The Average Case Analysis of Partition Sorts*

Richard Cole   David C. Kandathil

Department of Computation,
Courant Institute of Mathematical Sciences, New York University,
251 Mercer Street, New York, NY 10012, USA.

Synopsis

This paper introduces a new family of in-place sorting algorithms, the *partition sorts*. They are appealing both for their relative simplicity and their efficient performance. They perform $\Theta(n \log n)$ operations on the average, and $\Theta(n \log^2 n)$ operations in the worst case.

The partition sorts are related to another family of sorting algorithms discovered recently by Chen [Che02]. He showed empirically that one version ran faster, on the average, than quicksort, and that the algorithm family performed $\Theta(n \log n)$ comparisons in the worst case; however no average case analysis was obtained.

This paper completes the analysis of Chen's algorithm family. In particular, a bound of $n \log n + O(n)$ comparisons and $\Theta(n \log n)$ operations is shown for the average case, and $\Theta(n \log^2 n)$ operations for the worst case. The average case analysis is somewhat unusual. It proceeds by showing that Chen's sorts perform, on the average, no more comparisons than the partition sorts.

Optimised versions of the partition sort and Chen's algorithm are very similar in performance, and both run marginally faster than an optimised quasi-best-of-nine variant of quicksort [BM93]. They both have a markedly smaller variance than the quicksorts.

1 Introduction

We consider the problem of in-place sorting of internally stored data with no a priori structure. We shall be concerned with the average case performance of deterministic sequential sorting algorithms. By in-place, we mean algorithms which use an additional $O(\log n)$ space; by average case, that all input orderings are equally likely.

A new family of in-place algorithms, which have a quicksort-like flavour, was proposed by Chen [Che02], and their worst case number of comparisons was shown to be $\Theta(n \log n)$. Empirical evidence was presented for the conjecture that their average case number of comparisons is close to the information theoretic lower bound and the fact that some versions run faster than quicksort on the average (for medium sized sets of order 10^4). The average case analysis of comparisons was posed as an open problem. Bounds on operation counts were not mentioned.

In this paper, we introduce a related new family of in-place algorithms, which we call *partition sorts*. The partition sorts are attractive, both because of their relative simplicity, and their asymptotic and practical efficiency. We show the following performance bounds for partition sort:

1. The average case number of comparisons is $n \log n + O(n)$, and for one version, is at most $n \operatorname{plog}(n+1) - 1.193n + O(\log n)$ comparisons[1]. They perform $\Theta(n \log^2 n)$ comparisons in the worst case.

*This work was supported in part by NSF grant CCR0105678.

[1]We let $\operatorname{plog} n$ denote the *piecewise log function* $\lfloor \log n \rfloor + \frac{2}{n}(n - 2^{\lfloor \log n \rfloor})$, which is an approximation to the log function, matching it at $n = 2^k$, for integers $k \geq 0$.

2. They perform $\Theta(n \log n)$ operations on the average and $\Theta(n \log^2 n)$ operations in the worst case.

As we show by means of a non-trivial argument, the average comparison cost of the partition sorts upper bounds that of Chen's algorithm family; this bound is also used in the analysis of exchanges for Chen's sorts. Using this and additional arguments, we obtain the following bounds for Chen's family of algorithms:

1. The number of comparisons performed by each version of Chen's algorithm is no more than the number of comparisons performed by a corresponding version of partition sort, on the average. Hence Chen's algorithm family performs $n \log n + O(n)$ comparisons on the average. We also give a simple and tight analysis for the worst case number of comparisons.

2. They perform $\Theta(n \log n)$ exchanges on the average and $\Theta(n \log^2 n)$ exchanges in the worst case.

The quicksort partition procedure is a central routine in both families of algorithms, and as a consequence, they both have excellent caching behaviour. Their further speedup relative to quicksort is due to a smaller expected depth of recursion.

