Ratio based stable in-place merging

Pok-Son Kim[1] and Arne Kutzner[2]

[1] Kookmin University, Department of Mathematics, Seoul 136-702, Rep. of Korea
pskim@kookmin.ac.kr
[2] Seokyeong University, Department of Computer Science, Seoul 136-704, Rep. of Korea kutzner@skuniv.ac.kr

Abstract. We investigate the problem of stable in-place merging from a ratio $k = \frac{n}{m}$ based point of view where m, n are the sizes of the input sequences with $m \leq n$. We introduce a novel algorithm for this problem that is asymptotically optimal regarding the number of assignments as well as comparisons. Our algorithm uses knowledge about the ratio of the input sizes to gain optimality and does not stay in the tradition of Mannila and Ukkonen's work [8] in contrast to all other stable in-place merging algorithms proposed so far. It has a simple modular structure and does not demand the additional extraction of a movement imitation buffer as needed by its competitors. For its core components we give concrete implementations in form of Pseudo Code. Using benchmarking we prove that our algorithm performs almost always better than its direct competitor proposed in [6].
As additional sub-result we show that stable in-place merging is a quite simple problem for every ratio $k \geq \sqrt{m}$ by proving that there exists a primitive algorithm that is asymptotically optimal for such ratios.

1 Introduction

Merging denotes the operation of rearranging the elements of two adjacent sorted sequences of size m and n, so that the result forms one sorted sequence of $m+n$ elements. An algorithm merges two sequences *in place* when it relies on a fixed amount of extra space. It is regarded as *stable*, if it preserves the initial ordering of elements with equal value.
There are two significant lower bounds for merging. The lower bound for the number of assignments is $m + n$ because every element of the input sequences can change its position in the sorted output. As shown e.g. in Knuth [7] the lower bound for the number of comparisons is $\Omega(m \log(\frac{n}{m} + 1))$, where $m \leq n$. A merging algorithm is called *asymptotically fully optimal* if it is asymptotically optimal regarding the number of comparisons as well as assignments.
We will inspect the merging problem on the foundation of a ratio based approach. In the following k will always denote the ratio $k = \frac{n}{m}$ of the sizes of the input sequences. The lower bounds for merging can be expressed on the foundation of such a ratio as well. We get $\Omega(m \log(k + 1))$ as lower bound for the number of comparisons and $m \cdot (k + 1)$ as lower bound for the number of assignments.

In the first part of this paper we will show that there is a simple asymptotically fully optimal stable in-place merging algorithm for every ratio $k \geq \sqrt{m}$. Afterward we will introduce a novel stable in-place merging algorithm that is asymptotically fully optimal for any ratio k. The new algorithm has a modular structure and does not rely on the techniques described by Mannila and Ukkonen [8] in contrast to all other works ([10,4,2,6]) known to us. Instead it exploits knowledge about the ratio of the input sizes to achieve optimality. In its core our algorithm consists of two separated operations named "Block rearrangement" and "Local merges". The separation allowed the omitting of the extraction of an additional movement imitation buffer as e.g. necessary in [6]. For core parts of the new algorithm we will give an implementation in Pseudo-Code. Some benchmarks will show that it performs better than its competitor proposed in [6] for a wide range of inputs.

A first conceptual description of a stable asymptotically fully optimal in-place merging algorithm can be found in the work of Symvonis [10]. Further work was done by Geffert et al. [4] and Chen [2] where Chen presented a simplified variant of Geffert et al's algorithm. All three publications delivered neither an implementation in Pseudo-Code nor benchmarks. Recently Kim and Kutzner [6] published a further algorithm together with benchmarks. These benchmarks proved that stable asymptotically fully optimal in-place merging algorithms are competitive and don't have to be viewed as theoretical models merely.

2 A simple asymptotically optimal algorithm for $k \geq \sqrt{m}$

Algorithm	Arguments	Comparisons	Assignments
Hwang and Lin (1) - ext. buffer (2) - m rotat.	u, v with $\|u\| \leq \|v\|$ let $m = \|u\|, n = \|v\|$	$m(t+1) + n/2^t$ where $t = \lfloor \log(n/m) \rfloor$	 $2m + n$ $n + m^2 + m$
Block Swapping	u, v with $\|u\| = \|v\|$	-	$3\|u\|$
Block Rotation	u, v	-	$\|u\| + \|v\| + \gcd(\|u\|, \|v\|)$ $\leq 2(\|u\| + \|v\|)$
Binary Search	u, x (searched element)	$\lfloor \log \|u\| \rfloor + 1$	-
Minimum Search	u	$\|u\| - 1$	-
Insertion Sort	u, let $m = \|u\|$	$\frac{m(m-1)}{2} + (m-1)$	$\frac{m(m+1)}{2} - 1$

