Sorting Algorithms for Broadcast Communications: Algorithms and Fundamental Analysis*

Shyue-Horng Shiau Chang-Biau Yang†
Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan 804, R.O.C.
shiaush@cse.nsysu.edu.tw cbyang@cse.nsysu.edu.tw

June 1, 2000

Abstract

In this paper, we shall propose two algorithms to solve the sorting problem under the broadcast communication model(BCM). The two sorting algorithms are based on the maximum finding and loser selection algorithms. The main ideas and structures of the two algorithms are same. We use successful broadcasts to build broadcasting layers logically and then to distribute the data elements into those logic layers properly. Thus, few data elements are contained on each layer and broadcast conflicts will be reduced. Suppose that there are n input data elements and n processors under BCM are available. Not only we show that both the average time complexities of the two algorithms are $\Theta(n)$, and the algorithm based on the loser selection gets better performance than the other, but also show that an intuition is wrong. The intuition is that a fast maximum finding algorithm will be a better base for a sorting algorithm under BCM. In this paper, we focus on the precise description of the two algorithms and their fundamental analysis. The mathematical analysis of the two sorting algorithms can be found in the companion paper[5].

Key words: parallel algorithm, sorting, selection, broadcast communication, conflict

1 Introduction

One of the simplest parallel computation models is the *broadcast communication model* (BCM)[1, 2, 6–9, 11, 15, 16, 19–22]. This model consists of some processors sharing one

*This research work was partially supported by the National Science Council of the Republic of China under NSC89-2213-E110-005.

†To whom all correspondence should be sent.

common channel for communications. Each processor in this model can communicate with others only through this shared channel. Whenever a processor broadcasts messages, any other processor can hear the broadcast messages via the shared channel. If more than one processor wants to broadcast messages simultaneously, a *broadcast conflict* occurs. When a conflict occurs, a conflict resolution scheme should be invoked to resolve the conflict. This resolution scheme will enable one of the broadcasting processors to broadcast successfully. The Ethernet, one of the famous local area networks, is an implementation of such a model.

The time required for an algorithm to solve a problem under BCM includes three parts: (1) resolution time: spent to resolve conflicts, (2) transmission time: spent to transmit data, (3) computation time: spent to solve the problem. For the sorting problem, it does not seem that transmission time and computation time can be reduced. Therefore, to minimize resolution time is the key issue to improve the time complexity.

The probability concept, proposed by Martel [9], is to estimate a proper broadcasting probability, which can reduce conflict resolution time. Another idea to reduce conflict resolution time is the layer concept proposed by Yang [20]. Applying the layer concept for finding the maximum among a set of n numbers [15], we can improve the time complexity from $\Theta(\log^2 n)$ to $\Theta(\log n)$ [16].

Some researchers [2, 6, 7, 10, 13, 22] have solved the sorting algorithm under BCM. The selection sort can be simulated as a straightforward method for sorting under BCM. The method is to have all processors broadcast their elements and to find the maximum element repeatedly. In other words, the maximum elements which are found sequentially form the sorted sequence. Levitan and Foster [7, 8] proposed a maximum finding algorithm, which is nondeterministic and requires $O(\log n)$ successful broadcasts in average. Martel [9], and Shiau and Yang [15] also proposed maximum finding algorithms, which require $O(\log n)$ and $\Theta(\log n)$ time slots (including resolution time slots) in average respectively. Thus, the time required for the straightforward sorting method based on the repeated maximum finding[15] is $\Theta(n \log n)$. In this paper, we can improve it from $\Theta(n \log n)$ to $\Theta(n)$ by applying the layer concept.

Martel *et al.* [10] showed the sorting problem can be solved in $O(n)$. They used a modified version of selection sort, i.e., sort by repeatedly finding maximum based on the results from previous iterations. It can be viewed as a modified Quicksort. Until now, Martel's algorithm is the fastest one for solving the sorting problem under BCM. However, the constants associated with $O(n)$ bound proved by Martel *et al.* [10] are not very tight.

In this paper, we shall apply the layer concept and propose two sorting algorithms under BCM. And we shall give the time complexity analysis of our two algorithms. Our analysis focuses on the average number of time slots including conflict slots, empty slots, and slots for successful broadcast. Here, it is assumed that each time slot requires constant time.

In the first sorting algorithm, we make use of the maximum finding method [15] to build layers and reduce conflicts. After the process of maximum finding terminates, we can use these successful broadcasts to build broadcasting layers logically and then to distribute the data elements into those logic layers properly. There are only few processors on the layers. Thus, conflict resolution time can be reduced. The average number of time

slots required for sorting n elements is T_n, where $\frac{7}{2}n - \frac{1}{2} \leq T_n \leq \frac{23}{6}n - \frac{7}{6}$.

In the second sorting algorithm, we make use of the loser selection method [12] to resolve conflicts (also see [14]). The method is to have each processor which gets a head flips a coin repeatedly. All which get heads continue to broadcast in the next time slot, until exactly one processor gets head and broadcasts its value successfully.

In the layer concept, when there is a successful broadcast, each processor whose element is greater than the successful one brings itself up to the upper layer. Therefore those processors are divided into two layers. After several successful broadcasts occur, a few logic layers are built, and the processors will be separated into those layers. As the first algorithm, there are only few processors on the layers, and conflict resolution time will be reduced. The average number of time slots is T_n, where $\frac{8}{3}n - S_n + \frac{2}{3} \leq T_n \leq \frac{10}{3}n - S_n - \frac{2}{3}$, and $S_n = O(\log n)$.

The mathematical analysis of our two sorting algorithms can be found in the companion paper[5]. In this paper, we focus on the precise description of our two algorithms and their fundamental analysis.

We show that the two bounds of time complexities of our algorithms are very tight. Comparing the two bounds, we not only show that the second algorithm is better than the first algorithm, but also show that an intuition is wrong. The intuition is that a fast maximum finding algorithm will be a better base for a sorting algorithm under BCM. In fact, the sorting algorithm based on loser selection has a better performance, although it takes a longer time to find the first maximum element ($\Theta(\log^2 n)$[16]). In more detail, to find the first maximum element, our first sorting algorithm takes $\Theta(\log n)$[15], and our second sorting algorithm takes $\Theta(\log^2 n)$[16]. But somehow the second sorting algorithm will speed up and rise above the first sorting algorithm after the first k elements are sorted among the n elements. One of our future works is to find out when it will happen.

