Probabilistic Integer Sorting

Alexandros V. Gerbessiotis

CS Department, New Jersey Institute of Technology, Newark, NJ 07102, USA.

Constantinos J. Siniolakis

The American College of Greece, 6 Gravias St., Aghia Paraskevi, Athens, 15342, Greece.

Abstract

We introduce a probabilistic sequential algorithm for stable sorting n uniformly distributed keys in an arbitrary range. The algorithm runs in linear time and sorts all but a very small fraction $2^{-\Omega(n)}$ of the input sequences; the best previously known bound was $2^{-\Omega(n/(\lg n \lg\lg n))}$. An EREW PRAM extension of this sequential algorithm sorts in $O((n/p + \lg p) \lg n / \lg (n/p + \lg n))$ time using $p \leq n$ processors under the same probabilistic conditions. For a CRCW PRAM we improve upon the probabilistic bound of $2^{-\Omega(n/(\lg n \lg\lg n))}$ obtained by Rajasekaran and Sen to derive a $2^{-\Omega(n \lg\lg n/ \lg n)}$ bound. Additionally, we present experimental results for the sequential algorithm that establish the practicality of our method.

Key words: Integer sorting - sorting uniform keys - probabilistic analysis.

1 Introduction

We introduce and analyze a probabilistic sequential algorithm for sorting uniformly distributed integers in an arbitrary range and also extended to work under the PRAM model of parallel computation. This is an important special case of integer sorting with a significant number of applications [6,7]. The parallel algorithm extensions we describe work on an EREW and CRCW PRAM. Several probabilistic algorithms have been proposed in the past for the problem of sorting n uniformly distributed integer keys. In [7], an algorithm is presented that sorts in $O(n)$ expected-time, and the same time bound is obtained in [8]. An improvement of the result in [8] is presented in [5], that yields an integer sorting algorithm that runs in $O(n)$ time with probability at least $1 - c^{-\sqrt{n}}$, for any constant $c > 0$, i.e. it sorts all but a very small fraction $c^{-\sqrt{n}}$ of the input sequences. A PRAM CRCW version of this algorithm sorts

in $O(\lg n)$ time using $n/\lg n$ processors under the same probabilistic bound. In an independent work, Chlebus [3], by using a more complicated algorithm realizes the same time bound as in [5], but achieves the probability bound of $1 - 2^{-cn/\lg^3 n}$, i.e. it sorts all but a very small fraction $2^{-cn/\lg^3 n}$ of the input sequences. Finally, in [10] a CRCW randomized version of the algorithm in [7] is presented that sorts in $O(\lg n)$ time using $n/\lg n$ processors with probability at least $1 - n^{-c}$ all but a small fraction $2^{-cn/(\lg n \lg\lg n)}$ of the input sequences. The sequential version of the algorithm in [10] improves, however, the probability bounds of previous approaches to $1 - 2^{-cn/(\lg n \lg\lg n)}$. We note that the parallel versions of the algorithms in [3,5] are not stable as opposed to the algorithm in [10]. As in the algorithm in [10] probability depends on the choices of the input keys and the ones in the randomized algorithm, in the remainder we shall use the terminology of [10, page 12 and Theorem 6.1] to describe the performance of the algorithm in [10] as $\tilde{O}(\lg n)$ with probability $1 - 2^{-cn/(\lg n \lg\lg n)}$. Running time is $\tilde{O}(f(n))$ if it is $O(f(n))$ with high probability, i.e. with probability at least $1 - 1/n^c$ for any constant $c > 0$. A probabilistic algorithm sorts within a stated time bound with probability at least $1 - g(n)$ ($g(n) < 1$, for all n) if and only if it sorts all but a very small fraction $g(n)$ of its input sequences.