Table 1. Complexity of the Toolbox-Algorithms

We now introduce some notations that will be used throughout the paper. Let u and v be two ascending sorted sequences. We define $u \leq v$ ($u < v$) iff $x \leq y$ ($x < y$) for all elements $x \in u$ and for all elements $y \in v$. $|u|$ denotes the size of the sequence u. Unless stated otherwise, m and n ($m \leq n$) are the sizes

of two input sequences u and v respectively. δ always denotes some block-size with $\delta \leq m$.
Tab. 1 contains the complexity regarding comparisons and assignments for six elementary algorithms that we will use throughout this paper. Brief descriptions of these algorithms except for "Minimum Search" can be found in [6]. In the case of "Minimum Search" we assume that u is unsorted, therefore a linear search is necessary.

First we will now show that there is a simple stable merging algorithm called Block-Rotation-Merge that is asymptotically fully optimal for any ratio $k \geq \sqrt{m}$. Afterward we will prove that there is a relation between the number of different elements in the shorter input sequence u and the number of assignments performed by the rotation based variant of Hwang and Lin's algorithm [5].

Algorithm 1: Block-Rotation-Merge (u, v, δ)

1. We split the sequence u into blocks $u_1 u_2 \dots u_{\lceil \frac{m}{\delta} \rceil}$ so that all sections u_2 to $u_{\lceil \frac{m}{\delta} \rceil}$ are of equal size δ and u_1 is of size $m \bmod \delta$. Let x_i be the last element of u_i $(i = 1, \cdots, \lceil \frac{m}{\delta} \rceil)$. Using binary searches we compute a splitting of v into sections $v_1 v_2 \dots v_{\lceil \frac{m}{\delta} \rceil}$ so that $v_i < x_i \leq v_{i+1} (i = 1, \cdots, \lceil \frac{m}{\delta} \rceil - 1)$.
2. $u_1 u_2 \dots u_{\lceil \frac{m}{\delta} \rceil} v_1 v_2 \dots v_{\lceil \frac{m}{\delta} \rceil}$ is reorganized to $u_1 v_1 u_2 v_2 \dots u_{\lceil \frac{m}{\delta} \rceil} v_{\lceil \frac{m}{\delta} \rceil}$ using $\lceil \frac{m}{\delta} \rceil - 1$ many rotations.
3. We locally merge all pairs $u_i v_i$ using $\lceil \frac{m}{\delta} \rceil$ calls of the rotation based variant of Hwang and Lin's algorithm ([5]).

The steps 2 and 3 are interlaced as follows: After creating a new pair $u_i v_i$ $(i = 1, \cdots, \lceil \frac{m}{\delta} \rceil)$ as part of the second step we immediately locally merge this pair as described in step 3.

Lemma 1. Block-Rotation-Merge *performs* $\frac{m^2}{2 \cdot \delta} + 2n + 6m + m \cdot \delta$ *many assignments at most if we use the optimal algorithm from Dudzinski and Dydek [3] for all block-rotations .*

Proof. For the first rotation from $u_1 u_2 \cdots u_{\lceil \frac{m}{\delta} \rceil} v_1$ to $u_1 v_1 u_2 \cdots u_{\lceil \frac{m}{\delta} \rceil}$ the algorithm performs $|u_2| + \cdots + |u_{\lceil \frac{m}{\delta} \rceil}| + |v_1| + \gcd(|u_2| + \cdots + |u_{\lceil \frac{m}{\delta} \rceil}|, |v_1|)$ assignments. The second rotation from $u_2 u_3 \cdots u_{\lceil \frac{m}{\delta} \rceil} v_2$ to $u_2 v_2 u_3 \cdots u_{\lceil \frac{m}{\delta} \rceil}$ requires $|u_3| + \cdots + |u_{\lceil \frac{m}{\delta} \rceil}| + |v_2| + \gcd(|u_3| + \cdots + |u_{\lceil \frac{m}{\delta} \rceil}|, |v_2|)$ assignments, and so on. For the last rotation from $u_{\lceil \frac{m}{\delta} \rceil - 1} u_{\lceil \frac{m}{\delta} \rceil} v_{\lceil \frac{m}{\delta} \rceil - 1} v_{\lceil \frac{m}{\delta} \rceil}$ to $u_{\lceil \frac{m}{\delta} \rceil - 1} v_{\lceil \frac{m}{\delta} \rceil - 1} u_{\lceil \frac{m}{\delta} \rceil} v_{\lceil \frac{m}{\delta} \rceil}$ the algorithm requires $|u_{\lceil \frac{m}{\delta} \rceil}| + |v_{\lceil \frac{m}{\delta} \rceil - 1}| + \gcd(|u_{\lceil \frac{m}{\delta} \rceil}|, |v_{\lceil \frac{m}{\delta} \rceil - 1}|)$ assignments. Additionally $\frac{m}{\delta}(3\delta + 3\delta + \delta^2) = 6m + m \cdot \delta$ assignments are required for the local merges. Altogether the algorithm performs $\delta \cdot ((\frac{m}{\delta} - 1) + (\frac{m}{\delta} - 2) + \cdots + 1) + n + n + 6m + m \cdot \delta = \frac{m^2}{2 \cdot \delta} - \frac{m}{2} + 2 \cdot n + 6m + m \cdot \delta \leq \frac{m^2}{2 \cdot \delta} + 2 \cdot n + 6m + m \cdot \delta$ assignments at most. □

