Merging by Decomposition Revisited

S. DVOŘÁK*† AND B. ĎURIAN‡

† *Centre of Tesla Rožnov, 1. máje 1000, 75661 Rožnov pod Radhoštěm, Czechoslovakia*

‡ *Research Institute of Computer Technology (VÚVT) Žilina, Nerudová 33, 01001 Žilina, Czechoslovakia*

This paper presents some modifications of stable merging by decomposition (referred to as DM here). The changes made reduce the time requirements considerably. Furthermore, a O(1)-space version of merging is described. The modifications of DM resemble improvements to the original Quicksort method for sorting, since both the algorithms are of the same generic scheme.

Received September 1986, revised March 1987

1. INTRODUCTION

In this article we are dealing with stable merging (&) of two segments $\mathscr{D} = A(1..m)$, $\mathscr{H} = A(m+1..n)$ of a one-dimensional array A into $A(1..n)$. A merging method is said to be stable if it preserves the relative order of records with identical keys. We assume non-decreasing sequences of integers in the merged segments. The generalisation for records with an arbitrary structure of keys and with a non-empty information part is straightforward.

An interesting merging method based upon a decomposition has been developed by Pratt[1] and simplified by Dudzinski and Dydek afterwards.[3] In what follows, we shall tune the latter procedure. Its generic scheme recalls immediately the famous Quicksort, suggested by C. A. R. Hoare, while the changes we are suggesting for DM in turn recall the changes implemented in the original Quicksort during its history.

2. THE GENERIC SCHEME OF DM

DM in Dudzinski and Dydek's version is based upon a decomposition of a merging problem into two subproblems of smaller size which is carried out in Fig. 1. In the shortest segment of those merged together ($\mathscr{D} = \mathscr{D}_1 x \mathscr{D}_2$ in Fig. 1, $|\mathscr{D}| \leqslant |\mathscr{H}|$) the median x is taken as a pivot for decomposition and upper segment $\mathscr{H}$ is split into

$$\mathscr{H}_1 = \{\xi \mid \xi \in \mathscr{H}, \xi < x\}, \quad \mathscr{H}_2 = \{\xi \mid \xi \in \mathscr{H}_1, \xi \geqslant x\}.$$

The inequalities in the definitions of $\mathscr{H}_1$ and $\mathscr{H}_2$ are to be obeyed for the merging to be stable. Then $x\mathscr{D}_2$ and $\mathscr{H}_1$ are exchanged, giving the arrangement

$$\mathscr{D}_1 \mathscr{H}_1 x \mathscr{D}_2 \mathscr{H}_2 \qquad (1)$$

with pivot x in the proper place. The original problem is decomposed into two smaller problems $\mathscr{D}_1$ & $\mathscr{H}_1$ and $\mathscr{D}_2$ & $\mathscr{H}_2$ now.

If the shortest segment is $\mathscr{H}$, we take x as median of $\mathscr{H} = \mathscr{H}_1 x \mathscr{H}_2$. Then sets

$$\mathscr{D}_1 = \{\xi \mid \xi \in \mathscr{D}, \xi \leqslant x\}, \quad \mathscr{D}_2 = \{\xi \mid \xi \in \mathscr{D}, \xi > x\}$$

are determined and the exchange $\mathscr{H}_1 x$ and $\mathscr{D}_2$ leads to decomposition (1) again.

The same procedure is applied now for merging $\mathscr{D}_1$ & $\mathscr{H}_1$ and $\mathscr{D}_2$ & $\mathscr{H}_2$; we arrive quite naturally at the following recursive merging procedure:

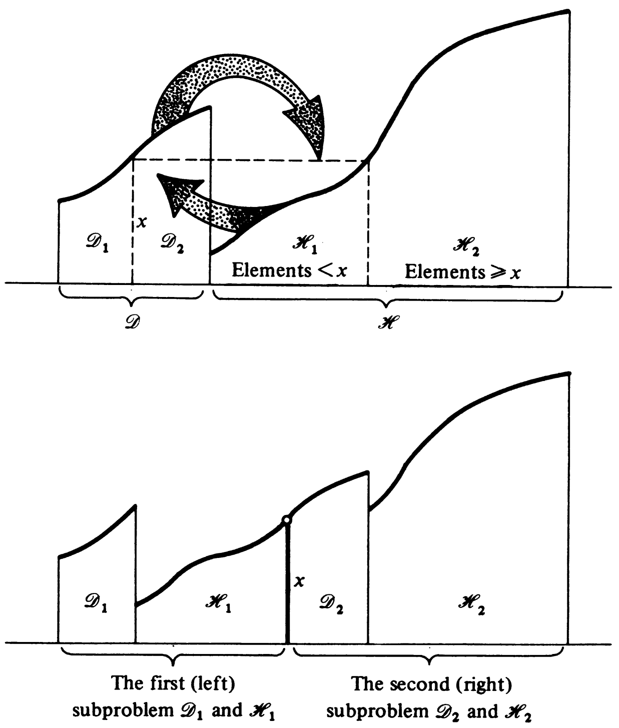

Figure 1

```
procedure DecMerge ($\mathscr{D}$, $\mathscr{H}$):
  if $|\mathscr{D}| > 0$ and $|\mathscr{H}| > 0$ then
    find the decomposition of $\mathscr{D}\mathscr{H}$;
    exchange middle parts as stated above
        to receive the arrangement $\mathscr{D}_1\mathscr{H}_1 x\mathscr{D}_2\mathscr{H}_2$;
    DecMerge ($\mathscr{D}_1$, $\mathscr{H}_1$);
    DecMerge ($\mathscr{D}_2$, $\mathscr{H}_2$)
  endif
endproc DecMerge;
```

But the above scheme is the classical one well known from different algorithms. The same scheme is employed in one of the most successful sorting methods – Quicksort.[4, 5]

Quicksort kept improving for more than 20 years to reach the present level of perfection. We have implemented similar improvements in DM. The consequences of these changes are even more expressive than in the Quicksort.

* To whom correspondence should be addressed.

3. DM TUNING

We propose the following changes in DM (in Dudzinski and Dydek's version):

– a modification in the decomposition itself (by the way, the original algorithm contains some serious flaws in this part),

– the use of rotational exchanges (following Pratt's technique),

– the final merging steps are performed together by direct merging with a small workspace,

– recursion has been eliminated and replaced by the direct control of a stack.

Now we shall describe some details of these changes.

(A) *Decomposition.* We avoid the unnecessary steps in decompositions. Let us assume we are to solve the particular merging problem

$$\underbrace{A(dd..dh)}_{\mathscr{D}} \& \underbrace{A(hd..hh)}_{\mathscr{H}} \rightarrow A(dd..hh); \quad hd = dh+1.$$

Assume that $|\mathscr{D}| \leqslant |\mathscr{H}|$ and let s be the index of median in $\mathscr{D}$. For $x = A(s)$, $\mathscr{D} = \mathscr{D}_1 x \mathscr{D}_2$ it holds:

– if $x \leqslant A(hd)$, the whole segment $\mathscr{D}_1 x$ occupies the beginning of the merged segments and we have to merge $\mathscr{D}_2$ and $\mathscr{H}$ only:

$$A(s+1..dh) \& A(hd..hh) \rightarrow A(s+1..hh);$$

– if $x > A(hh)$ then $x\mathscr{D}_2 = A(s..dh)$ and $\mathscr{H}$ will be exchanged. $x\mathscr{D}_2$ is in the proper place then and it remains to merge $\mathscr{D}_1$ and $\mathscr{H}$;

– otherwise $A(hd) < x \leqslant A(hh)$. Therefore, there is a $t \in [hd, hh]$ such that

$$A(t) < x \leqslant A(t+1).$$

The subscript t will be determined by the binary search in $[hd, hh]$; t defines the decomposition $\mathscr{H} = \mathscr{H}_1 \mathscr{H}_2$ with

$$\mathscr{H}_1 = A(hd..t) = \{\xi \mid \xi \in \mathscr{H}, \xi < x\},$$
$$\mathscr{H}_2 = A(t+1..hh) = \{\xi \mid \xi \in \mathscr{H}, \xi \geqslant x\}.$$

This is just the decomposition of $\mathscr{H}$ which we need according to DM description.

For $|\mathscr{D}| > |\mathscr{H}|$, similar steps are carried out. In this way we do not work with parts which remain in place.