We introduce a sequential integer stable sorting algorithm that sorts n uniformly distributed integers in an arbitrary range in $O(n)$ time and achieves the improved probability bound $1-2^{-cn}$, for any constant $c > 0$, i.e. our algorithm sorts all but a very small fraction 2^{-cn} of the input sequences. A straightforward extension of this algorithm to the EREW PRAM model sorts n keys with p processors on an EREW PRAM in time $O((n/p + \lg p)\lg n/\lg(n/p + \lg p))$ within the same probability bound. By using the techniques introduced in this work we also improve on the probabilistic bound of [10] on the CRCW PRAM; the new bound of $1-2^{-cn\lg\lg n/\lg n}$ improves upon the former bound of $1-2^{-cn/(\lg n \lg\lg n)}$ of [10]. We finally present experimental results that compare the performance of our algorithm to quicksort (a comparison-based sorting algorithm used as a control algorithm), radix-sort, and hybrid-sort [5,6,8] and show the practicality and optimality of our designs.

We first show a technical combinatorial lemma and for that we present some well-known probabilistic bounds. Let $X_1, \ldots, X_n$ be a sequence of n independent Bernoulli random variables each with mean P, and let $\log x$, $\lg x$ be the logarithm of x to the base e and two respectively. Then, the random variable $X_{n,P} = \sum_{i=1}^{n} X_i$ follows a binomial distribution. We therefore obtain the following bound for the probability on the tail of the binomial distribution [2]. For any $u > 1$, $\text{Prob}[X_{n,P} \geq unP] \leq (\frac{e}{u})^{unP} e^{-nP}$. We also use the following result of [1], for all integers n, m, $n \geq m$: $\binom{n}{m} \leq (\frac{ne}{m})^m$. The main result of this section follows. It is assumed that k is large enough for $(eD/B)(ed/D)^D \log k \leq k^\beta$ and $d < k^\beta$ to hold.

Lemma 1.1 *For any positive constants d and β, if k balls are independently placed into $k^{1+\beta}$ bins with probability at most $d/k^{1+\beta}$ of a ball being placed in a fixed bin, then there are at most $Bk/\lg k$ balls into bins of size greater than or equal to D with probability at least $1-2^{-ck+1}$, for any positive constants B, c and D such that $0 < c < \beta(B - 2B/D)$.*

Proof : For any positive constant B, denote by $\mathcal{E}$ the event that there are at least $\hat{B} = Bk/(D \lg k)$ bins of size greater than or equal to D. By definition, the probability of a ball being placed in a fixed bin is at most $d/k^{1+\beta}$. Then, by a combinatorial argument [5], we bound Prob[$\mathcal{E}$] as follows.

$$\begin{aligned}\text{Prob}[\mathcal{E}] &\leq \binom{k^{1+\beta}}{\hat{B}}\binom{k}{D}^{\hat{B}}(\frac{d}{k^{1+\beta}})^{D\hat{B}} \leq (\frac{eD}{B}k^{\beta}\lg k(\frac{ek}{D}\frac{d}{k^{1+\beta}})^{D})^{\hat{B}} \\ &\leq (\frac{eD}{B}k^{\beta}\lg k(\frac{ed}{Dk^{\beta}})^{D})^{\hat{B}}.\end{aligned}$$

By assumption, k is large enough to ensure that $(eD/B)(ed/D)^D \log k \leq k^{\beta}$. Therefore, for any $0 < c < \beta(B - 2B/D)$ the following bound is derived.

$$\text{Prob}[\mathcal{E}] \leq (k^{2\beta}\frac{1}{k^{\beta D}})^{\hat{B}} \leq (k^{2\beta-\beta D})^{\hat{B}} \leq 2^{(2\beta-\beta D)Bk/D} \leq 2^{-ck}.$$

We have therefore established that the probability that there are at least $\hat{B}$ bins of size greater than or equal to D is at most 2^{-ck} for some appropriately bounded positive constant c. Denote by $\mathcal{Z}$ the event that the total number of balls falling into these $\hat{B}$ bins is at least $Bk/\lg k$. Consider any fixed set of $\hat{B}$ bins out of the total $k^{1+\beta}$ bins. By definition, placing a ball into a bin is independent of all other placements. Therefore, the total number of balls falling into these bins is stochastically dominated by the binomially distributed random variable $X_{n,P} = X_{k,\hat{B}d/(k^{1+\beta})} = X_{k,dB/(Dk^{\beta}\lg k)}$. By employing the probability bound for the tail of the binomial distribution established earlier, the following upper bound on Prob[$\mathcal{Z}$] is derived.