Lemma 2. Block-Rotation-Merge *is asymptotically optimal regarding the number of comparisons.*

Corollary 1. *If we assume a block-size of* $\lfloor\sqrt{m}\rfloor$ *then* Block-Rotation-Merge *is asymptotically fully optimal for all* $k \geq \sqrt{m}$.

So, for $k \geq \sqrt{m}$ there is a quite primitive asymptotically fully optimal stable in-place merging algorithm. In the context of complexity deliberations in the next section we will rely on the following Lemma.

Lemma 3. *Let* λ *be the number of different elements in* u. *Then the number of assignments performed by the rotation based variant of Hwang and Lin's algorithm is* $O(\lambda \cdot m + n) = O((\lambda + k) \cdot m)$.

Proof. Let $u = u_1 u_2 \dots u_\lambda$, where every $u_i (i = 1, \cdots, \lambda)$ is a maximally sized section of equal elements. We split v into sections $v_1 v_2 \dots v_\lambda v_{\lambda+1}$ so that we get $v_i < u_i \leq v_{i+1}$ $(i = 1, \cdots, \lambda)$. (Some v_i can be empty.) We assume that Hwang and Lin's algorithm already merged a couple of section and comes to the first elements of the section $u_i (i = 1, \cdots, \lambda)$. The algorithm now computes the section v_i and moves it in front of u_i using one rotation of the form $\cdots u_i \dots u_\lambda v_i \cdots$ to $\cdots v_i u_i \dots u_\lambda \cdots$. This requires $|u_i| + \cdots + |u_\lambda| + |v_i| + \gcd(|u_i| + \cdots + |u_\lambda|, |v_i|) \leq 2(m + |v_i|)$ many assignments. Afterward the algorithm continues with the second element in u_i. Obviously there is nothing to move at this stage because all elements in u_i are equal and the smaller elements from v were already moved in the step before. Because we have only λ different sections we proved our conjecture. □

Corollary 2. *Hwang and Lin's algorithm is fully asymptotically optimal if we have either* $k \geq m$ *or* $k \geq \lambda$ *where* λ *is the number of different elements in the shorter input sequence* u .

3 Novel asymptotically optimal stable in-place merging algorithm

We will now propose a novel stable in-place merging algorithm called Stable-Optimal-Block-Merge that is fully asymptotically optimal for any ratio. Notable properties of our algorithm are: It does not rely on the block management techniques described in Mannila and Ukonnen's work [8] in contrast to all other such algorithms proposed so far. It degenerates to the simple Block-Rotation-Merge algorithm for roughly $k \geq \sqrt{m}/2$. The internal buffer for local merges and the movement imitation buffer share a common buffer area. The two operations "block rearrangement" and "local merges" stay separated and communicate using a common block distribution storage. There is no lower bound regarding the size of the shorter input sequence.

Algorithm 2: Stable-Optimal-Block-Merge

Step 1: Block distribution storage assignment

Let $\delta = \lfloor\sqrt{m}\rfloor$ be our block-size. We split the input sequence u into $u = s_1 t s_2 u'$ so that s_1 and s_2 are two sequences of size $\lfloor m/\delta \rfloor + \lfloor n/\delta \rfloor$ and t is a sequence of maximal size with elements equal to the last element of s_1. We assume that

there are enough elements to get a nonempty u' and call s_1 together with s_2 our *block distribution storage* (in the following shortened to bd-storage).

Step 2: Buffer extraction

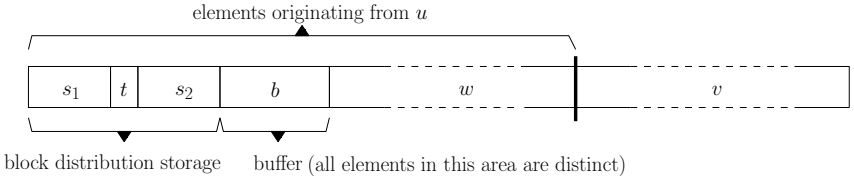

Fig. 1. Segmentation after the buffer extraction

In front of the remaining sequence u' we extract an ascending sorted buffer b of size δ so that all pairs of elements inside b are distinct. Simple techniques how to do so are proposed e.g. in [9] or [6]. Once more we assume that there are enough elements to do so. Now let w be the remaining right part of u' after the buffer extraction.

