Proceedings of KFIS Autumn Conference 2007 Vol. 17, No. 2.

대칭비교에 의한 Stable Minimum Storage 머징의 복잡도

Complexity of Stable Minimum Storage Merging by Symmetric Comparisons

김복선

서울시 성북구 국민대학교 수학과
E-mail: pskim@kookmin.ac.kr

요 약

Symmerge is a stable minimum storage algorithm for merging that needs $O(m\log\frac{n}{m})$ element comparisons, where m and n are the sizes of the input sequences with $m \leqq n$. According to the lower bound for merging, the algorithm is asymptotically optimal regarding the number of comparisons. The objective of this paper is to consider the relationship between m and n for the spanning case with the recursion level m-1.

Key Words : stable merging, algorithm complexity

1. Introduction

Merging denotes the operation of rearranging the elements of two adjacent sorted sequences of sizes m and n, so that the result forms one sorted sequence of $m+n$ elements. An algorithm merges two adjacent sequences with minimum storage when it needs $O(log^2(m+n))$ bits additional space at most. This form of merging represents a weakened form of in-place merging and allows the usage of a stack that is logarithmically bounded in $m+n$. Minimum storage merging is sometimes also referred to as in situ merging. A merging algorithm is regarded as stable, if it preserves the initial ordering of elements with equal value. Some lower bounds for merging have been proven so far. The lower bound for the number of assignments is $m+n$, because every element may change its position in the sorted result. The lower bound for the number of comparisons is $O(m\log\frac{n}{m})$ for $m \leqq n$. This can be proven by a combinatorial inspection combined with an argumentation using decision trees. An accurate presentation of these bounds is given by Knuth [1].

A minimum storage merging algorithm was proposed by Dudzinski and Dydek [2] in 1981. They presented a divide and conquer algorithm that is asymptotically optimal regarding the number of comparisons but nonlinear regarding the number of assignments. In 2004 Kim and Kutzner [3] also presented a new stable minimum storage merging algorithm performing $O(m\log\frac{n}{m})$ comparisons and $O((m+n)\log m)$ assignments for two sequences of size m and n $m \leqq n$. This algorithm is based on a simple strategy of symmetric comparisons, which will be explained in detail by an example. In complexity-analysis we realized that the algorithm may reach the recursion level $m-1$. The objective of this paper is to consider the relationship between m and n for the spanning case with the recursion level m-1.

2. The Symmerge Algorithm

We start with a brief introduction of our approach to merging. Let us assume that we have to merge the two sequences u=(0,2,5,9) and v=(1,4,7,8). When we compare the input with the sorted result, we can see that in the result the last two elements of u occur on positions belonging to v, and the first two elements of v appear on positions belonging to u (see Fig. 1). So, 2 elements should be exchanged between u and v. The kernel of our algorithm is to compute this number of side-changing elements efficiently and then to exchange such a number of elements. This number can be determined by a process of symmetrical comparisons of elements that happens according to the following principle:

We start at the leftmost element in u and at the rightmost element in v and compare the elements at these positions. We continue doing so by *symmetrically comparing* element-pairs from the outsides to the middle. Fig. 1 shows the resulting pattern of mutual comparisons for our example. There can occur at most one position, where the relation between the compared elements alters from 'not greater' to 'greater'. In Figure 1 two thick lines mark this position. These thick lines determine the number of side-changing elements as well as the bounds for the rotation. Then by recursive application of this technique to the arising subsequences we get a sorted result. Due to this technique of *symmetric comparisons* we call our algorithm Symmerge.

So far we introduced the computation of the number of side-changing elements as linear process of symmetric comparisons. But this computation may also happen in the style of a binary search. Then only $\lfloor \log(\min(|u|, |v|)) \rfloor + 1$ comparisons are necessary to compute the number of side-changing elements.

