Programming Research Group

CONVERTING SORTING ALGORITHMS
TO FAST RANDOMIZED ONES VIA SEARCHING

A. V. Gerbessiotis and C. J. Siniolakis

PRG-TR-22-97

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Converting Sorting Algorithms to Fast Randomized Ones via Searching*

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

Programming Research Group,
Computing Laboratory,
Oxford University,
Parks Road,
Oxford OX1 3QD,
UK

June 1997

Abstract

In this work we present an algorithmic transformation, that given two algorithms implementing respectively, sorting and searching, provides a randomized sorting algorithm that with high probability requires less time than the original sorting algorithm. This claim is true for a wide range of the parameters that describe the running time of the procedures for sorting and searching.

Keywords: Analysis of algorithms - design of algorithms - randomized algorithms - sorting.

*The work of the first author was supported in part by EPSRC(UK) under grant GR/K16999, and the second author was supported by a Bodossaki Foundation Graduate Scholarship.

1 Introduction

In this work we present an algorithmic transformation, that given two algorithms implementing respectively, sorting (procedure SORT), and searching (procedure SEARCH) provides a randomized sorting algorithm RND_SORT that with high probability requires less time than SORT for a wide range of the parameters that describe the running time of procedures SORT and SEARCH.

Particularly, we claim that, if SORT is a sorting algorithm and SEARCH is a searching algorithm (on a sorted table of size n) with time complexity $O(n \lg^\alpha n)$ and $c_s \lg^\beta n$ respectively, where $\alpha \geq \beta$, then, there exists a randomized sorting algorithm RND_SORT that requires time $(c_s + o(1))n \lg^\beta n$, with high probability. The logarithm of x to the base e and 2 is denoted by $\log x$ and $\lg x$ respectively.

The claims summarized in the following section will be used in the proof of our main result.

2 Random Sampling

We present in this section results related to random sampling that follow those presented in [1, 2]. In the discussion and proofs to follow we use the notation originally used in [2]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n input keys ordered such that $x_i < x_{i+1}$, for $i \in \{1, 2, \ldots, n-1\}$. We are assuming that the keys are distinct because by appending to each key its index we can make them so. For integers $k \geq 2$ and $s \geq 1$, let $Y = \{y_1, y_2, \ldots, y_{ks-1}\}$ be a randomly chosen subset of $ks - 1$ keys of X, also ordered so that $y_i < y_{i+1}$, $i \in \{1, 2, \ldots, ks-2\}$. The following k subsets $X_0, X_1, \ldots, X_{k-1}$ are then formed.

$$\begin{aligned} X_0 &= \{x : x < y_s\}, \\ X_i &= \{x : y_{is} < x < y_{(i+1)s}\}, \quad i \in \{1, \ldots, k-2\}, \\ X_{k-1} &= \{x : y_{(k-1)s} < x\}. \end{aligned}$$

For all $i \in \{0, 1, \ldots, k-1\}$, let $n_i = |X_i|$ and $P_i(j)$ be the probability that $n_i = j$. The following is shown in [2]. For the binomial coefficients, we assume that $\binom{i}{j} = 0$ if $j > i$.

Claim 1 *For all i,*

$$P_i(j) = P(j) = \frac{\binom{j}{s-1}\binom{n-j-1}{(k-1)s-1}}{\binom{n}{ks-1}}.$$

We would like to establish upper bounds on the size of the X_i's by proving that the probability of having such a set of size greater than $(1+\varepsilon)$ times the mean $(n-k+1)/k$ is sufficiently small, in particular that

$$\sum_{j \geq \lceil (1+\varepsilon)(n-k+1)/k \rceil + 1} P_j$$

is very small, where $0 < \varepsilon < 1$. The following is shown in [2].

Claim 2 *Let $k \geq 2$, $s \geq 1$, $ks < n/2$, $n \geq 1$, $0 < \varepsilon < 1$, $\rho > 0$, and*

$$s \geq \frac{1+\varepsilon}{\varepsilon^2}\left(2\rho \log n + \log\left(2\pi k^2 (ks-1) e^{1/(3(ks-1))}\right)\right).$$

Then, the probability that any one of the X_i, for all $i \in \{0, 1, \ldots, k-1\}$, is of size more than $\lceil (1+\varepsilon)(n-k+1)/k \rceil$ is at most $n^{-\rho}$.