The segmentation of our input sequences after the buffer extraction is shown in Fig. 1.

Step 3: Block rearrangement

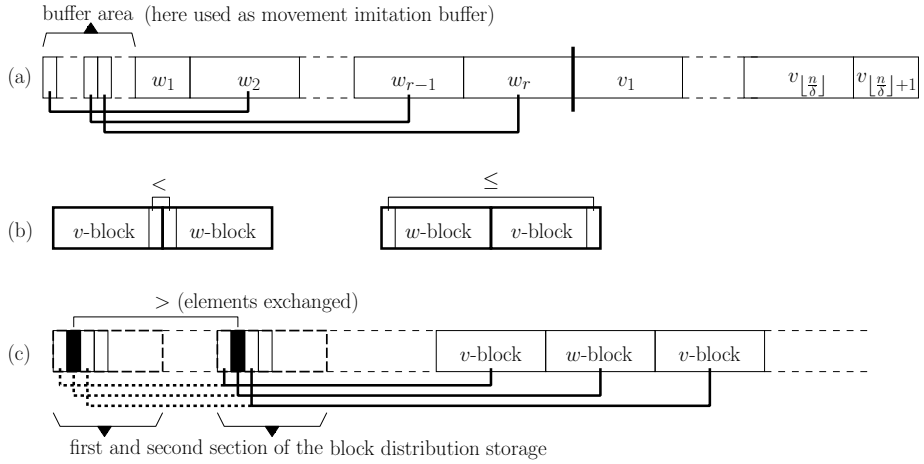

Fig. 2. Graphical remarks to the block rearrangement process

We logically split the sequence wv into blocks of equal size δ as shown in Fig. 2 (a). The two blocks w_1 and $v_{\lfloor \frac{n}{\delta} \rfloor+1}$ are undersized and can even be empty. In the following we call every block originating from w a w-block and every block

originating from v a v-block. The minimal w-block of a sequence of w-blocks is always the w-block with the lowest order (smallest elements) regarding the original order of these blocks.

We rearrange all blocks except of the two undersized blocks w_1 and $v_{\lfloor \frac{n}{\delta} \rfloor + 1}$, so that the following 3 properties hold:
(1) If a v-block is followed by a w-block, then the the last element of the v-block must be smaller than the first element of the w block (Fig. 2(b)).
(2) If a w-block is followed by a v-block, then the first element of the w-block must be smaller or equal to the last element of the v-block (Fig. 2(b)).
(3) The relative order of the v-blocks as well as w-blocks stays unchanged.

This rearrangement can be easily realized by "rolling" the w-blocks through the v-blocks and "drop" minimal w-blocks so that the above properties are fulfilled. During this rolling the w-blocks stay together as group but they can be moved out of order. So, due to the need for stability, we have to track their positions. For this reason we mirror all block replacements in the buffer area using a technique called movement imitation (The technique of movement imitation is described e.g. in [10] and [6]). Every time when a minimal w-block was dropped we can find the position of the next minimal block using this buffer area.
Later we will have to find the positions of w-blocks in the block-sequence created as output of the rearrangement process. For this purpose we store the positions of w-blocks in the block distribution storage as follows:
The block distribution storage consist of two sections of size $\lfloor m/\delta \rfloor + \lfloor n/\delta \rfloor$ and the i-th element of the first section together with the i-th element of the second section belong to the i-th block in the result of the rearrangement process. Please note that, due to the technique used for constructing the bd-storage, such pairs of elements are always different with the first one smaller than the second one. If the i-th block originates from w we exchange the corresponding elements in the bd-storage otherwise we leave them untouched. Fig. 2(c) shows this graphically.

Step 4: Local merges
We visit every w-block and proceed as follows:
Let p be the w-block to be merged and let q be the sequence of all v-originating elements immediately to the right of p that are still unmerged. Further let x be the first element of p.

(1) Using a binary search we split q into $q = q_1 q_2$ so that we get $q_1 < x \leq q_2$. It holds $|q_1| < \delta$ due to the block rearrangement applied before. (2) We rotate pq_1q_2 to q_1pq_2. (3) We locally merge p and q_2 by Hwang and Lin's algorithm, where we use the buffer area as internal buffer.

This visiting process starts with the rightmost w-block and moves sequentially w-block by w-block to the left. The positions of the w-blocks are detected using the information hold in the bd-storage. Every time when we locate the position of a w-block in the bd-storage we bring the corresponding bd-storage elements back to their original order. So, after finishing all local merges both sections of the bd-storage are restored to their original form.