This paper is organized as follows. In Section 2, we shall present our first sorting algorithm based on the maximum finding[15]. This section includes three subsections. In Subsection 2.1, we shall review the previous results for the maximum finding algorithm. based on the maximum finding algorithm, we shall present our first sorting algorithm and its analysis in Subsections 2.2 and 2.3. In Section 3, we shall present our second sorting algorithm based on the loser selection[12]. The structure of this section is similar to Section 2. Finally, Section 4 concludes the paper.

2 The first sorting algorithm based on maximum finding

In this section, we shall first briefly review the maximum finding algorithm under BCM, which is proposed and analyzed by Shiau and Yang [15]. Grabner and Prodinger[4] gave a precise asymptotic analysis. Based on the maximum finding, we present our first sorting algorithm in Subsection 2.2 and its analysis in Subsection 2.3.

We shall use the notation $\langle x_0, x_1, x_2, \cdots, x_t \rangle$ to denote a linked list, where x_t is the head of the list. Suppose $L_1 = \langle x_0, x_1, x_2, \cdots, x_t \rangle$ and $L_2 = \langle y_0, y_1, y_2, \cdots, y_{t'} \rangle$, then $\langle L_1, L_2 \rangle$ represents the concatenation of L_1 and L_2, which is $\langle x_0, x_1, x_2, \cdots, x_t, y_0, y_1, y_2, \cdots, y_{t'} \rangle$.

For shortening the description in the algorithms, we use the term "data element" to represent "the processor storing the data element" and to represent the data element itself at different times if there is no ambiguity.

2.1 The previous results for maximum finding

In the maximum finding algorithm, broadcast conflicts can be used to build broadcasting layers and then to distribute the data elements into those layers. When a broadcast conflict occurs, a new upper layer is built up. Each data element which joins the conflict flips a coin with equal probabilities. That is, the probability of getting a head is $\frac{1}{2}$. All which get heads continue to broadcast in the next time slot, and bring themselves up to the new upper layer. This new layer becomes the active layer. The others which get tails abandon broadcasting, and still stay on the current layer. At any time, only the processors which are alive and on the active layer may broadcast.

The maximum finding algorithm can be represented as a recursive function: $Maxfind(C, L_m)$, where C is a set of data elements (processors). Initially, each processor holds one alive data element and sets an initial linked list $L_m = \langle x_0 \rangle$, where $x_0 = -\infty$. We rewrite the algorithm as follows.

Algorithm Maximum-Finding: $Maxfind(C, L_m)$

Step 1: Each alive data element in C broadcasts its value. Note that a data element is alive if its value is greater than the head of L_m. Each element smaller than or equal to the head drops out (becomes dead).

Step 2: There are three possible cases:

Case 2a If a broadcast conflict occurs, then do the following.

Step 2a.1: Each alive data element in C flips a coin. All which get heads form C' and bring themselves to the upper layer, and the others form $C"$ and stay on the current layer.

Step 2a.2: Perform $L_m = Maxfind(C', L_m)$.

Step 2a.3: Perform $L_m = Maxfind(C", L_m)$.

Step 2a.4: Return L_m.

Case 2b: If exactly one data element successfully broadcasts its value y, then return $\langle L_m, y \rangle$.

Case 2c: If no data element broadcasts, then return L_m. (C is empty or all data elements of C are dead.)

In the above algorithm, we can get a sequence of successful broadcasts, which is represented by the linked list $L_m = \langle x_0, x_1, x_2, \cdots, x_{t-1}, x_t \rangle$, where $x_{i-1} < x_i$, $1 \leq i \leq t$ and $x_0 = -\infty$. Note that L_m is held by each processor and the head x_t is the maximum.

Shiau and Yang [15] proved that the total number of time slots is M_n in average, where $4 \ln n - 4 < M_n < 5 \ln n + \frac{5}{2}$. Grabner and Prodinger[4] gave a precise asymptotic formula that $M_n \sim (\frac{\pi^2}{3 \ln 2}) \ln n \sim (4.74627644\ldots) \ln n$.

Here we use the lemma

$$\frac{4}{n} \leq M_n - M_{n-1} \leq \frac{5}{n}, \tag{1}$$

which was proved by Shiau and Yang[15], rather than the precise asymptotic formula $M_n \sim (\frac{\pi^2}{3 \ln 2}) \ln n \sim (4.74627644\ldots) \ln n$. This is done because it is easy to apply our result into the proof directly by fundamental mathematics.

2.2 The sorting algorithm based on maximum finding

Our first sorting algorithm is based on our maximum finding. In the sorting algorithm, the key point is to use successful broadcasts which occur within the progress of maximum finding to build broadcasting layers and then to distribute the data elements into those layers properly.

When a series of successful broadcasts occur within the progress of maximum finding, some correspondent layers are built up. All processors which participated in the maximum finding are divided into those layers. Each layer performs the sorting algorithm until the layer is empty or contains exactly one data element.

Our first sorting algorithm can be represented as a recursive function: $Sorting(M)$, where M is a set of data elements (processors). In the algorithm, each processor holds one data element and maintains a linked list L_s. Our first sorting algorithm is as follows:

Algorithm Sorting: $Sorting(M)$

Step 1: Each data element in M broadcasts its value.

Step 2: There are three possible cases:

Case 2a: If exactly one data element successfully broadcasts its value y, then return $\langle y \rangle$.

Case 2b: If no data element broadcasts, then return $NULL$.

Case 2c: If a broadcast conflict occurs, then perform
$L_m = Maxfind(M, \langle x_0 \rangle)$, where $L_m = \langle x_0, x_1, x_2, \cdots, x_{t-1}, x_t \rangle$ is a series of successful broadcasts and $x_{i-1} < x_i$, $1 \leq i \leq t$ and $x_0 = -\infty$.

Step 2c.1: Set $L_s = NULL$.

Step 2c.2: For $i = t$ down to 1

Set $L_s = \langle x_i, L_s \rangle$.

do $L_s = \langle Sorting(M_i), L_s \rangle$,

where $M_i = \{x | x_{i-1} < x < x_i\}$.

Step 2c.3: Return L_s.

In the above algorithm, x_1 is the element of the first successful broadcast, and x_t is the element of the last successful broadcast and also the largest element in M.