We outline the analysis of the partition sort (Section 2) and the average case cost of comparisons for Chen's algorithm (Section 3). We also present some empirical studies of these algorithms (Section 4).

Prior Work

It is well known [FJ59] that, for comparison based sorts of n items, $\log n! = n \log n - n \log e + \frac{1}{2} \log n + O(1) = n \log n - 1.443n + O(\log n)$ is a lower bound on both the average and the worst case number of comparisons, in the decision tree model. Merge-insertion sort [FJ59] has the best currently known worst case comparison count: $n\,\mathrm{plog}\,(\frac{3}{4}n) - n + O(\log n) = n\,\mathrm{qlog}\,n - 1.441n + O(\log n)$, at the expense of $\Theta(n^2)$ exchanges[2]; it is, however, not clear how to implement it in $\Theta(n \log n)$ operations while maintaining this comparison count; the best result is an in-place mergesort which uses merge-insertion sort for constant size subproblems [Rei92]; this achieves $\Theta(n \log n)$ operations and comes arbitrarily close to the above comparison count, at the cost of increasingly large constants in the operation count. The trivial lower bound of $\Theta(n)$ on both the average and the worst case number of exchanges is met by selection sort at the expense of $\Theta(n^2)$ comparisons. A different direction is to simultaneously achieve $\Theta(n \log n)$ comparisons, $\Theta(n)$ exchanges, and $\Theta(1)$ additional space, thus meeting the lower bounds on all resources; this was recently attained [FG03] but its efficiency in practice is unclear.

Hoare's quicksort [Hoa61, Hoa62] has been the in-place sorting algorithm with the best currently known average case performance, namely $2n \ln n = 1.386n \log n$ comparisons, with worst case number of exchanges $\Theta(n \log n)$; it runs in $\Theta(\log n)$ additional space with modest exchange counts, and has excellent caching behaviour. Its worst case number of comparisons, however, is $\Theta(n^2)$. These facts remain essentially the same for the many deterministic variants of the algorithm that have been proposed (such as the best-of-three variant [Hoa62, Sin69] which performs $\frac{12}{7} n \ln n = 1.188n \log n$ comparisons on the average, and the quasi-best-of-nine variant [BM93] which empirically seems to perform $1.094n \log n$ comparisons on the average).

Classical mergesort performs $n \log n - n + 1$ comparisons and $\Theta(n \log n)$ operations in the worst case and exhibits good caching behaviour, but is not in-place, requiring $\Theta(n)$ additional space. In-place mergesorts with $\Theta(1)$ additional space have been achieved [Rei92, KPT96] with the same bounds but their complex index manipulations slow them down in practice.

Heapsort, due to Williams and Floyd [Wil64, Flo64], is the only practical in-place sorting algorithm known that performs $\Theta(n \log n)$ operations in the worst case. Bottom-up heapsort [Weg93], originally due to Floyd, performs $n \log n + O(n)$ comparisons on the average and $\frac{3}{2} n \log n + O(n)$ comparisons in the worst case, with $\Theta(1)$ additional space; weak-heapsort [Dut93, EW00] performs $n \log n + 1.1n$ comparisons in the worst case while using n additional bits. The exchange counts are also modest. Nonetheless, its average case

[2] $\mathrm{qlog}\,n \stackrel{\mathrm{def}}{=} \lfloor \log \frac{3}{4} n \rfloor + \log \frac{4}{3} + \frac{2}{n}(n - \frac{4}{3} 2^{\lfloor \log \frac{3}{4} n \rfloor})$, i.e., the approximation to the log function with equality roughly when $n = \frac{4}{3} 2^k$, for integers $k \geq 0$.

behaviour is not competitive with that of quicksort; it does not exhibit substantial locality of reference and deteriorates due to caching effects [LL97].

2 The Partition Sort

We present the partition sort as a one-parameter family of algorithms, with parameter $\gamma > 1$.