$$\begin{aligned}\text{Prob}[\mathcal{Z}] &\leq \binom{k^{1+\beta}}{\hat{B}}\binom{k}{Bk/\lg k}(\frac{d\hat{B}}{k^{1+\beta}})^{Bk/\lg k} = \binom{k^{1+\beta}}{\hat{B}}(\frac{ed}{Dk^{\beta}})^{Bk/\lg k} \\ &\leq (\frac{eD}{B}(\frac{ed}{Dk^{\beta}})^{D}k^{\beta}\lg k)^{Bk/(D\lg k)}.\end{aligned}$$

By assumption, k is large enough to ensure that $(eD/B)(ed/D)^D \log k \leq k^{\beta}$. Therefore, the following bound holds for any $0 < \hat{c} < \beta(B - 2B/D)$.

$$\text{Prob}[\mathcal{Z}] \leq k^{-\beta(D-2)(Bk/(D\lg k))} \leq 2^{-\hat{c}k}.$$

We have established that the probability that at least $Bk/\lg k$ balls fall into some fixed set of $\hat{B} = Bk/(D\lg k)$ bins is at most $2^{-\hat{c}k}$. We have also shown that the probability that there are at least $Bk/(D\lg k)$ bins of size at least D is at most 2^{-ck}. Therefore, for any $0 < \hat{c} = c < \beta(B - 2B/D)$ and $D > 2$ – the latter inequality is for the right hand side of the former inequality involving $\hat{c}$ to be positive – there are at most $Bk/\lg k$ balls into bins of size greater than or equal to D with probability at least $1 - 2^{-ck+1}$, and the lemma follows. ■

Corollary 1.1 *For any positive constant $d \leq 2$, if k balls are independently placed into $k\lg k$ bins with probability at most $d/(k\lg k)$ of a ball being placed in a fixed bin, then there are at most $Bk/\lg k$ balls into bins of size greater than or equal to D with probability at least $1 - 2^{-k\lg\lg k/\lg k}$, for any positive constants $B \geq De$ and $D \geq 2e$.*

Proof : In Lemma 1.1, we first fix $\beta = \lg\lg k/\lg k$ so that $k^\beta = \lg k$. On the derivation for Prob$[\mathcal{E}]$, as $d \leq 2$, $D \geq 2e$ and $B \geq De$, we get that $D \geq de$ and $B \geq De$. Therefore the condition $(eD/B)(ed/D)^D \log k \leq k^\beta = \lg k$ is satisfied. Then, for $\beta = \lg\lg k/\lg k$ and $D \geq 2e$ we get that Prob$[\mathcal{E}] \leq 2^{(2\beta-\beta D)Bk/D} \leq 2^{-2k\lg\lg k/\lg k}$. For Prob$[\mathcal{Z}]$ we derive similarly Prob$[\mathcal{Z}] \leq k^{-\beta(D-2)(Bk/(D\lg k))} \leq 2^{-2k\lg\lg k/\lg k}$, and the corollary follows. ■

2 An Improved version of R-S-Sort

In this section we show how to modify the $O(n)$ expected-time Radix-plus-Sifting (R-S-Sort) algorithm of [7] to obtain a run-time of $O(n)$ with probability at least $1 - 2^{-cn+1}$, for any constant $c > 0$, under the assumption that the n input keys are uniformly distributed integers in an arbitrary range. We address in fact a slightly more general problem; its solution will provide us with an integer sorting algorithm. The main algorithmic techniques utilized in the design of our algorithm derive from the methods of [5,7,8,10]. The basic idea behind the algorithm in [7] is to sort the keys with respect to some reduced number of Most Significant Bits (MSBs). As noted in [10] this nearly sorts the input. Suppose for example that we have sorted the n integer keys with respect to their $\lceil\lg n\rceil$ MSBs. For $1 \leq i \leq 2^{\lceil\lg n\rceil}$, let S_i be the (possibly empty) sequence of keys with value i for their $\lceil\lg n\rceil$ MSBs. Then, the original sorting problem reduces to $2^{\lceil\lg n\rceil}$ disjoint subproblems, namely the problems of sorting $S_1, \ldots, S_{2^{\lceil\lg n\rceil}}$. In particular, the required sorted sequence is $S_1^* \circ \ldots \circ S_{2^{\lceil\lg n\rceil}}^*$, where S_i^* is the sequence of keys in S_i in sorted order and $\circ$ denotes concatenation. Therefore, the total time required for the algorithm is $O(n) + \sum_{i\in\{1,\ldots,2^{\lceil\lg n\rceil}\}} O(|S_i|\lg|S_i|)$ (if say, mergesort or heapsort is used