For example, suppose that M has 2 data elements $\{1, 2\}$. We can get a series of successful broadcasts L_m after $Maxfind(M, \langle x_0 \rangle)$ is performed. The first successful

broadcast may be one of the two elements randomly, each case having probability $\frac{1}{2}$. If the first successful broadcast is 2, then $L_m = \langle x_0, 2\rangle$, and the final sorted linked list is $\langle x_0, Sorting(M_1), 2\rangle$, where $M_1 = \{x|x_0 < x < x_1\} = \{x| -\infty < x < 2\} = \{1\}$. If the first successful broadcast is 1, then $L_m = \langle x_0, 1, 2\rangle$, and the final sorted linked list is $\langle x_0, Sorting(M_1), 1, Sorting(M_2), 2\rangle$, where $M_1 = \{x|x_0 < x < x_1\} = \{x| -\infty < x < 1\} = \emptyset$ and $M_2 = \{x|x_1 < x < x_2\} = \{x|1 < x < 2\} = \emptyset$.

Martel *et al.* [10] showed the sorting problem can be solved in $O(n)$. However, the constant bounds associated with $O(n)$ proved by Martel *et al.* [10] are not very tight.

On the high level view, our first sorting algorithm has a similar approach as that proposed by Martel *et al.* [10]. On the low level view, our first algorithm is based on the layer concept, thus it can be analyzed easily and a tighter bound can be obtained. And the analysis can be applied to sorting algorithms based on another maximum finding algorithm.

2.3 Analysis of the sorting algorithm based on maximum finding

We shall prove that the average time complexity of our first sorting algorithm is $\Theta(n)$, where n is the number of input data elements. Let T_n denote the average number of time slots. When there is zero or one input data element for the algorithm, one empty slot or one slot for successful broadcast is needed. Thus $T_0 = 1$ and $T_1 = 1$.

If $n = 2$, we have the following recursive formula:

$$T_2 = M_2 + R_2, \text{ where } R_2 = \left\{ \begin{array}{rrr} & \frac{1}{2}[& T_1] \\ + & \frac{1}{2}[T_0 & +T_0] \end{array} \right\}. \tag{2}$$

We shall explain the above two equations. In the first equation, the first term M_2 is the average number of time slots required for the maximum finding algorithm while $n = 2$. The second term R_2 is the average number of time slots required for finishing the sorting after the maximum finding ends.

In the equation for R_2, the first successful broadcast may be either one of the 2 elements randomly, each case having probability $\frac{1}{2}$. The subterm of first term, T_1, arises when the first successful broadcast is the largest element. Thus, the layer between $x_0 = -\infty$ and the largest element contains one element, which is the second largest element, and T_1 time slots are required.

In second term, the subterm, $[T_0 + T_0]$, arises when the first successful broadcast is the second largest element and the second successful broadcast is the largest element. Thus, the layers between $x_0 = -\infty$, the second and the largest element are both empty and it takes T_0 time slots in both layers.

Shiau and Yang [15] proved that $M_2 = 5$, then we have

$$T_2 = 5 + \{\frac{1}{2}[1] + \frac{1}{2}[1 + 1]\} = \frac{13}{2}. \tag{3}$$

If $n = 3$, we also have the following recursive formula:

$$T_3 = M_3 + R_3, \text{ where } R_3 = \left\{ \begin{array}{rrr} & \frac{1}{3}[& T_2] \\ + & \frac{1}{3}[T_0 & +T_1] \\ + & \frac{1}{3}[R_2 & +T_0] \end{array} \right\}. \tag{4}$$

In the equation for R_3, the first successful broadcast may be any one of the 3 elements randomly, each case having probability $\frac{1}{3}$. The subterm of first term, T_2, arises when the first successful broadcast is the largest element. Thus, the layer between $x_0 = -\infty$ and the largest element contains two elements, which is the second and third largest element, and it takes T_2 time slots for sorting.

In the second term, the subterm, $[T_0 + T_1]$, arises when the first successful broadcast is the second largest element and the second successful broadcast is the largest element. Thus, the layer between the second and the largest element is empty and it takes T_0 time slots. And the layer between $x_0 = -\infty$ and the second largest element contains one element and it takes T_1 time slots.

In third term, the subterm, $[R_2 + T_0]$, arises when the first successful broadcast is the third largest element. Thus, after the first successful broadcast, there are two possible successful broadcast sequences, each having probability $\frac{1}{2}$. The first case is that the second successful broadcast is the largest data element. Thus, the layer between the first two successful broadcasts contains only one data element which is the second largest data element. The second case is that the second successful broadcast is the second largest data element, and the largest data element will be broadcast in the third successful broadcast. Thus, the layers between the first, second and third successful broadcasts, are both empty. Comparing with R_2 in Eq.(2), we can find that the above two cases are equivalent to those in R_2. Thus, R_2 represents the number of time slots required for the layers bounded by the sequence of successful broadcasts L_m in finding the maximum. Note that the time slots required for L_m is counted in M_3. Finally, the layer between $x_0 = -\infty$ and the third largest element is empty and it takes T_0 time slots.

Generalizing Eq.(4), we have

$$T_n = M_n + R_n, \text{ where } R_n = \left\{ \begin{array}{lll} & \frac{1}{n}[& T_{n-1}] \\ + & \frac{1}{n}[T_0 & +T_{n-2}] \\ + & \frac{1}{n}[R_2 & +T_{n-3}] \\ + & \cdots & \\ + & \frac{1}{n}[R_{k-1} & +T_{n-k}] \\ + & \cdots & \\ + & \frac{1}{n}[R_{n-2} & +T_1] \\ + & \frac{1}{n}[\ R_{n-1} & +T_0] \end{array} \right\}. \tag{5}$$

In the above equations, M_n represents the average number of time slots required for finding the maximum among n data elements, and R_n represents the average number of time slots required for finishing sorting after the maximum is found.

In the equation for R_n, the first successful broadcast may be any one of the n elements randomly, each case having probability $\frac{1}{n}$. The meanings of T_{n-1} and $T_0 + T_{n-2}$ are similar to those in Eq.(4).

In the kth term, the subterm, $R_{k-1}+T_{n-k}$, arises when the first successful broadcast is the kth largest element. Thus, R_{k-1} is the number of time slots required for finishing the sorting on the $k-1$ elements after the maximum is found. And the layer between $x_0 = -\infty$ and the first successful broadcast element contains $n-k$ data elements and it takes T_{n-k} time slots for sorting $n-k$ data elements.