3 The Algorithmic Transformation

This section is devoted to the proof of the following theorem.

Theorem 1 *Let* Sort *be a sorting algorithm and* Search *be a searching algorithm with running time $O(n \lg^\alpha n)$ and $c_s \lg^\beta n$ respectively, where $\alpha \geq \beta$. Then, there exists a randomized sorting algorithm* Rnd_Sort *that requires time $(c_s + o(1)) n \lg^\beta n$, with probability at least $1 - n^{-O(\lg^\delta n)}$, for any constant $\delta \geq 0$.*

Proof: In the randomized sorting algorithm Rnd_Sort, we assume that the input is an array $X[1, \ldots, n]$. We require several additional arrays: the array $T[1, \ldots, n]$ holds the sorted output, and the arrays $A[1, \ldots, n-k+1]$, $C[1, \ldots, k]$, $D[0, \ldots, k]$, $S[1, \ldots, ks-1]$, and $\bar{S}[1, \ldots, k-1]$ provide temporary working space. The space requirements can be reduced if we re-utilize unused space, that is, arrays T and $\bar{S}$ are necessary and sufficient for the algorithm to execute. We assume the existence of a random number generator, realized by function Random(r), that generates uniformly distributed integers in the range $\{1, 2, \ldots, r\}$. Procedure Sort(X, n) takes two parameters, the set X of keys to be sorted, of cardinality n. Procedure Search(x, X, n) takes three parameters, the element x to be located in a sorted table X of cardinality n. The algorithm is presented in figure 1.

Analysis of the algorithm: In stage 1 a sample S of size $ks - 1$ is selected at random without replacement among the n input keys of X. The "oversampling" parameter s is chosen so that it satisfies the conditions of claim 2. We note that the operation $X \leftarrow X - X[r]$ can be easily implemented by swapping elements $X[r]$ and $X[n-i+1]$. Hence, this stage requires time $O(ks)$.

In stage 2 the sample S of size $ks - 1$ is sorted by employing Sort. By definition of Sort, this stage takes time $O(ks \lg^\alpha (ks))$. The sorted sample S is used in stage 3 to determine in $O(k)$ time the set $\bar{S}$ of $k - 1$ splitters that evenly partition S. In stage 4 the bucket destination of every element in $X - \bar{S}$ is determined with respect to the $k - 1$ splitters by employing Search. Since, $|X - \bar{S}| = n - k + 1$, by definition of Search, this stage takes time $c_s (n - k + 1) \lg^\beta (k-1)$.

In stage 5 the keys in $X - \bar{S}$ are redistributed according to their bucket destination. This stage is equivalent to one of integer sorting [4], where the $n - k + 1$ integers to be sorted belong to the range $\{1, 2, \ldots, k\}$, induced by the $k - 1$ splitters. It follows from the description of this stage that it takes time $O(n)$. Subsequently, in stage 6 the keys in each bucket are sorted by employing Sort. By way of claim 2 the size of each bucket is bounded above by $\lceil (1+\varepsilon)(n-k+1)/k \rceil$ with probability at least $1 - kn^{-\rho}$. Hence, by

```
Procedure RND_SORT(X, n)

Stage 1 :  for i ← 1 to ks − 1
               do r = RANDOM(n − i + 1)
                  S[i] = X[r]
                  X ← X − X[r]
Stage 2 :  SORT(S, ks − 1)
Stage 3 :  for i ← 1 to k − 1
               do S̄[i] = S[is]
Stage 4 :  X ← X ∪ (S − S̄)
           for i ← 1 to n − k + 1
               do A[i] = SEARCH(X[i], S̄, k − 1)
Stage 5 :  for i ← 1 to k
               do C[i] ← 0
           for i ← 1 to n − k + 1
               do C[A[i]] ← C[A[i]] + 1
           for i ← 2 to k
               do C[i] ← C[i] + C[i − 1]
                  D[i] ← C[i]
           for i ← n − k + 1 downto 1
               do T[C[A[i]]] ← X[i]
                  C[A[i]] ← C[A[i]] − 1
Stage 6 :  D[0] ← 0
           for i ← 1 to k
               do SORT(T[D[i − 1] + 1, . . . , D[i]], D[i] − D[i − 1])
Stage 7 :  T ← MERGE(T, S̄, n − k + 1, k − 1)
```