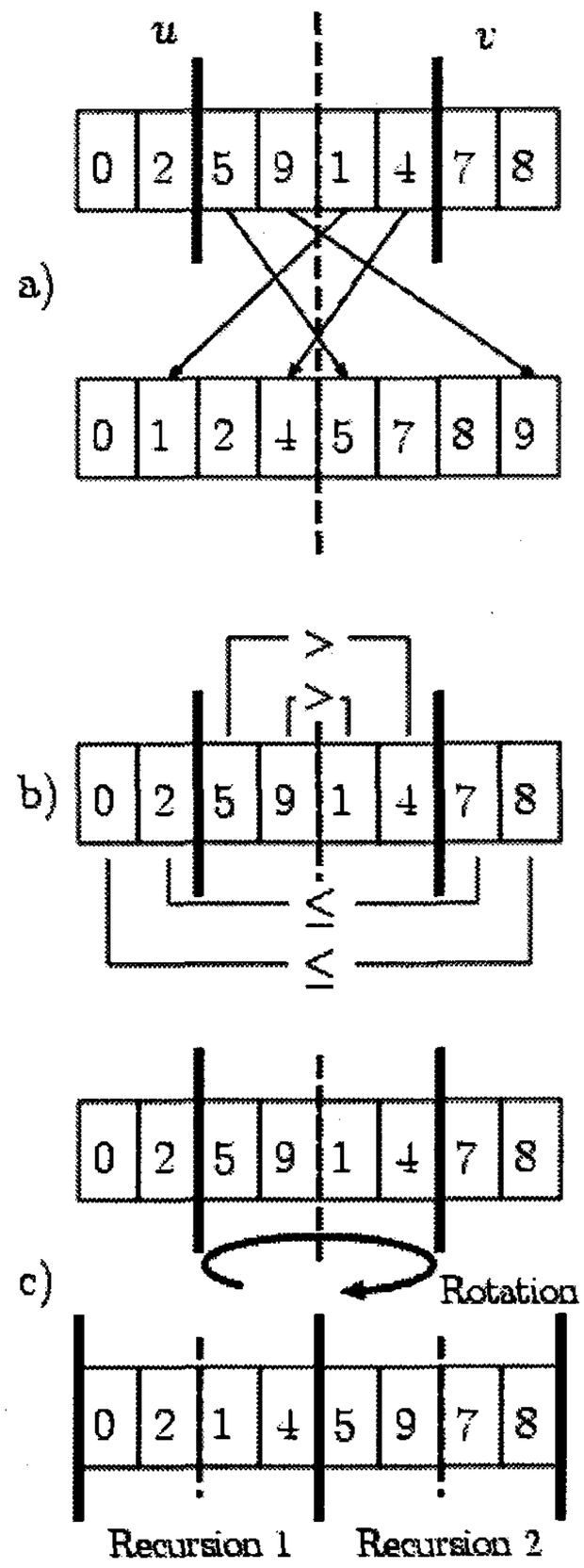

Fig. 1. Symmerge example

2.1 Formal Definition

Let u and v be two adjacent ascending sorted sequences. We define $u \leqq v$ $(u < v)$ iff. $x \leqq y$ $(x < y)$ for all elements $x \in u$ and for all elements $y \in v$. We merge u and v as follows:

If $|u| \leqq |v|$, then

(a1) we decompose v into $v_1 w v_2$ such that $|u| = |u|$ and either $|v_2| = |v_1|$ or $|v_2| = |v_1| + 1$.

(a2) we decompose u into $u_1 u_2$ $(|u_1| \geqq 0, |u_2| \geqq 0)$ and w into $w_1 w_2$ $(|w_1| \geqq 0, |w_2| \geqq 0)$ such that $|u_1| = |w_2|$, $|u_2| = |w_1|$ and $u_1 \leqq w_2$, $u_2 > w_1$.

(a3) we recursively merge u_1 with $v_1 w_1$ as well as u_2 with $w_2 v_2$. Let u' and v' be the resulting sequences, respectively.

else

(b1) we decompose u into $u_1 w u_2$ such that

$|u| = |u|$ and either $|v_2| = |u_1|$ or $|v_2| = |u_1| + 1$.