for sorting within each such set) which is known to be $O(n)$ on the average [7]. Our algorithm works for any universe U equipped with a total order $<$ that can be evaluated in constant time by one processor for any given pair of arguments. Let U^* be the set of ordered sequences $x_1, \ldots, x_m \in U$ with $x_1 < \ldots < x_m$. We assume keys of U and U^* to be represented by means of arrays in the obvious way. Given $X = x_1, \ldots, x_m \in U^*$ and $y \in U$, we define the *rank* of y in X, denoted by $\mathrm{rank}(y : X)$, to be the cardinality of the set $\{x_i \mid 1 \leq i \leq m \text{ and } x_i \leq y\}$. Similarly, given $X = x_1, \ldots, x_m \in U$, we define the *position* of x_i in X, denoted by $\mathrm{pos}(x_i : X)$, to be i. The following is shown.

Theorem 2.1 *Let U be a universe equipped with a total order $<$ that can be evaluated in constant time for any given pair of arguments. For any constant $\beta > 0$, let $f : U \to \{1, \ldots, \lceil n^{1+\beta} \rceil\}$ be a non-decreasing function that can be evaluated in constant time, and suppose that $X_1, \ldots, X_n$ are random variables such that for some positive constant d,*

$$Prob[f(X_i) = k] \leq \frac{d}{\lceil n^{1+\beta} \rceil}, \tag{1}$$

for all i such that $1 \leq i \leq n$ and $k \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$. Then, for any constant $c > 0$, there exists a sequential algorithm for stable sorting the sequence (of values) $X_1, \ldots, X_n$ in time $O(n)$ with probability at least $1 - 2^{-cn+1}$.

Remark. Suppose for example that $X_1, \ldots, X_n$ are independent and uniformly distributed over $[1, \ldots, \lceil n^{\alpha} \rceil]$, $\alpha \geq 1 + \beta$. Then, as noted in [5], equation (1) is satisfied with:

$$f(x) = \begin{cases} 1 & \text{if } x < 1, \\ \lfloor (x-1) \lceil \frac{\lceil n^{\alpha} \rceil}{\lceil n^{1+\beta} \rceil} \rceil^{-1} \rfloor + 1 & \text{if } 1 \leq x \leq \lceil n^{\alpha} \rceil, \\ n & \text{if } x > \lceil n^{\alpha} \rceil, \end{cases}$$

and $\mathrm{Prob}[f(X_i) = k] \leq 2/\lceil n^{1+\beta} \rceil$ for all $1 \leq i \leq n$ and $k \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$. Notice that the above function can be efficiently implemented by simply "extracting" from the x_i's an appropriate number of MSBs.

Proof of Theorem 2.1: We show how to modify the R-S-Sort algorithm in order to obtain $O(n)$ run-time with probability at least $1 - 2^{-cn+1}$, for any constant $c > 0$. The input is initially stored in array $S = \langle x_1, x_2, \ldots, x_n \rangle$. After termination of the algorithm the sorted sequence S is assumed to reside in an auxiliary array S^*. Algorithm iRS-Sort consists of the following six steps.

Step 1. For all $1 \leq i \leq n$, compute $y_i = f(x_i)$, while retaining for each key y_i the information pertaining to key x_i.

STEP 2. Sort the sequence (array) $\langle y_1, \ldots, y_n \rangle$ using a stable radix-sort algorithm, where $y_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, $1 \leq i \leq n$, is viewed as a $(1+\beta)$-digit number with digits in the range $[1, \ldots, n]$.