Step 5: Final sweeping up
On the left there is a still unmerged sub-sequence $s_1 t s_2 b w_1 v'$ where v' is the

subsection of v that consists of the remaining unmerged elements. We proceed as follows: (1) We split v' into $v' = v'_1 v'_2$ so that $v'_1 < x \leq v'_2$ where x is the last element of s_2. Afterward we rotate $bw_1 v'_1 v'_2$ to $v'_1 bw_1 v'_2$ and locally merge w_1 and v'_2 using Hwang and Lin's algorithm with the internal buffer. (2) In the same way we split v'_1 into $v' = v'_{1,1} v'_{1,2}$ so that we get $v'_{1,1} < y \leq v'_{1,2}$ where y is the last element of s_1. We rotate $s_1 t s_2 v_{1,1} v_{1,2}$ to $s_1 v_{1,1} t s_2 v_{1,2}$ and locally merge s_1 with $v'_{1,1}$ and s_2 with $v_{1,2}$' using the Block-Rotation-Merge algorithm with a block-size of $\lfloor \sqrt{m} \rfloor$. (3) We sort the buffer area using Insertion-Sort and merge it with all elements right of it using the rotation based variant of Hwang and Lin's algorithm.

Lack of Space in Step 1:
The inputs are so asymmetric that u' becomes empty. Using a binary search we split v into $v = v_1 v_2$ so that we get $v_1 < t \leq v_2$ and rotate $s_1 t s_2 v_1 v_2$ to $s_1 v_1 t s_2 v_2$. Using the Block-Rotation-Merge algorithm with a block-size $\lfloor \sqrt{m} \rfloor$ we locally merge s_1 with v_1 and s_2 with v_2. If s_2 is empty we ignore it and directly merge s_1 with v in the same style.

Extracted buffer smaller than $\lfloor \sqrt{m} \rfloor$ in Step 2:
We assume that we could extract a buffer of size λ with $\lambda < \lfloor \sqrt{m} \rfloor$. We change our block-size δ to $\lfloor |u| / \lambda \rfloor$ and apply the algorithm as described but with the modification that we use the rotation based variant of Hwang and Lin's algorithm for all local merges.

Corollary 3. Stable-Optimal-Block-Merge *is stable.*

Theorem 1. *The* Stable-Optimal-Block-Merge *algorithm requires* $O(m+n) = O(m \cdot (k+1))$ *assignments..*

Proof. It is enough to prove that every step is performed with $O(m+n)$ assignments. In the first step no assignments occur at all. The buffer extraction in step 2 requires $O(m)$ assignments, as shown in [6]. In step 3 the "rolling" of the w-blocks through the v-blocks together with the "dropping" of the minimal w-blocks requires $3\sqrt{m} \cdot (\sqrt{m} + \frac{n}{\sqrt{m}}) = O(m+n)$ assignments. The rotations for the integrated "movement imitation" contribute $O(\sqrt{m} \cdot (\sqrt{m} + \frac{n}{\sqrt{m}})) = O(m+n)$ assignments. The marking of the positions of the w-blocks in the bd-storage needs $O(\sqrt{m})$ assignments. So, altogether step 3 requires $O(m+n)$ assignments. In step 4 each w-block rotation requires $\sqrt{m} + \sqrt{m} + \gcd(\sqrt{m}, \sqrt{m}) = 3\sqrt{m}$ assignments at most. So all w-block rotations need $3\sqrt{m} \cdot \sqrt{m} = O(m)$ assignments. The local mergings using Hwang and Lin's algorithm consume $2m + n$ assignments altogether. The reconstruction of the original order of the exchanged elements in the bd-storage contributes $O(\sqrt{m})$ assignments. In step 5 the first rotation requires $4\sqrt{m}$ assignments at most and the local merging of w_1 and v'_2 needs $3\sqrt{m}$ assignments at most. The second rotation requires $3\sqrt{m} + \frac{n}{\sqrt{m}}$ assignments at most. The success in step 1 implies that roughly $k \leq \sqrt{m}/2$, so we get $k \cdot \sqrt{m} \leq m$. Further we have $\lfloor m/\delta \rfloor + \lfloor n/\delta \rfloor$ is roughly equal to $(k+1) \cdot \sqrt{m} = \frac{m+n}{\sqrt{m}}$. So, according to Lemma 1 each local merging with Block-Rotation-Merge needs $\frac{(k\sqrt{m}) \cdot k\sqrt{m}}{2 \cdot \sqrt{m}} + 2 \cdot n + 6k\sqrt{m} + k(\sqrt{m}\sqrt{m}) \leq \frac{k \cdot m}{2} + 2n + 6m + k \cdot m$ assignments at