(b2) we decompose v into $v_1 v_2$ $(|v_1| \geq 0, |v_2| \geq 0)$ and w into $w_1 w_2$ $(|w_1| \geq 0, |w_2| \geq 0)$ such that $|v_1| = |w_2|$, $|v_2| = |w_1|$ and $w_1 \leq v_2$, $w_2 > v_1$.

(b3) we recursively merge $u_1 w_1$ with v_1 as well as $w_2 u_2$ with v_2. Let u' and v' be the resulting sequences, respectively.

$u'v'$ then contains all elements of u and v in sorted order.

Fig. 2. contains an accompanying graphical description of the process described above. The steps (a1) and (b1) manage the situation of input sequences of different length by cutting a subsection w in the middle of the longer sequence as ''active area''. This active area has the same size as the shorter of either input sequences. The decomposition formulated by the steps (a2) and (b2) can be achieved efficiently by applying the principle of the symmetric comparisons between the shorter sequence u (or v) and the active area w. After the decomposition step (a2) (or (b2)), the subsequence $u_2 v_1 w_1$(or $w_2 u_2 v_1$) is rotated so that we get the sub-sequences $u_1 v_1 w_1$ and $u_2 w_2 v_2$($u_1 w_1 v_1$ and $w_2 u_2 v_2$). The treatment of pairs of equal elements as part of the ''outer blocks'' (u_1, w_2 in (a2) and w_1, v_2 in (b2)) avoids the exchange of equal elements and so any reordering of these.

It is obvious that Symmerge is stable because of the decomposition condition $u_1 \leq w_2$, $u_2 > w_1$ (or $w_1 \leq v_2$, $w_2 > v_1$).