Substituting $R_k = T_k - M_k$, $2 \le k \le n-1$ into Eq.(5), we have

$$T_n = M_n + \left\{ \begin{array}{lll} & \frac{1}{n}[& T_{n-1}] \\ + & \frac{1}{n}[T_0 & +T_{n-2}] \\ + & \frac{1}{n}[(T_2 - M_2) & +T_{n-3}] \\ + & \cdots & \\ + & \frac{1}{n}[(T_{k-1} - M_{k-1}) & +T_{n-k}] \\ + & \cdots & \\ + & \frac{1}{n}[(T_{n-2} - M_{n-2}) & +T_1] \\ + & \frac{1}{n}[(T_{n-1} - M_{n-1}) & +T_0] \end{array} \right\}. \tag{6}$$

Regrouping the above equation, we obtain

$$T_n = M_n + \frac{1}{n}[T_0 + \sum_{2 \le k \le n-1} (T_k - M_k) + \sum_{0 \le k \le n-1} T_k]. \tag{7}$$

Lemma 1

$$\frac{1}{n+1} T_n - \frac{1}{3} T_2 = \sum_{3 \le k \le n} \frac{1}{k+1} (M_k - M_{k-1}).$$

Proof. Rearranging Eq.(7), we have

$$T_n = M_n + \frac{1}{n}[T_0 + \sum_{2 \le k \le n-2} (T_k - M_k) + (T_{n-1} - M_{n-1}) + \sum_{0 \le k \le n-2} T_k + T_{n-1}]. \tag{8}$$

Substituting $n-1$ into Eq.(7), we get

$$T_{n-1} = M_{n-1} + \frac{1}{n-1}[T_0 + \sum_{2 \le k \le n-2} (T_k - M_k) + \sum_{0 \le k \le n-2} T_k]. \tag{9}$$

Substituting Eq.(9) into Eq.(8), we get

$$\begin{array}{rl} T_n & = M_n + \frac{1}{n}[(n-1)(T_{n-1} - M_{n-1}) + (T_{n-1} - M_{n-1}) + T_{n-1}] \\ T_n & = M_n - M_{n-1} + \frac{n+1}{n} T_{n-1} \\ \frac{1}{n+1} T_n - \frac{1}{n} T_{n-1} & = \frac{1}{n+1}(M_n - M_{n-1}). \end{array}$$

Then, we have

$$\begin{array}{lll} \frac{1}{n+1} T_n & -\frac{1}{n} T_{n-1} & = \frac{1}{n+1}(M_n - M_{n-1}) \\ \frac{1}{n} T_{n-1} & -\frac{1}{n-1} T_{n-2} & = \frac{1}{n}(M_{n-1} - M_{n-2}) \\ \frac{1}{n-1} T_{n-2} & -\frac{1}{n-2} T_{n-3} & = \frac{1}{n-1}(M_{n-2} - M_{n-3}) \\ & & \cdots \\ \frac{1}{4} T_3 & -\frac{1}{3} T_2 & = \frac{1}{4}(M_3 - M_2). \end{array}$$

Summing the above equalities, we obtain

$$\frac{1}{n+1} T_n - \frac{1}{3} T_2 = \sum_{3 \le k \le n} \frac{1}{k+1} (M_k - M_{k-1}).$$

■

Lemma 2

$$4(\frac{1}{3}-\frac{1}{n+1}) \leq \sum_{3\leq k\leq n}\frac{1}{k+1}(M_k - M_{k-1}) \leq 5(\frac{1}{3}-\frac{1}{n+1}).$$

Proof. Substituting Eq.(1) into$\frac{1}{n+1}(M_n - M_{n-1})$, we get

$$\begin{array}{rll} \frac{1}{n+1}\frac{4}{n} & \leq \frac{1}{n+1}(M_n - M_{n-1}) & \leq \frac{1}{n+1}\frac{5}{n} \\ 4(\frac{1}{n}-\frac{1}{n+1}) & \leq \frac{1}{n+1}(M_n - M_{n-1}) & \leq 5(\frac{1}{n}-\frac{1}{n+1}). \end{array}$$

Similarly, we obtain

$$\begin{array}{rll} 4(\frac{1}{n-1}-\frac{1}{n}) & \leq \frac{1}{n}(M_{n-1} - M_{n-2}) & \leq 5(\frac{1}{n-1}-\frac{1}{n}) \\ & \cdots & \\ 4(\frac{1}{3}-\frac{1}{4}) & \leq \frac{1}{4}(M_3 - M_2) & \leq 5(\frac{1}{3}-\frac{1}{4}). \end{array}$$

Summing the above inequalities, we obtain

$$4(\frac{1}{3}-\frac{1}{n+1}) \leq \sum_{3\leq k\leq n}\frac{1}{k+1}(M_k - M_{k-1}) \leq 5(\frac{1}{3}-\frac{1}{n+1}).$$

■

Theorem 3

$$\frac{7}{2}n - \frac{1}{2} \leq T_n \leq \frac{23}{6}n - \frac{7}{6}.$$

Proof. By Lemma 1 and Lemma 2, we obtain

$$4(\frac{1}{3}-\frac{1}{n+1}) \leq \frac{1}{n+1}T_n - \frac{1}{3}T_2 \leq 5(\frac{1}{3}-\frac{1}{n+1}).$$

Substituting Eq.(3) $T_2 = \frac{13}{2}$ into the above inequality, we get

$$\begin{array}{rll} (\frac{4}{3}+\frac{1}{3}\frac{13}{2})(n+1)-4 & \leq T_n & \leq (\frac{5}{3}+\frac{1}{3}\frac{13}{2})(n+1)-5 \\ \frac{21}{6}(n+1)-4 & \leq T_n & \leq \frac{23}{6}(n+1)-5 \\ \frac{7}{2}n-\frac{1}{2} & \leq T_n & \leq \frac{23}{6}n-\frac{7}{6}. \end{array}$$

Thus, the proof completes. ■

Note that the companion paper[5] provides the asymptotic bound $T_n \sim 3.679826\ldots n$.

Since the computation before each broadcast in each processor requires only constant time and each time slot needs constant time, by Lemma 3, we have the following theorem.

Theorem 4

The average time complexity of our sorting algorithm based on the maximum finding is $\Theta(n)$.

3 The second sorting algorithm based on loser selection

In this section, the loser selection algorithm will be reviewed briefly. It was proposed and analyzed by Prodinger[12](also see [14]). Based on the loser selection, we present our second sorting algorithm in Subsection 3.2 and its analysis in Subsection 3.3.

3.1 The previous results for loser selection

In the loser selection problem[12], it is assumed that a party of n persons want to select one of the members (the loser) to pay a beer for everybody. The procedure is as follows: Everybody flips a coin (with outputting head and tail, each with probability $\frac{1}{2}$); then, recursively, the head-party continues to select one of their members. However, there is one exception: If all persons have thrown "tails", then all of them have to repeat the procedure. The loser selection algorithm is as follows:

Algorithm Loser-Selection: $SelectLoser(C)$

Step 1: Each data element in C flips a coin(The probability of getting a head is $\frac{1}{2}$).