most. The buffer sorting using insertion sort contributes $O(m)$ assignments and the final call of Hwang and Lin's algorithm requires $n + m + \sqrt{m}$ assignments. So, step 5 needs altogether $O(m + n)$ assignments as well.
In the first exceptional case "Lack of Space in Step 1" we have roughly $k \geq \sqrt{m}/2$ and directly switch to Block-Rotation-Merge. According to Corollary 1 Block-Rotation-Merge is fully asymptotically optimal for such k.
In the second exceptional case "Extracted buffer smaller than $\lfloor\sqrt{m}\rfloor$" we change the block-size to $\lfloor |u| / \lambda \rfloor$ with $\lambda < \sqrt{m}$ and use the rotation based variant of Hwang and Lin's algorithm for local merges. A recalculation of the steps 3 to 5, were we use Lemma 3 in the context of all local merges, proves that the number of assignments is still $O(m + n)$. □

Lemma 4. *If $k = \sum_{i=1}^{n} k_i$ for any $k_i > 0$ and integer $n > 0$, then $\sum_{i=1}^{n} \log k_i \leq n \log(k/n)$.*

Proof. It holds because the function $\log x$ is concave. □

Theorem 2. *The* Stable-Optimal-Block-Merge *algorithm requires $O(m \log(\frac{n}{m} + 1)) = O(m \log(k + 1))$ comparisons.*

Proof. As in the case of the assignments it is enough to show that every step keeps the asymptotic optimality. Step 1 contains one binary search over m merely. The buffer extraction in step 2 requires m comparisons at most, as shown in [6]. The rearrangement of all blocks except of the two undersized blocks w_1 and $v_{\lfloor \frac{n}{\delta} \rfloor + 1}$ in step 3 requires $2\sqrt{m} + \frac{n}{\sqrt{m}}$ comparisons at most. The detection of the minimal element in the movement imitation buffer demands $\sqrt{m} \cdot \sqrt{m}$ many comparisons at most. In step 4 the binary searches for splitting the q-sequences cost $\sqrt{m} \cdot \log \sqrt{m}$ comparisons at most. Now let $(m_1, n_1), (m_2, n_2), \cdots, (m_r, n_r)$ be the sizes of all r-groups that are locally merged by Hwang and Lin's algorithm. According to Lemma 4, Table 1 and since $r < \sqrt{m}$ this task requires $\sum_{i=1}^{r}(m_i(\log(\frac{n_i}{m_i}) + 1) + m_i) = \sum_{i=1}^{r}(m_i \log(\frac{n_i}{m_i}) + 2m_i) \leq \sum_{i=1}^{r} m_i \log(\frac{n_i}{m_i}) + 2m = \sum_{i=1}^{r}(m_i \log n_i - m_i \log m_i) + 2m = \sqrt{m} \sum_{i=1}^{r}(\log n_i - \log m_i) + 2m \leq \sqrt{m}(\sqrt{m} \log \frac{n}{r} - \sqrt{m} \log \frac{m}{r}) + 2m \leq m(\log(\frac{n}{m} + 1)) + 2m = O(m \log(\frac{n}{m} + 1))$ comparisons. The asymptotic optimality in step 5 as well as in the exceptional case "Lack of Space in Step 1" is obvious due to Lemma 2. The change of the block-size in the second exceptional case "Extracted buffer smaller than $\lfloor\sqrt{m}\rfloor$" triggers a simple recalculation of step 3 and step 4, where we leave the details to the reader. □

Corollary 4. Stable-Optimal-Block-Merge *is an asymptotically fully optimal stable in-place merging algorithm.*

Pseudo-code implementations for the core operations "block rearrangement" and "local merges" are given in Alg. 1 and Alg. 2, respectively. Both code segments contain calls of the toolbox algorithms mentioned in section 2. The Pseudo-code definitions for these toolbox algorithms are summarized in Tab. 2.

Pseudo-code Definition	Description of the Arguments
HWANG-AND-LIN($A, first1, first2, last$)	u is in $A[first1 : first2 - 1]$, v is in $A[first2 : last - 1]$
BINARY-SEARCH($A, first, last, x$)	delivers the position of the **first** occurrence of x in $A[first : last-1]$
MINIMUM($A, pos1, pos2$)	delivers the index of the minimal element in $A[pos1 : pos2 - 1]$
BLOCK-SWAP($A, pos1, pos2, len$)	u is in $A[pos1 : pos1 + len - 1]$, v is in $A[pos2 : pos2 + len - 1]$
BLOCK-ROTATE($A, first1, first2, last$)	u, v as in HWANG-AND-LIN

EXCHANGE($A, pos1, pos2$) is equal to BLOCK-SWAP($A, pos1, pos2, 1$).