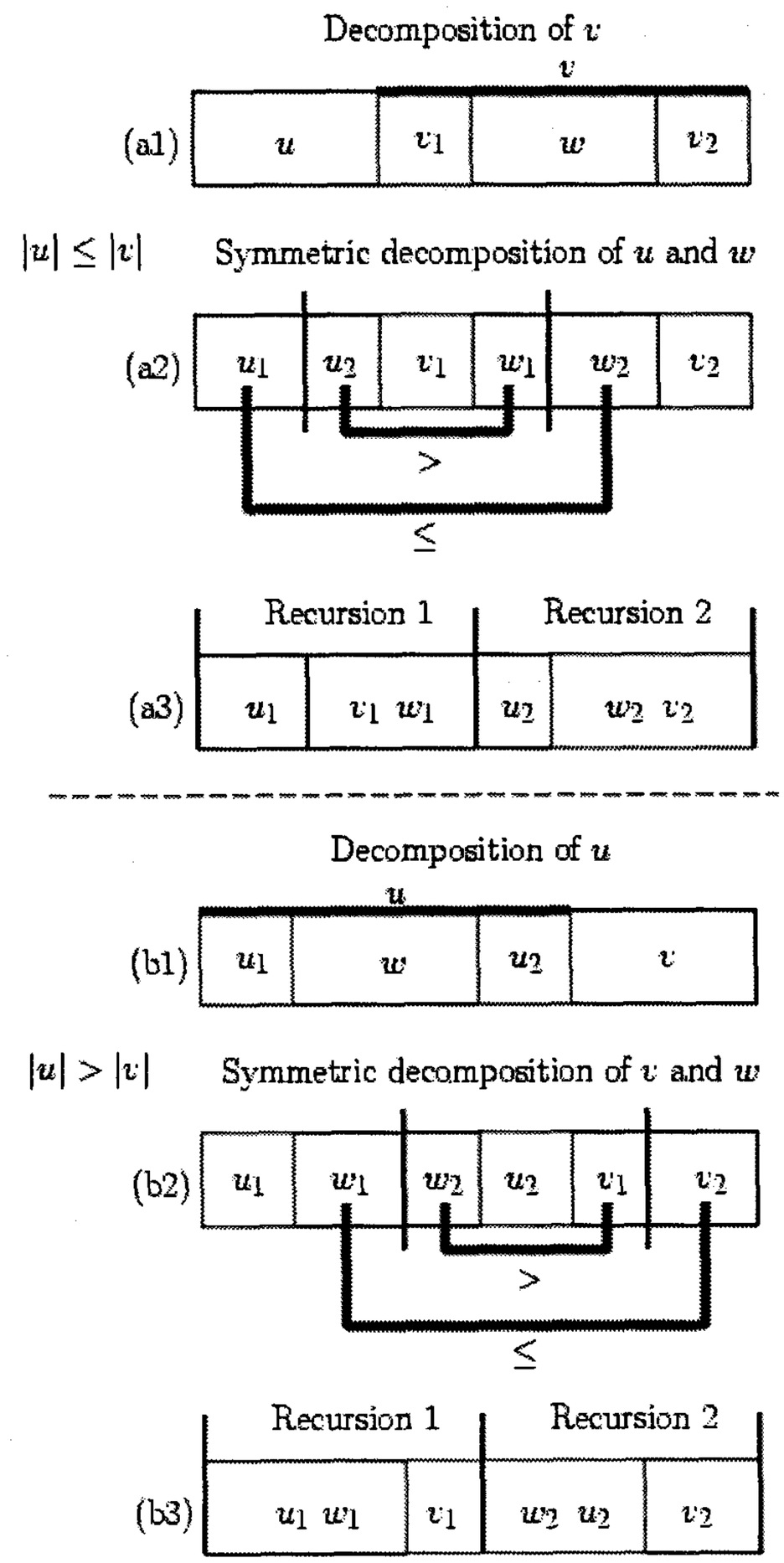

Fig. 2. Illustration of Symmerge

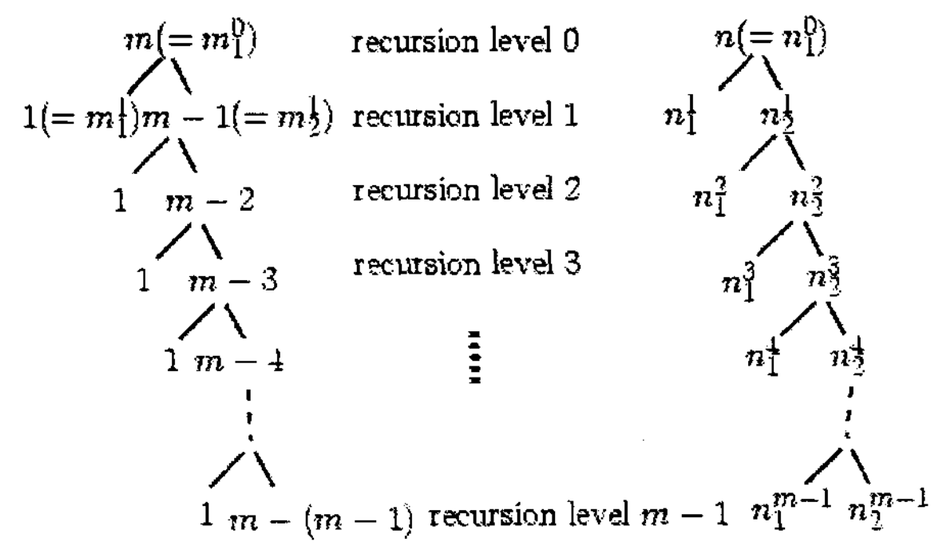

Fig. 3. Spanning case with $m-1$ recursion level

3. Complexity

Unless stated otherwise, let us denote $m = |u|$, $n = |v|$, $m \leq n$. Further let m_j^i and n_j^i denote sizes of sequences merged on the ith recursion level (initially $m_1^0 = m$ and $n_1^0 = n$). The worst case complexity of Symmerge regarding the number of comparisons and assignments are given in [3] as follows:

Theorem 1. The Symmerge algorithm needs $O(m\log\frac{n}{m})$ comparisons.

Theorem 2. If we take the rotation algorithm given in [2], then Symmerge requires $O((m+n)\log m)$ element assignments.