The sort of an array of n items begins with a recursive sort of the first $\left\lfloor \frac{n}{\gamma} \right\rfloor$ items, forming a subarray S of sorted items.

The heart of the algorithm is a *partition sort completion procedure* that completes the sort of the following type of partially sorted array. The array consists of two adjacent subarrays S and U: The portion to the left, S, is sorted; the remainder, U, is unsorted; $s \stackrel{\text{def}}{=} |S|$ and $u \stackrel{\text{def}}{=} |U|$. For a call (S, U), sort completion proceeds in two steps:

Multiway Partitioning: The items of U are partitioned into $s+1$ *buckets*, U_0, U_1, $\cdots$, U_s, defined by consecutive items of the sorted subarray $S = (s_1, s_2, \cdots, s_s)$, where $s_1 < s_2 \cdots < s_s$. U_k contains the items in U lying between s_k and s_{k+1} (we let $s_0 \stackrel{\text{def}}{=} -\infty$ and $s_{s+1} \stackrel{\text{def}}{=} +\infty$). Thus the items of S act as pivots in a multiway partitioning of U. We describe the implementation of this partitioning below.

Sorting of Buckets: Each bucket U_k, $0 \leq k \leq s$, is sorted by insertion sort. In order to bound the worst case operation count, if $|U_k| > c \log n$ (where c is a suitable constant) U_k is sorted recursively; in this case we say U_k is *large*, otherwise we say it is *small*.

The multiway partitioning may be thought of as performing, for each item in U, a binary search over the sorted subarray S. Nevertheless, so as to implement it in-place and to minimise the number of exchanges performed, we use a recursive multiway partitioning procedure, essentially due to Chen [Che02], described below for a call (S, U).

First the unsorted subarray U is partitioned about the median x of S; this creates the ordering $S_L\ x\ S_R\ U_L\ U_R$, with $S_L < x < S_R$ and $U_L < x < U_R$[3]. Next the blocks $\{x\} \cup S_R$ and U_L are swapped (as described in the next paragraph), preserving the order of $\{x\} \cup S_R$ but not necessarily of U_L, yielding the ordering $S_L\ U_L'\ x\ S_R\ U_R$ with $S_L \cup U_L' < x < S_R \cup U_R$. Then the subarrays (S_L, U_L') and (S_R, U_R) are partitioned recursively. The base case for the recursion arises when the sorted subarray S is empty; then the unsorted subarray U forms one of the sought buckets.

The partitioning of U about x may be done using any of the partitioning routines developed for quicksort. A simple way of swapping blocks $\{x\} \cup S_R$ and U_L is to walk through items in $\{x\} \cup S_R$ from right to left, swapping each item a thus encountered with the item in a's destination position.

It is readily seen that each item $u \in U$ performs exactly the same comparisons as in a binary search.

By solving the smaller subproblem first, and eliminating the remaining tail recursion, the additional space needed may be limited to $\Theta(\log n)$ in the worst case.

Remark 1 *The case $\gamma = \sqrt{n}$ corresponds to an in-place sequential implementation of parallel quicksort [Rei85].*

2.1 Worst Case Analysis of Operations

We analyse the algorithm for the case $\gamma = 2$ in detail, focusing on the comparisons initially.

Observe that the number of comparisons occurring during the insertion sorts is bounded by $O(n \log n)$. Thus, it suffices to analyse the remaining comparisons.

Let $B(s, u)$ be the worst case number of non-insertion sort comparisons for sort completions on inputs of size (s, u), and let $C(n)$ be the worst case number of non-insertion sort comparisons for the partition sort on inputs of size n.

[3] $a < B$ means that for all $b \in B$, $a < b$. $B < c$ is defined analogously.