Figure 1: Algorithm RND_SORT.

definition of SORT, this stage takes time $O((1+\varepsilon)(n-k+1)\lg^{\alpha}((1+\varepsilon)(n-k+1)/k))$ with high probability.

Finally, in stage 7 the sorted sequence of splitters $\bar{S}$ is merged with the sorted sequence of keys $X - \bar{S}$. We note that no actual merging has to occur since, by way of stages 4 and 5, the rank of $\bar{S}$ in $X - \bar{S}$ is already known. Hence, this stage requires time $O(k)$.

We conclude that with high probability the overall time $T(n)$ of algorithm RND_SORT is bounded above, for $0 < \varepsilon < 1$, by

$$T(n) = c_s(n-k+1)\lg^{\beta}(k-1) + O\left((n-k+1)\lg^{\alpha}\frac{(n-k+1)}{k}\right) + O\left(ks\lg(ks)\right) + O\left(n\right).$$

By substituting in the expression for $T(n)$ for $\varepsilon = 1/\sqrt{\lg n}$, $s = (2\rho+3)\lg^2 n$, and $k = n/(\rho\lg^{2+\gamma} n))$, for any constant $\gamma > \alpha + 1$, we obtain that with probability at least

	Quick	Quick*	Ratio	Merge	Merge*	Ratio	Heap	Heap*	Ratio
64k	1.00	0.93	1.07	0.86	0.97	0.88	2.38	1.67	1.42
128k	1.97	1.88	1.05	1.91	2.03	0.94	5.10	3.40	1.50
256k	4.04	3.79	1.06	3.95	4.32	0.91	10.89	6.72	1.62
512k	8.27	7.73	1.07	8.60	8.77	0.98	23.18	12.53	1.84
1024k	17.56	15.78	1.11	18.74	17.98	1.04	49.41	21.54	2.29
2048k	36.86	33.74	1.09	39.57	36.00	1.10	106.82	37.18	2.87

Table 1: Experimental results. A * indicates that the corresponding algorithm has been tranformed by our method.

$1 - n^{-\rho+1}$ algorithm RND_SORT requires time

$$T(n) = c_s n \lg^{\beta} n + O(n(\lg \lg n)^{\alpha}),$$

since by definition ρ is $O(\lg^{\delta} n)$, for some constant $\delta \geq 0$. The theorem follows. ■

4 Remarks

In practical applications procedure SEARCH realizes a binary search method and therefore requires $c_s \lg n$ time, for some constant $c_s > 0$, to locate an item in a table of size n. Hence, $\beta = 1$ in the previous analysis, and our method can be used to speed-up any sorting algorithm SORT with time complexity $d\, n \lg^{\alpha} n$, $\alpha > 1$. For the special case $\alpha = 1$ [3, 4], our method provides an algorithm that improves upon multiplicative constant factors provided that $d > c_s$. We also note that this transformation can be used to speed-up sorting algorithms that require time $n^{1+\epsilon}$, for any $\epsilon > 0$.

These theoretical results are compared with our empirical results in table 1 using three $O(n \lg n)$ sequential benchmark algorithms: quick-sort, merge-sort, and heap-sort [3, 4]. We ran our experiments on a 75MHz TFP Silicon Graphics Power Challenge 64-bit RISC processor. All software has been written in C and was compiled using `cc` with options `-sopt -O3` under the operating system IRIX Release 6.2-BETA. For each of 6 test problem instances, we performed a series of 8 experiments, and the running times indicated are the average of those experiments. The column indicated as "Ratio" gives the ratio of the performance of a sorting algorithm to that of the corresponding transformed one (a transformed algorithm is indicated by appending a * to the name of a sorting algorithm). The input consisted of uniformly distributed strings. All measurements are normalized with respect to the performance of quick-sort on 64000 elements.