Table 2. Pseudo-code Definitions of the Toolbox Algorithms

Algorithm 1 Pseudo-code of the procedure for the block rearrangement

REARRANGE-BLOCKS(A, $first1$, $first2$, $last$, buf, $bds1$, $bds2$, $blockSize$)

```
1  ▷ w_2 ... w_x is in A[first1 : first2 − 1], v_1 ... v_{y−1} is in A[first2 : last − 1]
2  ▷ buffer b is in A[buf : buf + ⌊√m⌋ − 1]
3  ▷ bd-storage s_{1|2} is in A[bds{1|2} : bds{1|2} + ⌊√m⌋ + ⌊n/√m⌋ − 1]
4
5  bufEnd ← buf + (first2 − first1) / blockSize
6  minBlock ← first1
7  while first1 < first2
8        do if first2 + blockSize < last and A[first2 + blockSize − 1] < A[minBlock]
9              then BLOCK-SWAP(A, first1, first2, blockSize)
10                  BLOCK-ROTATION(A, buf, buf + 1, bufEnd)
11                  if minBlock = first1
12                     then minBlock ← first2
13                  first2 ← first2 + blockSize
14             else BLOCK-SWAP(A, minBlock, first1, blockSize)
15                  EXCHANGE(A, buf, buf + (minBlock − first1) / blockSize)
16                  EXCHANGE(A, bds1, bds2)
17                  buf ← buf + 1
18                  if buf < end
19                     then minIndex ← MINIMUM(A, buf, bufEnd)
20                          minBlock ← first1 + (minIndex − buf) ∗ blockSize
21           bds1 ← bds1 + 1; bds2 ← bds2 + 1
22           first1 = first1 + blockSize
```

3.1 Optimizations

We now report about several optimizations that help improving the performance of the algorithm without any impact on its asymptotic properties. The immediate mirroring of all w-block movements in the movement imitation buffer (occurs in Step 3) triggers a rotation (line 10 in Alg. 1) every time when a v-block is moved

Algorithm 2 Pseudo-code of the function for local merges

```
LOCAL-MERGES(A, first, last, buf, bds1, bds2, blockSize, numBlocks)
 1  ▷ A[first : last − 1] contains all blocks in distributed form
 2
 3  index ← ((last − first) / blockSize) − 1
 4  while numBlocks > 0
 5        do while A[bsd1 + index] < A[bsd2 + index]
 6                 do index ← index − 1
 7           first2 ← first + ((index + 1) ∗ blockSize)
 8           if first2 < last
 9              then b ← BINARY-SEARCH(first2, last, A[first2 − blockSize])
10                   BLOCK-ROTATION(A, first2 − blockSize, first2, b)
11                   HWANG-LIN(A, b − blockSize, b, last, buf)
12                   last ← b − blockSize
13           EXCHANGE(A, bds1 + index, bds2 + index)
14           numBlocks ← numBlocks − 1; index ← index − 1
15  return last
```

into front of the group of w-blocks. The number of necessary rotations can be reduced by first counting the number of v-blocks moved into front of the w-blocks. This counting follows a single update of the movement imitation buffer if the placement of a minimal w-block happens. In the context of the movement of v-blocks into front of w-blocks (Step 3) the floating hole technique (for a description see [4] or [6]) can be applied for reducing the number of assignments. Similarly the floating hole technique can also be applied during the local merges (Step 4) by combining the block swap to the internal buffer with the rotation that moves smaller v-originating elements to the front of the w-block. In the special case "Extracted buffer smaller then $\lfloor\sqrt{m}\rfloor$" the sorting of the buffer b in Step 5 is unnecessary because the buffer is already sorted after Step 3 and stays unchanged during Step 4. Insertion-Sort can be replace by some more efficient sorting algorithm. Please note that there is no need for stability in the context of the buffer sorting because all buffer elements are distinct.

4 Experimental work

We did some experimental work with our algorithm in order to get an impression of its performance. We compared it with the stable fully asymptotically optimal algorithm presented in [6] as well as the simple standard algorithm that relies on external space of size m. The results of our experimental work summarizes Tab. 3 where every line shows average values for 50 runs with different data. We took a standard desktop computer with 2GHz processor as hardware platform. All coding happened in the C programming language. For the measurement of the number of assignments we applied the optimal block rotation algorithm

n	m	Stable-O.-B.-Merge			SOFSEM 2006 Alg.			Linear Standard Alg.		
		$\#comp$	$\#assign$	t_e	$\#comp$	$\#assign$	t_e	$\#comp$	$\#assign$	t_e
2^{21}	2^{21}	5843212	37551852	227	5961524	49666369	335	4194239	8388608	121
2^{21}	2^{18}	1500433	15866835	100	1505766	17182008	122	2359288	4718592	71
2^{21}	2^{15}	280611	17350896	87	280412	12681115	68	2129890	4259840	64
2^{21}	2^{12}	43611	4422493	35	47330	10512479	53	2100804	4202496	63
2^{23}	2^{9}	8057	16350956	133	8589	38150052	202	8373039	16778240	251
2^{23}	2^{6}	1200	15459824	131	1271	30749720	161	8234508	16777344	254
2^{23}	2^{3}	172	11322991	119	170	7535160	68	7572307	16777232	301
2^{23}	2^{0}	23	4163489	55	24	4163489	55	4225121	16777218	261

t_e : Execution time in ms, $\#comp$: Number of comp., m, n : Lengths of inp. seq.