Now we consider the relationship between m and n for the spanning case with the recursion depth m-1, where each (m_2^i, n_2^i) is partitioned to $(1, n_1^{i+1})$ and $(m_2^i - 1(= m_2^{i+1}), n_2^{i+1})$ as shown in Fig 3.

Theorem 3. If Symmerge algorithm reaches the recursion level $m-1$ for two input sequences of sizes m, n $(m \leqq n)$, then $n \geqq 2^{m-1}(m+2) - m$.

Proof. At the beginning the longer sequence u of size n is decomposed into three parts so that the middle part has the same size as the shorter sequence v of size m and then the technique of symmetric comparisons is applied. As result m and n are partitioned to (m_1^1, n_1^1) and (m_2^1, n_2^1) where it holds $m_1^1 = 1$, $m_2^1 = m-1$, $n_1^1 = \frac{n-m}{2} + m - 1$ and $n_2^1 = \frac{n-m}{2} + 1$. To reach the next recursion level, it must be satisfied that $m-1 \leqq n_2^1 = \frac{n-m}{2} + 1$. Suppose that, just as on the first recursion level, $(m-1(= m_2^1), \frac{n-m}{2} + 1(= n_2^1))$ is again partitioned to $(1, n_1^2)$ and $(m-2, n_2^2)$ on the second recursion level. Then

$$m-2 \leqq n_2^2 = \frac{\frac{n-m}{2} + 1 - (m-1)}{2} + 1$$

$$= \frac{n-m+2-2(m-1)+2^2}{2^2}$$

$$= (n - \sum_{i=0}^{1} 2^i(m-i) + \sum_{i=0}^{1} 2^{i+1})/2^2.$$

On the kth recursion level, suppose $(m-(k-1)(= m_2^{k-1}), n_2^{k-1})$ is partitioned to $(1, n_1^k)$ and $(m-k, n_2^k)$, where

$$n_2^k = (n-m+2-2(m-1)+2^2-2^2(m-2) + 2^3 + \cdots - 2^{k-1}(m-(k-1))+2^k)/2^k$$

$$= (n - \sum_{i=0}^{k-1} 2^i(m-i) + \sum_{i=0}^{k-1} 2^{i+1})/2^k.$$

In order to reach the next recursion level $k+1$, it must be satisfied that $m-k \leqq n_2^k$, and so on. Hence, to reach the recursion depth $m-1$, we need the assumption $m-(m-1) \leqq n_2^{m-1}$, where

$$n_2^{m-1} = (n-m+2-2(m-1)+2^2 - 2^2(m-2)+2^3 - \cdots - 2^{i-1}(m-(i-1)) + 2^i - \cdots - 2^{m-2}(m-(m-2))+2^{m-1})/2^{m-1}$$

$$= (n - \sum_{i=0}^{m-2} 2^i(m-i) + \sum_{i=0}^{m-2} 2^{i+1})/2^{m-1}$$

$$= (n-(m-2)(\sum_{i=0}^{m-2} 2^i) + \sum_{i=1}^{m-2} i2^i)/2^{m-1}.$$

Therefore $2^{m-1} \leqq n-(m-2)(2^{m-1}-1) + 2^{m-1}(m-3)+2$

$$= n - 2^{m-1}(m+1)+m)$$

Hence $2^{m-1}(m+2) - m \leqq n$

참 고 문 헌

[1] D. E. Knuth, "The Art of Computer Programming," Addison-Wesley, Vol. 3: Sorting and Searching, 1973.

[2] K. Dudzinski and A. Dydek, "On a Stable Storage Merging Algorithm Information Processing Letters," Vol. 12, No. 1, pp. 5-8, 1981.

[3] P. S. Kim and A. Kutzner, "Stable Minimum Storage Merging by Symmetric Comparisons," In Albers, S., Radzik, T. (eds.), Algorithms-ESA 2004, Springer, Lecture Notes in Computer Science **3221**, pp. 714-723, 2004.