STEP 3. Let D be some positive constant to be specified later. Determine the $l_1 \leq l$ non-empty subsequences S_{v_i} satisfying $|S_{v_i}| < D$, with $v_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, $1 \leq i \leq l_1$, and sort each one of them by some deterministic stable sorter, while retaining for each key $x_j \in S$, $1 \leq j \leq n$, its position in both S and S_{v_i}, i.e., $\mathrm{pos}(x_j : S)$ and $\mathrm{pos}(x_j : S_{v_i})$ respectively.

STEP 4. For every non-empty subsequence S_{v_i}, $1 \leq i \leq l_1$, formed in step 3, place each key $x_j \in S_{v_i}$ in the output array S^*, at position $\mathrm{pos}(x_j : S) + (\mathrm{rank}(x_j : S_{v_i}) - \mathrm{pos}(x_j : S_{v_i}))$.

STEP 5. Determine the $l_2 = l - l_1$ non-empty subsequences S_{w_i}, with $w_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, $1 \leq i \leq l_2$, that satisfy $|S_{w_i}| \geq D$, and subsequently place their keys in a contiguous block of memory (array), say, $R = \langle r_1, \ldots, r_{|R|} \rangle$. Sort R by any deterministic $O(n \lg n)$ worst-case running-time stable comparison-sort algorithm, while retaining for each key $r_i \in R$ its position in both S and R, i.e., $\mathrm{pos}(r_i : S)$ and $\mathrm{pos}(r_i : R)$ respectively.

STEP 6. For all $i \in \{1, \ldots, |R|\}$, place r_i in the output array S^* at position $\mathrm{pos}(r_i : S) + (\mathrm{rank}(r_i : R) - \mathrm{pos}(r_i : R))$.

Step 1 requires $O(n)$ time, since by definition, $f(x_i)$, for any $x_i \in U$, can be evaluated in constant time, Step 2 uses a stable radix-sort algorithm [6]. When step 2 is executed, y_i, $1 \leq i \leq n$, belongs to the range $[1, \ldots, \lceil n^{1+\beta} \rceil]$, and, therefore, step 2 requires, by way of [6], time $O((\lg(n^{1+\beta})/\lg n)n) = O((1+\beta)n)$. For all $i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, let S_i be the (possibly empty) sequence $S_i = \langle x_j \mid 1 \leq j \leq n \text{ and } f(x_j) = i \rangle$, and let $S = S_1 \circ \ldots \circ S_{\lceil n^{1+\beta} \rceil} = \langle x_{w_1}, \ldots, x_{w_n} \rangle$ be the concatenation of the S_i's. The position $\mathrm{pos}(x_j{:}S)$ of any key x_j will be used in step 4 below to determine the rank of x_j in the sorted output sequence. Then, the original sorting problem reduces to solving $l \leq n$ disjoint and non-empty subproblems, namely, the problems of sorting $S_{u_1}, \ldots, S_{u_l}$ such that S_{u_i} is non-empty, where $u_i \in \{1, \ldots, n^{1+\beta}\}$, $1 \leq i \leq l$. In particular, the required sorted sequence S^* is $S^* = S^*_{u_1} \circ \ldots \circ S^*_{u_l}$, where $S^*_{u_i}$ is the sequence of keys in the non-empty S_{u_i} in sorted order. In step 3, if D is sufficiently small we can use specially tailored sorters to sort the S_{v_i}'s. Otherwise, we may employ any deterministic $O(n \lg n)$ running-time sorter (eg merge-sort). Both kinds of sorters can be found in [6]. Hence, since the number of non-empty sequences S_{v_i} is $l_1 \leq l$ and step 3 requires $\sum_{i=1}^{l_1} O(|S_{v_i}| \lg |S_{v_i}|) = O(l_1 D \lg D) = O(n)$ time, as D is a constant and $l_1 \leq l \leq n$. For the sorted S_{v_i}, the rank of x_j in S_{v_i} can be computed in constant time, and therefore, step 4 takes time $O(n)$. Step 5 requires $O(n + |R| \lg |R|)$ time and since the sequence R becomes sorted at the completion of step 5, the rank of r_i in R can be computed in constant time

	w_1			w_2			w_3		
Problem size	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}
Radix-Sort	1.00	91	8190	1.28	112	10586	14.86	199	16703
Hybridsort	0.28	19	5511	0.30	22	9066	0.32	25	13683
Quicksort	0.34	48	5862	0.36	55	7197	0.39	66	8785
Our Algorithm	0.12	14	5470	0.12	18	7052	0.14	25	8398