Step 2: All which get heads form C' and each data element in C' broadcasts its value.

Step 3: There are three possible cases:

Case 3a: If a broadcast conflict occurs then $Cr = SelectLoser(C')$.

Case 3b: If exactly one data element successfully broadcasts its value then return Cr.

Case 3c: If no data element broadcasts, go to step 1.(No data element in C got head.)

This whole procedure is called a round. When the conflict has been resolved and the loser has been selected, we say that "the round is over ". Let S_n denote the average number of coin flips for selecting a loser from n persons.

Prodinger[12] used Rice's method to show that S_n has an asymptotic logarithmic bound. And he also pointed out that even though the loser selection problem seems to be very simple, the analysis is not totally trivial. By fundamental analysis, we show that S_n has a basic bound(it is easily obtained by Lemma 8).

Proposition 5 *[12]*
$S_0 = S_1 = 0$, *and for* $n \geq 2$,

$$S_n \left(1 - 2^{-n}\right) = 1 + \sum_{k=0}^{n} 2^{-n} C_k^n S_k \ .$$

Therefore we find that $S_2 = 2$, $S_3 = \frac{7}{3}$, $S_4 = \frac{8}{3}$.

Lemma 6 *[12]*

The average depth S_n of the tree built by n persons who are selecting a loser by a coin flipping process is

$$S_n \sim \log_2 n + \frac{1}{2} - \delta(\log_2 n)$$

with the periodic function (of period 1 and very small amplitude)

$$\delta(x) = \frac{1}{L} \sum_{k \neq 0} \zeta(1 - \chi_k)\Gamma(1 - \chi_k) e^{2k\pi i x}$$

where $\frac{1}{L} = \log 2$ and $\chi_k = \frac{2k\pi i}{L}, k \in Z$. $\zeta(s)$ denotes Riemann's ζ-function; $\Gamma(s)$ denotes the Γ-function.

3.2 The sorting algorithm based on loser selection

Based on the loser selection[12], our second algorithm make use of the successful broadcast(the loser) to build two layers and then to spread the participator into the two layers.

When a successful broadcast(the loser) occurs, a new upper layer is built up. All processors which participated the conflict are divided into two layers. Each data element, except the successful one, which is greater than or equal to the successful one(the loser) will bring itself up to the new upper layer. This new layer now becomes the active layer. And others which are smaller than the successful one will stay on the current layer. After each data element on the upper layer has broadcast its value, the current layer will become the active layer again. At any time, only the processors which are on the active layer may broadcast.

Our second sorting algorithm can be represented as a recursive function: $Sorting(M)$, where M is a set of data elements (processors). Our second sorting algorithm is as follows:

Algorithm Sorting: $Sorting(M)$

Step 1: Each data element in M broadcasts its value.

Step 2: There are three possible cases:

- Case 2a: If exactly one data element successfully broadcasts its value, then return the only one element as the sorted sequence.
- Case 2b: If no data element broadcasts then return a $NULL$ sequence.
- Case 2c: If a broadcast conflict occurs then $Cr = SelectLoser(M)$. And return $Sorting(U')$, Cr, $Sorting(L')$ as the sorted sequence, where U' contains all whose values greater than or equal to Cr (other than Cr itself), and L' contains all whose values smaller than Cr.

Figure 1 shows an example for illustrating our second sorting algorithm. Initially, all data elements broadcast. Hence, in time slot 1, all of 1,2,...,8 broadcast and a conflict

occurs. Then we apply the loser selection algorithm to resolve the conflict. After the first loser 4 is selected(4 is randomly selected), the layer structure is built and the party are divided into two parts. The first round is over. The larger numbers 5,6,7,8 are on the upper layer. On the other hand, the smaller numbers 1,2,3 are on the lower layer. And the upper layer becomes the active layer now. The rounds for dividing the active layer into two layers will be performed recursively. Until the active layer contains only one data element or no data element, the active layer will turn down to the lower layer.

round 1	2	3	4	5	6	7	8	9	10
s	s	s	n	s	n	x	s	x	x
		$\emptyset$	$\emptyset$						
	$7,6,8$	$\hat{8}$		$\emptyset$	$\emptyset$				
$5,7,6,8$	$\hat{5}$	$7,6$		$\hat{7}$					
$\hat{4}$	$\emptyset$			6		6	3	3	
$3,1,2$							$\hat{2}$		
							1		1

Figure 1: An example for the second sorting algorithm. Assume the data elements are 5, 7, 3, 4, 1, 6, 2, 8. A number with a hat,such as $\hat{5}$, means that it is selected as a loser in the round. $\emptyset$ denotes an empty set, s:select a loser, n:no data element broadcasts, x: exactly one data element broadcasts.

3.3 Analysis of the sorting algorithm based on loser selection

In this subsection, we shall prove that the average time complexity of our second sorting algorithm is $\Theta(n)$, where n is the number of input data elements. Let T_k denote the average number of time slots. When there is zero or one input data element for the algorithm, one empty slot or one slot for successful broadcast is needed. Thus $T_0 = 1$ and $T_1 = 1$. We have the following recursive formula:

$$\begin{aligned} T_n = S_n + \tfrac{1}{n}[\quad & (T_0 \quad + \quad T_{n-1}) \\ + \ & (T_1 \quad + \quad T_{n-2}) \\ + \ & \quad \cdots \\ + \ & (T_{n-1} \quad + \quad T_0) \quad]. \end{aligned} \tag{10}$$

Therefore we find that $T_2 = 4$.

We shall explain Eq.(10) term by term. The first term, S_n, represents the average number of time slots for loser selection. Then, the selected loser might be one of the n elements randomly, each case having probability $\frac{1}{n}$. The third term, $(T_0 + T_{n-1})$, arises when the selected loser is the largest element. Thus, the upper layer is empty and it takes T_0 time slots. The lower layer contains $n-1$ data elements which are smaller than the selected loser. The number of time slots required for this lower layer is T_{n-1}. The fourth term, $(T_1 + T_{n-2})$, arises when the second largest element is selected to be a loser. All other subterms have similar meanings.

Rearranging Eq. (10), we have

$$T_n = S_n + \frac{2}{n}\sum_{k=0}^{n-1} T_k \tag{11}$$

which includes two recursive functions, T_n and S_n.

