storage requirements:

interret
 b, 1, 2, 2, t, w, new, old, count, windex;
 b <-0; 1 <-1; conment b: basis pointer,
 cate text;
 conment b: see text;
</pre>

or j * 1+2 while j < n do

move : count < 0; windex < b+1; new < b+1; comment We shall move l+2t-1 elements. Control : count;

right : old < 2×new-b+l-l; count < count+l; a[new] < a[old]; new < old; if new < b+t then go to right;</pre>

left : old < if new < b+l+t then new-t else 2*(new-t)
 -l-b;
 comment Now there is the chance that old is equal
 to windex. In this case a cycle is finished and we
 have to check if there is more work to be done;
 if old=windex then</pre>

if count=l+2×t-1 then go to continue; for new windex+1 step 1 until b+l, b+l+2 step 2 until b+2×t-1 do begin if a[new] > w then begin w < a[new]; windex < new; go to right end; end; end The case old=windex is finished; a[new] < a[old]; new < old; count < count+1</pre>

go if new<b+t then right else left; continue : b < b+t-1; l < l+t-1; go to start fin : end Merge_exchange;

L New S D+t then do to the stant,

old «<u>if</u> new <u>s</u> beitt <u>then</u> new-t <u>else</u> 2% heW=E) -9-b:

comment Now there is the chance that old is squal to windex. In this case a cycle is finished and we have to check if there is more work to be dens; if old=windex then

oggin afnewl www.count *-count#1;

If alther that the point for the contributer of the second sec

References

- [1] Batcher, K.E.: "Sorting networks and their application" Proc. AFIPS Spring Joint Comp. Conf. 32(1968) 307-314.
- [2] Espelid, T.O.: "Analysis of a Shellsort Algorithm" BIT 13,4 (1973) 394-400.
- [3] Knuth, D.E.: "The Art of Computer Programming", Vol. 3, Addison Wesley, Reading, Mass. 1973.
- [4] Kronrod, M.A.: "Optimal ordering algorithm without operational field" Soviet Math. Dokl. 10 (1969) 744-746 (Russian).

- The secondary which is a second of a second second second second in the second se
- Addison_Reality, Reading, Mass. [373;
 - File Mpphrod, M.A.: "Optimal Gropring algorithin without operational field"
 - Soviet Math. Dokl. 10 (1983) 7/H-746 (Burgsin).