Each item $u \in U$ performs at most $\lceil \log(s+1) \rceil$ comparisons during the multiway partitioning. It remains to bound the comparisons occuring during the sorting of buckets. If $l_k \stackrel{\text{def}}{=} |U_k|$, then $\sum_{k=0}^{s} l_k = u$, and

$$B(s,u) \leq \max_{l_k} \left(u \lceil \log(s+1) \rceil + \sum_{k=0}^{s} C(l_k) \right) = u \lceil \log(s+1) \rceil + C(u)$$

where we may have overestimated since a term $C(l_k)$ is needed only if $l_k \geq c \log n$.

Clearly, $C(1) = 0$ and, for $n > 1$:

$$C(n) \leq B\left(\left\lfloor \frac{n}{2} \right\rfloor, \left\lceil \frac{n}{2} \right\rceil \right) + C\left(\left\lfloor \frac{n}{2} \right\rfloor \right) \leq \left\lceil \frac{n}{2} \right\rceil \left\lceil \log\left(\left\lfloor \frac{n}{2} \right\rfloor + 1 \right) \right\rceil + C\left(\left\lceil \frac{n}{2} \right\rceil \right) + C\left(\left\lfloor \frac{n}{2} \right\rfloor \right)$$

Thus, $C(n) = \Theta(n \log^2 n)$, since the bound is tight on an input in sorted order, and we have shown:

Lemma 1 *The partition sort, with* $\gamma = 2$, *performs* $\Theta(n \log^2 n)$ *comparisons in the worst case.*

For larger γ, including the operation count, we can show:

Theorem 1 *The partition sort performs* $\Theta(\gamma\, n \log^2 \frac{n}{\gamma})$ *operations in the worst case.*

2.2 Average Case Analysis of Operations

Again, we analyse the case $\gamma = 2$ in detail; as before, the comparison count dominates the operation count, and thus it suffices to analyse comparisons.

For simplicity, we assume initially that the algorithm sorts all buckets large and small using insertion sort. We shall ensure, with a suitable choice of c, that precisely because we forsake recursive calls to partition sort, this introduces an inefficiency in the sorting of large buckets on the average, and hence that the average case bound that we shall obtain is a valid (if slightly weak) bound for the original algorithm too.

Let $B(s,u)$ be the average case number of comparisons for sort completions on inputs of size (s,u), and $C(n)$, the average case number of comparisons for the partition sort on inputs of size n. Let $I(l)$ denote the average case number of comparisons for sorting a bucket of size l. It is easy to show:

Lemma 2 *If* $s+1 = 2^i + h$ *with* $0 \leq h < 2^i$, *then the partitioning for an input of size* (s,u) *performs* $u\left(i + \frac{2h}{s+1}\right)$ *comparisons on the average.*

We now bound $B(s,u)$ by overestimating the number of comparisons required, on the average, for the sorting of buckets; with $s+1 = 2^i + h$, where $0 \leq h < 2^i$, we have:

$$B(s,u) = u\left(i + \frac{2h}{s+1}\right) + \sum_{\text{item } a} \sum_{l=2}^{u} \Pr[a \text{ is in } U \text{ and its bucket has size } l]\ \frac{I(l)}{l}$$

The probability for an arbitrary item a in $S \cup U$ to be in U and to be in a bucket of size l, may be overestimated as follows. For an arbitrary item a, the end points of a bucket of size l (which are in S) may be chosen in at most l distinct ways, and given such a pair of end points, the probability that they form a bucket (which is the same as saying that the remaining $s-2$ items in S are chosen from outside this range) is exactly:

$$\binom{s+u-l-2}{s-2} \Big/ \binom{s+u}{s}$$

unless a is among the least or greatest l items of $S \cup U$; to avoid anomalies, it suffices to pretend the buckets at the two extremes are combined to form a single bucket; then the same probability applies to every item, and clearly we have only overestimated the cost of the algorithm. Hence:

$$B(s,u) \leq u\left(i + \frac{2h}{s+1}\right) + (s+u) \sum_{l=2}^{u} I(l) \binom{s+u-l-2}{s-2} \Big/ \binom{s+u}{s}$$