Lemma 7

$$T_n = 2n + 2 - S_n + 2(n+1)\sum_{k=3}^{n} \frac{1}{k+1}(S_k - S_{k-1}).$$

Proof. Rearranging Eq. (11), we have

$$T_n = S_n + \frac{2}{n}\left(\sum_{k=0}^{n-2} T_k + T_{n-1}\right). \tag{12}$$

Substituting n with $n-1$, we get

$$T_{n-1} = S_{n-1} + \frac{2}{n-1}\sum_{k=0}^{n-2} T_k.$$

Rearranging the above equality, we have

$$\sum_{k=0}^{n-2} T_k = \frac{n-1}{2}\left(T_{n-1} - S_{n-1}\right).$$

Substituting the above equality into Eq. (12), we get

$$\begin{aligned} T_n &= S_n + \frac{2}{n}\left[\frac{n-1}{2}\left(T_{n-1} - S_{n-1}\right) + T_{n-1}\right] \\ T_n &= S_n + \frac{n-1}{n}T_{n-1} - \frac{n-1}{n}S_{n-1} + \frac{2}{n}T_{n-1} \\ T_n - \frac{n+1}{n}T_{n-1} &= S_n - \frac{n-1}{n}S_{n-1}. \end{aligned}$$

Dividing both sides by $n+1$, we have

$$\begin{aligned} \frac{1}{n+1}T_n - \frac{1}{n}T_{n-1} &= \frac{1}{n+1}S_n - \frac{1}{n+1}\frac{n-1}{n}S_{n-1} \\ &= \frac{1}{n+1}S_n - \frac{1}{n}\frac{n-1}{n+1}S_{n-1}. \end{aligned}$$

By the above equality, we have

$$\begin{aligned} \tfrac{1}{n+1}T_n - \tfrac{1}{n}T_{n-1} &= \tfrac{1}{n+1}S_n - \tfrac{1}{n}\tfrac{n-1}{n+1}S_{n-1} \\ \tfrac{1}{n}T_{n-1} - \tfrac{1}{n-1}T_{n-2} &= \tfrac{1}{n}S_{n-1} - \tfrac{1}{n-1}\tfrac{n-2}{n}S_{n-2} \\ &\cdots \\ \tfrac{1}{4}T_3 - \tfrac{1}{3}T_2 &= \tfrac{1}{4}S_3 - \tfrac{1}{3}\tfrac{2}{4}S_2 \end{aligned}$$

Summing the above equalities, we obtain

$$\begin{aligned}\tfrac{1}{n+1}T_n - \tfrac{1}{3}T_2 &= \tfrac{1}{n+1}S_n - \tfrac{1}{3}\tfrac{2}{4}S_2 + \sum_{k=4}^{n}\left(\tfrac{1}{k}S_{k-1} - \tfrac{1}{k}\tfrac{k-1}{k+1}S_{k-1}\right)\\ &= \tfrac{1}{n+1}S_n - \tfrac{1}{3}\tfrac{2}{4}S_2 + \sum_{k=4}^{n}\tfrac{1}{k}\tfrac{2}{k+1}S_{k-1}.\end{aligned}$$

Since $T_2 = 4$ and $S_2 = 2$, we have

$$\begin{aligned}\tfrac{1}{n+1}T_n &= \tfrac{4}{3} + \tfrac{1}{n+1}S_n - \tfrac{1}{3}\tfrac{2}{4}2 + \sum_{k=4}^{n}\tfrac{1}{k}\tfrac{2}{k+1}S_{k-1}\\ &= 1 + \tfrac{1}{n+1}S_n + 2\sum_{k=4}^{n}\left(\tfrac{1}{k}S_{k-1} - \tfrac{1}{k+1}S_{k-1}\right)\\ &= 1 + \tfrac{1}{n+1}S_n + 2(\tfrac{1}{4}S_2 - \tfrac{1}{n+1}S_n) + 2\sum_{k=3}^{n}\tfrac{1}{k+1}(S_k - S_{k-1})\\ &= 2 - \tfrac{1}{n+1}S_n + 2\sum_{k=3}^{n}\tfrac{1}{k+1}(S_k - S_{k-1}).\end{aligned}$$

Multiplying both sides by $n+1$, we have

$$\begin{aligned}T_n &= 2\,(n+1) - S_n + 2(n+1)\sum_{k=3}^{n}\frac{1}{k+1}(S_k - S_{k-1})\\ &= 2n + 2 - S_n + 2(n+1)\sum_{k=3}^{n}\frac{1}{k+1}(S_k - S_{k-1}).\end{aligned}$$

This completes the proof. ■

The above lemma will be divided into two parts to get the bounds of T_n. The first part S_n is solved by Prodinger[12]. We shall focus on the second part $\sum_{k=3}^{n}\frac{1}{k+1}(S_k - S_{k-1})$ and show it can be bounded.

In Lemma 6, fluctuations with a similar pattern, such as $\delta(x)$, surface in a great many areas of the analysis of algorithms. Their amplitude is usually $\ll 10^{-5}$ [3]. To apply the precise asymptotic formula and its fluctuation into the proof can be found in the companion paper[5]. However, we shall apply the basic mathematical method [15], in the following, to obtain that $S_n - S_{n-1}$ can be bounded within a tight range. It will be applied into $\sum_{k=3}^{n}\frac{1}{k+1}(S_k - S_{k-1})$ in Lemma 7. And the bound can be found easily.

Lemma 8

$$\frac{1}{n} \le S_n - S_{n-1} \le \frac{2}{n}, for\ n \ge 3.$$

Proof. By Proposition5, we have

$$\begin{aligned}S_n = 1 + \tfrac{1}{2^n}[\quad & C_n^n S_n\\ & + C_{n-1}^n S_{n-1}\\ & + \cdots\\ & + C_1^n S_1\\ & + C_0^n S_n \qquad].\end{aligned} \tag{13}$$

And,

$$\begin{aligned} S_{n-1} = 1 + \tfrac{1}{2^n}[\quad & 2C_{n-1}^{n-1}S_{n-1} \\ & +2C_{n-2}^{n-1}S_{n-2} \\ & +\cdots \\ & +2C_1^{n-1}S_1 \\ & +2C_0^{n-1}S_{n-1} \quad]. \end{aligned} \tag{14}$$