(B) *Exchange.* The exchange of segments as specified above is one of the key phases of the algorithm. The algorithm used by Dudzinski and Dydek for this purpose can be considered as the implementation of the inverse permutation to

$$\pi = \left(\begin{array}{cccc|cccc} 0 & 1 & \dots & m-1 & m & m+1 & \dots & n-1 \\ n-m & n-m+1 & \dots & n-1 & 0 & 1 & \dots & n-m-1 \end{array}\right)$$

on $A(0..n-1)$. Although this method requires only $n + gcd(m, n)$ transfers of exchanged elements we found it to be slower than 'rotational' exchange in Pratt's DM. Exchange times for the former method are 130–160% of times for the latter. Therefore, we used rotational exchanges. If $Rot(a, b)$ is the rotation of $A(a..b)$,

```
procedure Rot(a, b):
  $i \leftarrow a; j \leftarrow b;$
  while $i < j$ do $A(i) \rightleftarrows A(j); i \leftarrow i+1; j \leftarrow j-1$ endloop
endproc Rot;
```

then the exchange of $A(dd..dh)$ and $A(hd..hh)$ is implemented as

```
procedure Xchg (dd, dh, hh):
  Rot (dd, dh); Rot (dh+1, hh); Rot (dd, hh)
endproc Xchg;
```

(C) *Recursion.* As well as the initial version of Quicksort, DM was presented as a recursive procedure. For Quicksort, it was soon recognised that it is better to control the stack of unsolved problems than to invoke the procedure recursively. For DM we applied the same idea putting into stack the middle and upper indices of the right subproblem. The lower index can be calculated from the upper index hh of the previous problem as $hh+2$.

(D) *Direct merging at the end.* For Quicksort, it has been proved to be advantageous to terminate partitioning as soon as the length of a segment to be partitioned falls below a certain limit. Then the sequence of short segments which are in correct relative order is sorted by a simple insertion technique. This would be possible for merging as well, if we were to implement merging without additional workspace to the stack.

But if we devote for merging workspace $W(1..\text{len}\,W)$, we are able to merge the segments directly (in one pass) as soon as the length of at least one of them is $\leqslant \text{len}\,W$. The workspace size can be rather small (we chose $\text{len}\,W = 16$). For instance, if $\mathscr{D} = A(dd..dh)$, $\mathscr{H} = A(hd..hh)$, $hd = dh+1$ and $|\mathscr{D}| \leqslant \text{len}\,W$, we move $\mathscr{D}$ into W and merge back from W and $\mathscr{H}$:

```
$s \leftarrow dd; t \leftarrow hd;$
for $i \leftarrow 1$ to $|\mathscr{D}|$ do
  $l \leftarrow W(i);$
  while $x > A(t)$ do $A(s) \leftarrow A(t); s \leftarrow s+1; t \leftarrow t+1$
  endloop;
  $A(s) \leftarrow x; s \leftarrow s+1$
endloop;
```

There is a sentinel ($>$ all the elements of $\mathscr{D}$, $\mathscr{H}$) assumed behind the last element of $\mathscr{H}$. For $|\mathscr{H}| \leqslant \text{len}\,W$ we proceed similarly. The sentinels are to be set only at the start. To transfer contiguous blocks, a fast block transfer can be employed if available. One-side merging alone leads to considerable time savings, for it eliminates the overhead in many final short exchanges.

The DM with all the suggested modifications is given in Pascal in the Appendix. Measured time values are collected in Table 1. Quicksort times were added for comparison. Since the efficiency of Quicksort is well known, this comparison reveals the cost of stability in merging.

The theoretical analysis of the tuned version of DM remains basically the same as in Dudzinski and Dydek,[3] therefore it is not repeated here. If longer records (of size reclen) are being considered, the time required for merging of m and $n-m$ records may be expressed as

$$T(n) = T_C * n \lg_2 m + T_M * \text{reclen} * \lg_2 (n/m),$$

where T_C is connected with comparisons and T_M with movements. Both T_C and T_M may be determined easily from merging time measurements for records of different lengths but with the same structure of keys.

Table 1. Timing of merging methods

n	Modified DM	Modified DM, no stack	Original DM	Quicksort
		Case $m = \frac{1}{4} * n$		
1000	434	634	1027	838
2000	1007	1426	2153	1871
3000	1654	2333	3460	2971
4000	2260	3254	4627	4080
5000	3014	4411	5953	5213
6000	3687	5267	7334	6359
7000	4320	6133	8507	7541
8000	5007	7212	9747	8750
9000	5766	8628	11073	9951
10000	6580	9693	12514	11131
		Case $m = \frac{1}{2} * n$		
1000	547	740	1347	1129
2000	1214	1647	2807	2441
3000	2013	2827	4434	3820
4000	2640	3641	5874	5859
5000	3646	5126	7580	7748
6000	4406	6238	9293	9080
7000	5054	7154	10867	11770
8000	5820	8052	12407	12021
9000	6893	9180	13920	15100
10000	7827	11039	15760	16600
		Case $m = \frac{3}{4} * n$		
1000	460	599	1120	879
2000	1020	1359	2394	1889
3000	1640	2332	3767	2959
4000	2253	3034	4966	4131
5000	3000	4395	6440	5240
6000	3600	5134	7767	6369
7000	4179	5940	9067	7541
8000	4840	6707	10226	8762
9000	5580	7512	11673	9869
10000	6313	9113	13113	11119

Working area in modified DM: 16 words. Measurements on samples of random integers, time in msec. Average values on three samples. Computer: KBR 1630. Programming language Fortran 77.