We may now obtain a bound on $C(n)$. For $n > 1$, we have:

$$C(n) \le B\left(\left\lfloor \frac{n}{2} \right\rfloor, \left\lceil \frac{n}{2} \right\rceil\right) + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \le \left\lceil \frac{n}{2} \right\rceil \left(i + \frac{2h}{\left\lfloor \frac{n}{2} \right\rfloor + 1}\right) + n \left[\sum_{l=2}^{\left\lceil \frac{n}{2} \right\rceil} \frac{\binom{n-l-2}{\left\lfloor \frac{n}{2} \right\rfloor - 2}}{\binom{n}{\left\lfloor \frac{n}{2} \right\rfloor}} I(l)\right] + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

where $\left\lfloor \frac{n}{2} \right\rfloor + 1 = 2^i + h$ and $0 \le h < 2^i$.

Clearly, $I(l) \le \binom{l}{2}$.

Lemma 3

$$\sum_{l=2}^{\frac{n+1}{2}} \frac{\binom{n-l-2}{\frac{n-5}{2}}}{\binom{n}{\frac{n-1}{2}}} \binom{l}{2} \le \frac{1}{2} + \frac{12}{n} + O\left(\frac{1}{n^2}\right)$$

Substituting in the equation for $C(n)$ yields:

Lemma 4 *If all input orderings are equally likely, the partition sort, with* $\gamma = 2$*, performs at most* $n \operatorname{plog}(n+1) - n + O(\log n)$ *comparisons on the average.*

By bounding $I(l)$ more exactly, we can show:

Theorem 2 *If all input orderings are equally likely, the partition sort, with* $\gamma = 2$*, performs at most* $n \operatorname{plog}(n+1) - \left(\frac{1}{2} + \ln 2\right) n + O(\log n) = n \operatorname{plog}(n+1) - 1.193n + O(\log n)$ *comparisons and* $\Theta(n \log n)$ *operations on the average.*

3 Chen's Algorithm

We slightly generalise Chen's algorithm to a three-parameter family of algorithms, with parameters γ, λ, and μ, described below. Typically, $\gamma \ge 2$ and a power of 2, $\lambda \ge 2$ and a power of 2, $\lambda \le \mu$, and $\gamma \le \mu$; Chen [Che02] had considered a one-parameter family with $\gamma = \lambda$ and $\mu = \lambda(\lambda - 1)$, for arbitrary $\lambda \ge 2$.

As with the partition sorts, the sort begins with a recursive sort of the first $\left\lfloor \frac{n}{\gamma} \right\rfloor$ items. The heart of the algorithm is provided by *Chen's sort completion procedure*, for completing the sort of a partially sorted array (S, U); as before, S is sorted and U is unsorted. This recursive procedure is similar to that used in the partition sort, except for proceeding differently when S is small relative to U, and has the following form:

Case $\mu(s+1) - 1 \ge (s+u)$: Thus, $(\mu - 1)(s+1) \ge u$; we say that such an input has the *balanced property.*

First, the unsorted subarray U is partitioned about the median x of S, resulting in blocks U_L and U_R; next, the blocks $\{x\} \cup S_R$ and U_L are swapped, preserving the order of items in $\{x\} \cup S_R$, but not necessarily of those in U_L, yielding the ordering $S_L\ U'_L\ x\ S_R\ U_R$ with $S_L \cup U'_L < x < S_R \cup U_R$; finally the subarrays (S_L, U'_L) and (S_R, U_R) are sorted recursively.

Case $\mu(s+1) - 1 < (s+u)$: Thus, $(\mu - 1)(s+1) < u$; we call this case *expansion*, and in particular for $\lambda = 2$, *doubling.*

First, the sorted subarray S is expanded as follows: Let S' be the leftmost subarray of size $\lceil \lambda(s+1) - 1 \rceil$. The subarray S' is sorted recursively, with S as the sorted subarray; then, the sort of the whole array continues recursively, with S' as the sorted subarray.