Subtracting Eq. (14) from Eq. (13) , we have

$$\begin{aligned} S_n - S_{n-1} = \tfrac{1}{2^n}[\quad & C_n^n S_n & & - C_{n-1}^{n-1}S_{n-1} \\ + \quad & C_{n-1}^n S_{n-1} - C_{n-1}^{n-1}S_{n-1} & & - C_{n-2}^{n-1}S_{n-2} \\ + \quad & C_{n-2}^n S_{n-2} - C_{n-2}^{n-1}S_{n-2} & & - C_{n-3}^{n-1}S_{n-3} \\ + \quad & \cdots \\ + \quad & C_3^n S_3 - C_3^{n-1}S_3 & & - C_2^{n-1}S_2 \\ + \quad & C_2^n S_2 - C_2^{n-1}S_2 & & - C_1^{n-1}S_1 \\ + \quad & C_1^n S_1 - C_1^{n-1}S_1 \\ + \quad & C_0^n S_n - C_0^{n-1}S_{n-1} & & - C_0^{n-1}S_{n-1} \quad]. \end{aligned}$$

Substituting $C_i^n - C_i^{n-1} = C_{i-1}^{n-1}$ and $S_1 = 0$ into the above equality, we get

$$\begin{aligned} (1 - \tfrac{1}{2^{n-1}})(S_n - S_{n-1}) = \tfrac{1}{2^n}[\quad & C_{n-2}^{n-1}(S_{n-1} - S_{n-2}) \\ + \quad & C_{n-3}^{n-1}(S_{n-2} - S_{n-3}) \\ + \quad & \cdots \\ + \quad & C_2^{n-1}(S_3 - S_2) \\ + \quad & C_1^{n-1}(S_2 - S_1) \\ - \quad & S_{n-1} \quad]. \end{aligned} \tag{15}$$

We shall prove this lemma by induction on n. The proof is divided into two parts. Now we prove the first part, $S_n - S_{n-1} \leq \frac{2}{n}$, for $n \geq 3$. When $n = 3$, it is trivially true since $S_3 - S_2 = \frac{7}{3} - 2 = \frac{1}{3} \leq \frac{2}{3}$. By hypothesis, assume that $S_3 - S_2 \leq \frac{2}{3}, S_4 - S_3 \leq \frac{2}{4}, \cdots, S_{n-1} - S_{n-2} \leq \frac{2}{n-1}$ are all true. We shall verify that $S_n - S_{n-1} \leq \frac{2}{n}$ is also true.

Substituting

$$\begin{aligned} & S_k - S_{k-1} \leq \tfrac{2}{k}, \text{ for } 4 \leq k \leq n-1, \\ & S_3 - S_2 = \tfrac{7}{3} - 2 = \tfrac{1}{3}, \\ & S_2 - S_1 = 2 - 0 = 2 \end{aligned}$$

into Eq. (15), we obtain the following inequality

$$\begin{aligned} & (1 - \tfrac{1}{2^{n-1}})(S_n - S_{n-1}) \\ & \leq \tfrac{1}{2^n}\left[\tfrac{2}{n-1}C_{n-2}^{n-1} + \tfrac{2}{n-2}C_{n-3}^{n-1} + \cdots + \tfrac{2}{4}C_3^{n-1} + \tfrac{1}{3}C_2^{n-1} + 2C_1^{n-1} - S_{n-1}\right]. \end{aligned} \tag{16}$$

Since

$$\tfrac{k}{n-i}C_{n-i-1}^{n-1} \quad = \tfrac{k}{n} \cdot C_{n-i}^n \text{ , for } 1 \leq i \leq n-2,$$

the expression of the right hand of Eq. (16) can be derived as follows:

$\frac{1}{2^n}\left[\frac{2}{n-1}C_{n-2}^{n-1}+\frac{2}{n-2}C_{n-3}^{n-1}+\cdots+\frac{2}{4}C_3^{n-1}+(\frac{2}{3}-\frac{1}{3})C_2^{n-1}+(\frac{2}{2}+\frac{2}{2})C_1^{n-1}-S_{n-1}\right]$
$=\frac{1}{2^n}[\frac{2}{n}C_{n-1}^{n}+\frac{2}{n}C_{n-2}^{n}+\cdots+\frac{2}{4}C_3^{n}+\frac{2}{n}C_2^{n}-\frac{1}{3}C_2^{n-1}+\frac{2}{2}C_1^{n-1}-S_{n-1}]$
$=\frac{1}{2^n}[\frac{2}{n}(2^n-2)-\frac{2}{n}C_1^n-\frac{1}{3}C_2^{n-1}+\frac{2}{2}C_1^{n-1}-S_{n-1}]$.

In the above expression, it can be easily verified that

$$-\tfrac{2}{n}C_1^n-\tfrac{1}{3}C_2^{n-1}+\tfrac{2}{2}C_1^{n-1}\le 0 \quad \text{,for } n\ge 3 \text{ and } n \text{ is an integer.}$$

We obtain

$$\begin{aligned}(1-\tfrac{1}{2^{n-1}})(S_n-S_{n-1}) &\le \tfrac{1}{2^n}\left[\tfrac{2}{n}(2^n-2)\right]\\ &= (1-\tfrac{1}{2^{n-1}})\cdot\tfrac{2}{n}.\end{aligned}$$

Dividing both sides by $1-\frac{1}{2^{n-1}}$, we have

$$S_n-S_{n-1}\le\frac{2}{n},\ \text{for } n\ge 3.$$

This finished the proof of the first part.

Since the second part, $S_n-S_{n-1}\ge\frac{1}{n}$, for $n\ge 3$ is quite similar to the first part, the proof of the second part is omitted. ■

Lemma 9

$$\frac{1}{3}-\frac{1}{n+1}\le\sum_{k=3}^{n}\frac{1}{k+1}(S_k-S_{k-1})\le 2(\frac{1}{3}-\frac{1}{n+1}).$$

Proof. By Lemma 8, we get

$$\begin{aligned}\sum_{k=3}^{n}\tfrac{1}{k+1}(S_k-S_{k-1}) &= \tfrac{1}{n+1}(S_n-S_{n-1})+\tfrac{1}{n}(S_{n-1}-S_{n-2})+\cdots+\tfrac{1}{4}(S_3-S_2)\\ &\le \tfrac{1}{n+1}\tfrac{2}{n}+\tfrac{1}{n}\tfrac{2}{n-1}+\cdots+\tfrac{1}{4}\tfrac{2}{3}\\ &=2(\tfrac{1}{3}-\tfrac{1}{n+1}).\end{aligned}$$

The proof of $\frac{1}{3}-\frac{1}{n+1}\le\sum_{k=3}^{n}\frac{1}{k+1}(S_k-S_{k-1})$ is similar. ■

Lemma 10

$$\frac{8}{3}n-S_n+\frac{2}{3}\le T_n\le\frac{10}{3}n-S_n-\frac{2}{3}.$$

Proof. Combining Lemma 7 and Lemma 9, we have

$$2n+2-S_n+2(n+1)(\frac{1}{3}-\frac{1}{n+1})\le T_n\le 2n+2-S_n+4(n+1)(\frac{1}{3}-\frac{1}{n+1})$$

Thus, $\frac{8}{3}n-S_n+\frac{2}{3}\le T_n\le\frac{10}{3}n-S_n-\frac{2}{3}$. ■

Note that the companion paper[5] provides the asymptotic bound $T_n\sim 2.878851\ldots n$.