Table 3. Practical comparison of various merge algorithms

presented in [3]. Although this algorithm is optimal regarding the number of assignments it is quite slow in practice due to its high computational demands. Therefore for the time measurements we applied a block-swap based algorithm presented e.g. in [1] using identical data.

Regarding the buffer extraction (Step 2) there are several alternatives. The extraction process can be started from the left end as well as from the right end of the input and we can choose between a binary search and linear search for the determination of the next element. All 4 possible combinations keep the asymptotic optimality. However, there is no clear "best choice" among them because the most advantageous combination can vary depending on the structure of the input. In the context of the Stable-Optimal-Block-Merge algorithm we decided for the variant "starting from the left combined with binary search", the SOFSEM 2006 algorithm already originally chose "starting from the right combined with linear search".

Except for two combinations of input sizes our new algorithm is always faster than its predecessor. The bad performance in the case $(2^{21}, 2^{15})$ reflects the lack of the implementation of the floating hole technique as mentioned in the section about optimizations. The application of Block-Rotation-Merge triggers unnecessary rotations in the case $(2^{23}, 2^3)$. This can be fixed by introduction of a check whether $k \geq m$ and a direct switch to the rotation based variant of Hwang and Lin's algorithm if true.

5 Conclusion

We investigated the problem of stable in-place merging from a ratio based point of view by introducing a ratio $k = \frac{n}{m}$, where m,n are the sizes of the input sequences with $m \leq n$. We could show that there is a simple asymptotically fully optimal (optimal regarding the number of comparisons as well as assignments) stable in-place merging algorithm for any ratio $k \geq \sqrt{m}$.

In the second part of this paper we introduced a novel asymptotically fully optimal stable in-place merging algorithm which is constructed on the foundation of deliberations regarding the ratio of the input sizes. Highlights of this

algorithm are: It has a modular structure and does not rely on techniques described by Mannila and Ukkonen [8] in contrast to all its known competitors ([10,4,6]). The tasks "block-distribution" and "local block mergings" are modular separated. As side effect they can share a common buffer area and the extraction of a separated movement imitation buffer is not necessary. The algorithm demands no lower bound for the size of the shorter input sequence (32 elements in case of the alg. in [4] and 10 elements for the alg. in [6]).
Our algorithm performs for a wide range of inputs remarkably better than its direct competitor presented in [6]. There is a superiority in particular for symmetrically sized inputs, a fact that is of importance in the context of the Mergesort algorithm.
The number of comparisons and assignments are good measurements for the efficiency of merging algorithms. However, the impact of other operations as e.g. numerical calculations and index comparisons deserves investigation as well. As motivation we would like to refer to a well known effect with the optimal block-rotation algorithm introduced by Dudzinski and Dydek in [3]. Their algorithm is optimal regarding the number of assignments but has a bad performance due to a included computation of a greatest common divisor. For our further work we plan to include deliberations regarding such so far uncounted operations.

References

1. J. Bentley. *Programming Pearls*. Addison-Wesley, Inc, 2nd edition, 2000.
2. J. Chen. Optimizing stable in-place merging. *Theoretical Computer Science*, 302(1/3):191–210, 2003.
3. K. Dudzinski and A. Dydek. On a stable storage merging algorithm. *Information Processing Letters*, 12(1):5–8, February 1981.
4. V. Geffert, J. Katajainen, and T. Pasanen. Asymptotically efficient in-place merging. *Theoretical Computer Science*, 237(1/2):159–181, 2000.
5. F.K. Hwang and S.Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM J. Comput.*, 1(1):31–39, 1972.
6. Pok-Son Kim and Arne Kutzner. On optimal and efficient in place merging. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM 2006*, volume 3831 of *Lecture Notes in Computer Science*, pages 350–359. Springer, 2006.
7. D. E. Knuth. *The Art of Computer Programming*, volume Vol. 3: Sorting and Searching. Addison-Wesley, 1973.
8. H. Mannila and Esko Ukkonen. A simple linear-time algorithm for in situ merging. *Information Processing Letters*, 18:203–208, 1984.
9. L. T. Pardo. Stable sorting and merging with optimal space and time bounds. *SIAM Journal on Computing*, 6(2):351–372, June 1977.
10. A. Symvonis. Optimal stable merging. *Computer Journal*, 38:681–690, 1995.