Table 1
Experimental results on a Linux-based PC.

in step 6 for each key, for a total of $O(|R|)$. The overall time complexity of the algorithm can be verified to be $O((1+\beta)n + |R| \lg |R|)$. Therefore, our goal is to show that for any positive constants c and β, there is some positive constant D such that $|R| = O(n/\lg n)$. Substituting in Lemma 1.1 for $k = n$ and for any positive constants B, c and D that satisfy the conditions of Lemma 1.1. the following is derived: $\text{Prob}[|R| \geq B\frac{n}{\lg n}] \leq 1 - 2^{-cn+1}$. In particular, for $\beta = 3/2$ and $D = 4$ we get that $\text{Prob}[|R| \geq 2\frac{n}{\lg n}] \leq 1 - 2^{-n+1}$. Therefore with probability at least $1 - 2^{-cn+1}$, it is $|R| \lg |R| = O(n)$ and the running time is $O(n)$ as claimed in the theorem. ■

We present indicative experimental results for IRS-Sort. We ran our experiments on a Sun Sparcstation 20 with 128Mbytes RAM and a Linux-based Pentium II 450Mhz PC with 128MB RAM. All software has been written in C – by the authors – and the source code has been compiled using the supplied `gcc` compiler on the Sun and PC platforms with option `-O3` set under the operating systems Solaris 2.7, and Linux. For each of three test problem instances, we performed a series of experiments, and the running times indicated are the average of those experiments. Table 1 and Table 2 compare the performance of quicksort (a comparison-based control algorithm), radix-sort, hybrid-sort and our newly proposed algorithm. Theorem 2.1 was used with $D = 5$ and $\beta = 2$. The input consisted of strings uniformly distributed over an alphabet of size 64. For the two platforms we used strings of length $w_1 = 40$, $w_2 = 60$ and $w_3 = 80$. All measurements are normalized to the performance of radix-sort on 64 keys with key length w_1 which is 0.00074 and 0.00073 seconds for the two platforms respectively. As indicated in Table 1 and Table 2, our algorithm outperforms all other algorithms in the problem instances tested.

In the remainder we describe an EREW PRAM version of the iRS-Sort algorithm. The algorithm sorts n keys on a p-processor EREW PRAM, where $p \leq n$, in time $O((n/p + \lg p) \lg n / \lg (n/p + \lg p))$ with probability at least $1 - 2^{-cn+1}$, for any constant $c > 0$. The analysis makes use of the EREW stable radix-sort algorithm of [9].

	w_1			w_2			w_3		
Problem size	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}
Radix-Sort	1.00	94	6049	0.63	64	4085	1.06	106	6692
Hybridsort	0.23	14	1977	0.26	16	2402	0.44	18	2749
Quicksort	0.15	15	1685	0.19	19	2006	0.29	23	2386
Our Algorithm	0.07	7	1544	0.07	9	1857	0.10	11	2163

Table 2
Experimental results on a Sparcstation 20.

Lemma 2.1 *A p-processor EREW PRAM algorithm for stable sorting n integers, $p \leq n$, in the range $[1, \ldots, \lceil n^\alpha \rceil]$, for any constant $\alpha \geq 1$, requires time $O((\lg n^\alpha / \lg(n/p + \lg p))(n/p + \lg p))$.*

The first multiplicative term in the bound for the running time indicates the number of rounds of count-sort used, and the second term indicates the running time of count-sort on n keys with p processors. We can thus verify that steps 1, 2, 3, 4, and 6 of the sequential iRS-Sort algorithm have corresponding steps in the parallel algorithm. Step 5 can be efficiently realized by using Cole's merge-sort algorithm [4] which sorts n keys with $p \leq n$ processors in time $O(n \lg n/p)$. We have therefore shown for the EREW PRAM model the following analogue of Theorem 2.1.