By Lemma 10 and Lemma 6, we have the following theorem.

Theorem 11

The average time complexity of our sorting algorithm based on loser selection is $\Theta(n)$.

4 Conclusion

The layer concept [20] can help us to reduce conflict resolution when an algorithm is not conflict-free under the broadcast communication model. In this paper, we apply the layer concept to solve the sorting problem and propose two sorting algorithms. We show that the time complexities are both $\Theta(n)$. The second sorting algorithm based on the loser selection algorithm gets better performance than the first. We bounded them within a tight range using a basic mathematical method. Their asymptotic bound by mathematical analysis can be found in the companion paper [5].

The generalized sorting problem[18] is to sort the first k largest elements among n elements, $1 \leq k \leq n$. Obviously, the maximum finding problem and the sorting problem are two extreme cases of the generalized sorting problem, with $k = 1$ and $k = n$ respectively. Thus, the two sorting algorithms in this paper are of the extreme case when $k = n$. And the extreme case when $k = 1$ has been studied in our previous works[15, 16].

In viewpoint of the generalized sorting problem, if one wants to find the maximum element(when $k = 1$), our first sorting algorithm, based on the maximum finding, requires $\Theta(\log n)$ time[15], and our second sorting algorithm, based on the loser selection, requires $\Theta(\log^2 n)$ time[16]. It is clear, when $k = 1$, the first sorting algorithm is superior to the second one.

And in this paper, we proved the second sorting has better performance when $k = n$. We show that the second sorting algorithm wins the race in the end, and the result implies that the second sorting algorithm will speed up and rise above the first sorting algorithm after the first m elements are sorted among the n elements. We are interesting in finding what value m will be.

If the value of m is found, and one wants to sort the first l largest elements among n elements, where $l \geq m$, we shall suggest him to apply the second sorting algorithm to sort the first l largest elements among n. To obtain the value of m, we have to generalize the two sorting algorithms and find the relation between the two generalizations. Based on the first sorting algorithm and the generalization method of quicksort[18], we have generalized the first sorting algorithm and given complexity analysis[17]. One of our future works is to generalize the second sorting algorithm, and construct the relation between the two generalizations.

References

[1] J. I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, 25(5):505–515, May 1979.

[2] R. Dechter and L. Kleinrock. Broadcast communications and distributed algorithms. *IEEE Transactions on Computers*, 35(3):210–219, Mar. 1986.

[3] P. Flajolet and R. Sedgewick. Mellin transforms and asymptotics: Finite differences and rice's integrals. *Theoretical Computer Science*, 144:101–124, 1995.

[4] P. J. Grabner and H. Prodinger. An asymptotic study of a recursion occurring in the analysis of an algorithm on broadcast communication. *Information Processing Letters*, 65:89–93, 1998.

[5] P. J. Grabner and H. Prodinger. Sorting algorithms for broadcast communicatios: Mathematical analysis. private communication, submitted to Theoretical Computer Science.

[6] J. H. Huang and L. Kleinrock. Distributed selectsort sorting algorithm on broadcast communication. *Parallel Computing*, 16:183–190, 1990.

[7] S. Levitan. Algorithms for broadcast protocol multiprocessor. In *Proc. of 3rd International Conference on Distributed Computing Systems*, pages 666–671, 1982.

[8] S. P. Levitan and C. C. Foster. Finding an extremum in a network. In *Proc. of 1982 International Symposium on Computer Architechure*, pages 321–325, 1982.

[9] C. U. Martel. Maximum finding on a multi access broadcast network. *Information Processing Letters*, 52:7–13, 1994.

[10] C. U. Martel and M. Moh. Optimal prioritized conflict resolution on a multiple access channel. *IEEE Transactions on Computers*, 40(10):1102–1108, Oct. 1991.

[11] C. U. Martel, W. M. Moh, and T. S. Moh. Dynamic prioritized conflict resolution on multiple access broadcast networks. *IEEE Transactions on Computers*, 45(9):1074–1079, 1996.

[12] H. Prodinger. How to select a loser. *Discrete Mathematics*, 120:149–159, 1993.

[13] K. V. S. Ramarao. Distributed sorting on local area network. *IEEE Transactions on Computers*, C-37(2):239–243, Feb. 1988.

[14] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms.* Addison-Wesley Publishing Company, USA, 1996.

[15] S. H. Shiau and C. B. Yang. A fast maximum finding algorithm on broadcast communication. *Information Processing Letters*, 60:81–96, 1996.

[16] S. H. Shiau and C. B. Yang. The layer concept and conflicts on broadcast communication. *Journal of Chang Jung Christian University*, 2(1):37–46, June 1998.

[17] S. H. Shiau and C. B. Yang. A fast sorting algorithm and its generalization on broadcast communications. In *Proc. of Sixth Annual International Computing and Combinatorics Conference(COCOON'2000), Bondi Beach, Sydney, Australia*, 2000.

[18] S. H. Shiau and C. B. Yang. The generalization of quicksort. In *Proc. of 2000 Workshop on Internet and Distributed Systems(WIND'2000), Taiwan, R.O.C.*, pages 94–99, 2000.

[19] C. Y. Tang and M. J. Chiu. Distributed sorting on the serially connected local area networks. In *Proc. of 1989 Singapore International Conference on Networks*, pages 458–462, 1989.

[20] C. B. Yang. Reducing conflict resolution time for solving graph problems in broadcast communications. *Information Processing Letters*, 40:295–302, 1991.

[21] C. B. Yang, R. C. T. Lee, and W. T. Chen. Parallel graph algorithms based upon broadcast communications. *IEEE Transactions on Computers*, 39(12):1468–1472, Dec. 1990.

[22] C. B. Yang, R. C. T. Lee, and W. T. Chen. Conflict-free sorting algorithm broadcast under single-channel and multi-channel broadcast communication models. In *Proc. of International Conference on Computing and Information*, pages 350–359, 1991.