4. STABLE MERGING IN O(1)-SPACE

DM (the original as well as modified versions) allow for one more modification which leads to stable merging algorithm with 0(1) workspace only: the stack is no longer needed. This is the first merging algorithm of this type since 1977, when L. T. Pardo's work was published.[2] In the tested range ($n \leqslant 10^4$) the efficiency of our 0(1)-space version is much higher than that of Pardo's merging despite the claimed linear behaviour of the latter method.

There is no need to put the limits of the right-merging subproblem ($\mathscr{D}_2 \& \mathscr{H}_2$) into the stack: when the current merging subproblem is solved we shall look for the limits of the problem next to the one just solved. Therefore,

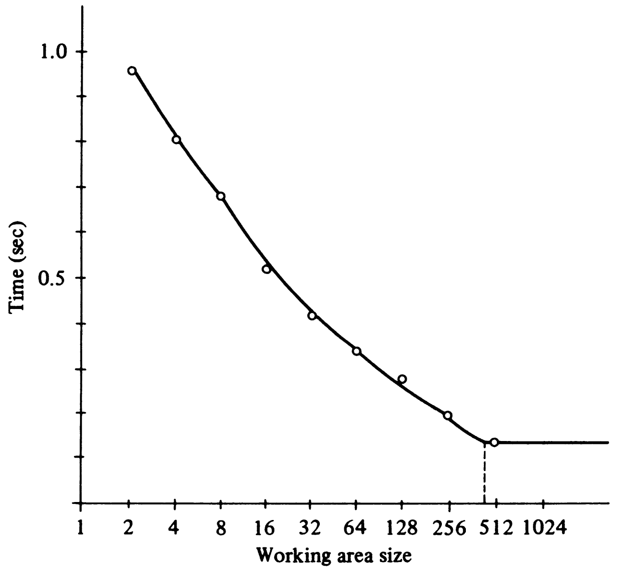

Figure 2

stack and stack operations (marked by underlining in the Appendix) are replaced by the following code:

```
while $hh < n$ do begin
  $dd := hh + 2$; $dh := dd$;
while $A[dh] \leqslant A[dh+1]$ do $dh := dh + 1$;
if $dh <= n$ then begin
  $hh := dh + 1$; $x := A[dh]$;
  while $x >= A[hh]$ do $hh := hh + 1$;
  $hh := hh - 1$; $hd := dh + 1$;
```

Some further minor changes are to be done. A sentinel is to be inserted into $A(n+2)$ in advance. It is clear that the search reduces the efficiency of DM. Anyway, the stackless version of DM remains still faster than Pardo's linear merging.

The same search can be embodied into Quicksort itself. This leads to a O(1)-space-sorting algorithm with only slightly lower efficiency as compared to standard Quicksort.[6]

5. CONCLUSION

The suggested changes in DM have radically improved its efficiency. Moreover, a O(1)-space version of DM has been described. This version – although slightly reduced in efficiency by the renewed search for segment limits – is still considerably faster than the only known method for stable O(1)-space merging (which is even linear in time, contrary to DM).

Let us recall that workspace size is a parameter in the modified DM; it can be varied widely according to available memory and required speed. The relationship between merging time and workspace size is illustrated in Fig. 2.

As far as we are aware, the modified DM as described here is the most efficient merging method for a workspace size less than $2 * n^{\frac{1}{2}}$. Above this limit the method of block merging is superior.[7,8] But there is still room for further promising improvements of DM.

REFERENCES

1. D. E. Knuth, The Art of Computer Programming, vol. III, *Sorting and Searching*. Addison-Wesley, Reading, Mass. (1975).
2. L. T. Pardo, Stable sorting and merging with optimal space and time bounds. *SIAM J. Comp.* **6** (2), 351–372 (1977).