As indicated in table 1, the speedup achieved by our method is consistently greater than 1 (for all large enough problem sizes), and ranges from 0.88 to 2.87. Moreover, as outlined in figure 3 the empirical results are consistent with the theoretical ones. Particularly, as the problem size increases, the performance of our method converges, for all three algorithms to the practical performance of binary search (compare with data from figure 2 and table 1).

We finally note that algorithm RND_SORT can be easily parallelized if parallel counterparts of the sequential algorithms for sorting (SORT), searching (SEARCH), and integer sorting are available. The running time of integer sorting of n elements in the range $\{0, \ldots, k-1\}$, $k \leq n$, is not required to be linear, but it is sufficient to be $o(1)$ of the running time of searching n elements in a table of size k.

References

[1] W. D. Frazer and A. C. McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM*, 17(3):496-507, 1970.

[2] A. V. Gerbessiotis and L. G. Valiant. Direct Bulk-Synchronous Algorithms. *Journal of Parallel and Distributed Computing,* 22:251-267, 1994.

[3] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5:10-15, 1962.

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

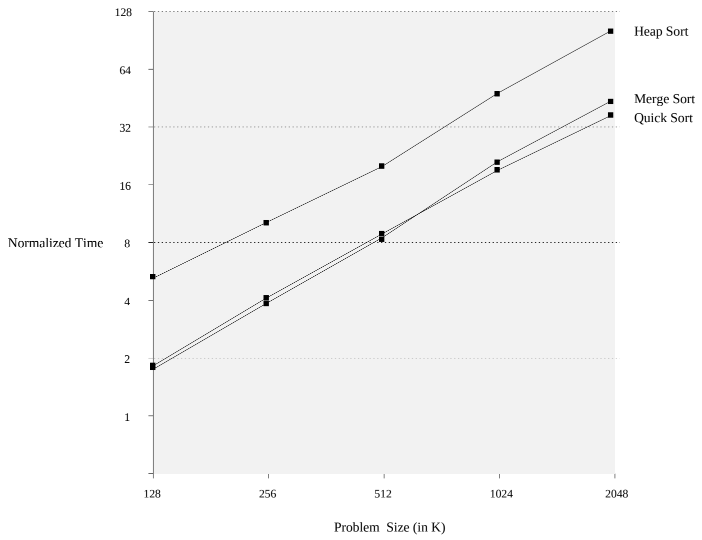

Figure 2: Performance of standard sorting algorithms.

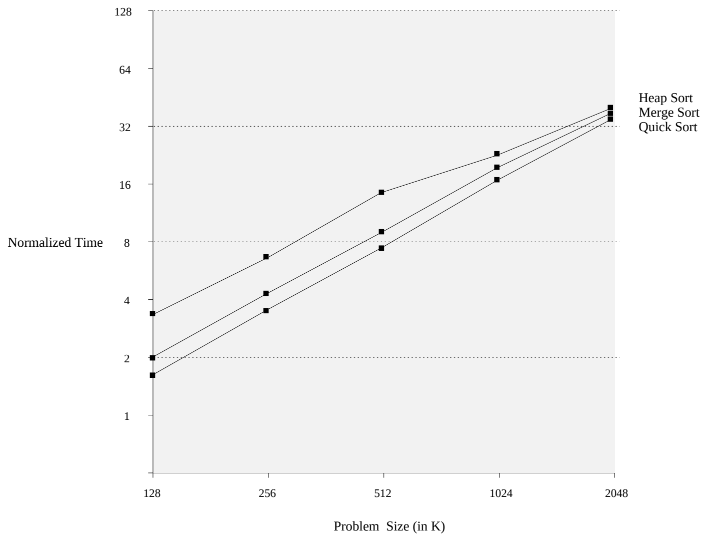

Figure 3: Performance of transformed sorting algorithms.