Theorem 2.2 *For any constant $c > 0$ there exists p-processor EREW PRAM algorithm for stable sorting all but a very small fraction 2^{-cn+1} of input sequences $X_1, \ldots, X_n$ in time $O((\lg n / \lg(n/p + \lg p))(n/p + \lg p))$, where $p \leq n$.*

Proof : We can verify that steps 1, 3, 4, and 6 of the sequential iRS-Sort algorithm have corresponding implementations in the parallel algorithm, and each such parallel step requires time $O(n/p + \lg p)$. Note that a parallel prefix operation on n keys with p processors on an EREW PRAM takes time $O(n/p + \lg p)$. By way of Lemma 2.1, step 2 requires time $O((\lg(n^{1+\beta}) / \lg(n/p + \lg p))(n/p + \lg p))$. Step 5 can be efficiently realized by using Cole's merge-sort algorithm [4] in time $O(n/p + \lg p)$, and the theorem follows. ■

The CRCW PRAM algorithm of [10] follows the steps of iRS-Sort and thus of the EREW PRAM algorithm with the major difference that in step 1, sorting is performed in [10] not by using a deterministic radix-sort EREW PRAM algorithm but the randomized CRCW PRAM algorithm of [9]. Keys in step 1 are sorted based on the values of the $\lg n + \lg \lg n$ most significant bits. The running time of this step is then $\tilde{O}(\lg n)$ for sorting n uniformly distributed integers. Sets are then formed based on the values of these $\lg n + \lg \lg n$ most significant bits. The remaining steps are those of the iRS-Sort and the EREW PRAM algorithm presented in the previous section. The improvement

in the probabilistic bound compared to the one claimed in [10] is obtained by analyzing the sizes of the sets formed in step 1. By using Corollary 1.1 all but $O(n/\lg n)$ sets have size at most D. The total size of the sets of size at least D is also by Corollary 1.1 $O(n/\lg n)$. The remaining steps are thus identical to those of the corresponding algorithm for the EREW PRAM. The probabilistic bound is worse, $2^{-(n\lg\lg n)/\lg n}$ rather than 2^{-cn+1}, an improvement over the $2^{-cn/(\lg n\lg\lg n)}$ bound of [10] The following is obtained.

Theorem 2.3 *For any constant $c > 0$ there exists $n/\lg n$-processor CRCW PRAM algorithm for stable sorting all but a very small fraction $2^{-n\lg\lg n/\lg n}$ of input sequences $X_1, \ldots, X_n$ in time $\tilde{O}(\lg n)$.*

This work was initiated at the Oxford University Computing Laboratory, Oxford, UK. An extended abstract appeared in the Workshop on Randomized Algorithms in Sequential Parallel and Distributed Computing (RALCOM'97), Greece, October 1997. The work of the first author was supported in part by EPSRC under grant GR/K16999 and of the second by a Bodossaki Foundation Graduate Scholarship. The experimental work of the first author was supported in part by NJIT SBR grant 421350 and NSF MRI-9977508.

References

[1] Béla Bollobás. *Random Graphs.* Academic Press, New York, 1984.

[2] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statistics,* 23:493-507, 1952.

[3] B. S. Chlebus. Parallel iterated bucket sort. *Information Processing Letters,* 31:181-183, 1989.

[4] R. Cole. Parallel merge sort. *SIAM Journal on Computing,* 17(4):770-785,1988.

[5] T. Hagerup. Hybridsort revisited and parallelized. *Information Processing Letters,* 32:35-39, 1989.

[6] D. E. Knuth. *The Art of Computer Programming. Volume III: Sorting and Searching.* Addison-Wesley, Reading, 1973.

[7] M. D. MacLaren. Internal sorting by radix plus sifting. *Journal of the ACM,* 13:404-411, 1966.

[8] H. Meijer and S. G. Akl. The design and analysis of a new hybrid sort algorithm. *Information Processing Letters,* 10:213-218, 1980.

[9] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM Journal on Computing,* 18:594-607, 1989.

[10] S. Rajasekaran and S. Sen. On parallel integer sorting. *Acta Informatica,* 29:1-15, 1992.