3. K. Dudzinski and A. Dydek, On a stable minimum storage merging algorithm, *Inf. Proc. Lett.* **12** (1), 5–8 (1981).
4. C. A. R. Hoare, Alg. 64: Quicksort. *Comm. ACM* **4** (7) (1961).
5. R. Sedgewick, Implementing Quicksort programs. *Comm. ACM* **21** (10), 847–856 (1978).
6. B. Ďurian, Quicksort without a stack. *Proceedings of MFCS 86, Bratislava (August 1986)*. Springer-Verlag, Berlin (1986).
7. B. Ďurian, Stable merging in $0(\sqrt{N})$ memory and nonstable merging in 0(1) memory [in Slovak]. *Informačné systémy* (1), 67–86 (1985).
8. S. Dvořák and B. Ďurian, Stable linear time sublinear space merging. *The Computer Journal* **30** (4), 372–375 (1987).

APPENDIX

The algorithm of the modified DM-merging

```
procedure MERGE (var m,n: integer);

{The stable merging of A[1..m] and A[m+1..n] into
A[1..n]}
var d, h, dd, dh, hd, hh, i, s, t, x, bcorr: integer;
var lenW, lenD, lenH, top: integer;
var stckD,stckH,W: array [1..16] of integer;
label dflen;

procedure XCHG(1, m, u: integer);
var i, j, x: integer;
begin
  i:= 1; j:= m;
  while j > i do
    begin x:= A[i]; A[i]:= A[j]; A[j]:= x; i:= i+1;
        j:= j-1 end;
  i:= m+1; j:= u;
  while j > i do
    begin x:= A[i]; A[i]:= A[j]; A[j]:= x; i:= i+1;
        j:= j-1 end;
  i:= 1; j:= u;
  while j > i do
  beginx:= A[i];A[i]:= A[j];A[j]:= x;i:= i+1;j:= j-1
  end;
end;


begin
  A[0]:= -maxint; A[n+1]:= maxint;
  lenW:= 16; hh:= -1;
  top:= 1; stckD[top]:= m; stckH[top]:= n;
  while top > 0 do begin
    dd:= hh+2;
    hh:= stckH[top]; dh:= stckD[top]; top:= top-1;
    hd:= dh+1;
dflen:
  while (hd > dd) and (hh > dh) and (A[dh] > A [hd]) do
  begin
    lenD:= hd-dd; lenH:= hh-dh;
    if lenD < = lenW then begin {left-hand side merging}
      for i:= 1 to lenD do W[i]:= A[i+dd-1];
      d:= dd; h:= hd;
    for i:= 1 to lenD do begin
      x:= W[i];
      while A[h] < x do
        begin A[d]:= A[h]; d:= d+1; h:= h+1
        end;
      A[d]:= x; d:= d+1
    end;
  end
  else if lenH < = lenW then begin {right-hand side
  merging}
    for i:= 1 to lenH do W[i]:= A[i+hd-1];
    d:= hh; h:= dh;
    for i:= lenH downto 1 do begin
      x:= W[i];
      while A[h] > x do
        begin A[d]:= A[h]; d:= d-1; h:= h-1
        end;
    end;
  end
  else begin {decomposition}
    if lenD < = lenH then begin
      s:= (dd+dh) div 2; x:= A[s]; bcorr:= 0;
      ifx < = A[hd]thenbegin dd:= s+1;goto dflen
      end;
      if A[hh] < x then begin
        XCHG(s,dh,hh); dh:= s-1; hh:=
        s+hh-hd; hd:= s;
        goto dflen
      end;
      d:= hd; h:= hh; t:= (d+h) div 2;
      repeat begin {binary search for x in [hd, hh]}
        if x > A[t] then d:= t else h:= t;
        t:= (d+h) div 2
      end
      until t = d;
    end
    else begin
      t:= (hd+hh) div 2; x:= A[t]; bcorr:= 1;
      if x > = A[dh] then begin hh:= t-1; goto
      dflen end;
      if A[dd] > x then begin
        XCHG(dd,dh,t); dd:= dd+t-dh; dh:= t;
        hd:= t+1;
        goto dflen
      end;
      d:= dd; h:= dh; s:= (d+h) div 2;
      repeat begin {binary search for x in [dd, dh]}
        if x > = A[s] then d:= s else h:= s;
        s:= (d+h) div 2
      end;
      until s = d;
      s:= h
    end;
    XCHG(s, dh, t);
    top:= top+1;stckD[top]:= t;stckH[top:= hh;
    dh:= s-1; hh:= s+t-hd-bcorr;
    hd:= s
   end;
  end;
 end;
end;
```