Chen's description [Che02] of his algorithm can be viewed as bottom-up; (s, u) is set to $(1, n-1)$ initially, which is analogous to setting $\gamma = \lambda$; we have replaced his single parameter with three.

3.1 Average Case Analysis of Comparisons

We prove that on average the partition sort with parameter γ performs at least as many comparisons as (*dominates*) Chen's algorithm with parameters (λ, μ, γ). We analyse the case $\lambda = 2$ in detail.

For purposes of the analysis, we introduce another sorting algorithm, which we call the *variant partition sort* (to contrast it with the original partition sort, henceforth qualified as *basic* to emphasise the distinction). We prove two claims:

Claim 1 *The basic partition sort dominates the variant partition sort.*

Claim 2 *If Claim 1 holds, then the basic partition sort dominates Chen's algorithm.*

It is helpful to view the basic partition sort as follows, when its input is an unbalanced problem instance (S, U) with $(\mu - 1)(s + 1) < u$ (where $s = |S|$ and $u = |U|$); we name the s items in S type A items, the immediately following $s + 1$ items in U type B items, and the remaining $t \stackrel{\text{def}}{=} u - (s + 1)$ items in U type C items; the basic partition sort then proceeds as follows:

P1: It places each of the u items of U in one of the $s + 1$ buckets delimited by items in S; we call these the A buckets. Each placement takes $\log(s + 1)$ comparisons.

P2: It sorts each of the $s + 1$ A buckets by insertion sort.

The variant partition sort, which mimics Chen's algorithm to a certain extent, proceeds as follows:

V1: It places each of the $s + 1$ type B items of U in one of the $s + 1$ A buckets delimited by items in S. Each placement takes $\log(s + 1)$ comparisons.

V2: It sorts each of the $s + 1$ A buckets by insertion sort.

V3: It places each of the t type C items of U in one of the $2s + 2$ buckets delimited by items in $A \cup B$; we call these the AB buckets. Each placement takes $1 + \log(s + 1)$ comparisons.

V4: It sorts each of the $2s + 2$ AB buckets by insertion sort.

Lemma 5 *For* $\lambda = 2$*, Claim 2 holds.*

Proof: We proceed by induction on $s + u$. Clearly, for $s + u = 1$, the result holds.

For $s + u > 1$, if $(\mu - 1)(s + 1) \geq u$, the initial call in Chen's algorithm is a non-doubling call, and hence the same as in the partition sort; for this case the result follows by induction applied to the recursive calls.

It remains to consider the unbalanced case with $(\mu - 1)(s + 1) < u$. Here the initial call in Chen's algorithm is a doubling call. In the variant partition sort, the same doubling call occurs, but thereafter it performs no more doubling calls, i.e., in its recursive calls on subproblems $(\tilde{s}, \tilde{u})$ it acts in the same way as the basic partition sort on input $(\tilde{s}, \tilde{u})$. We consider the two top level recursive calls generated by the doubling call $(s, s + 1)$, namely $(\frac{s-1}{2}, u_1)$ and $(\frac{s-1}{2}, s + 1 - u_1)$, and the call after the doubling completes, $(2s + 1, u - s - 1)$. For each value of u_1, all possible orderings of the items are equally likely, given that the initial (s, u) problem was distributed uniformly at random. Consequently, the inductive hypothesis can be applied to these three inner calls, showing that the basic partition sort dominates Chen's algorithm in each of these inner calls. Consequently, the variant partition sort dominates Chen's algorithm for the original (s, u) call. Given our assumption that the basic partition sort dominates the variant partition sort, this proves the inductive step. **Q. E. D.**

Lemma 6 *For* $\lambda = 2$*, Claim 1 holds.*

Proof: To compare the basic partition sort with its variant conveniently, we view the execution of the basic partition sort in the following alternative way; this form has exactly the same comparison cost as the previous description.

P1′: It places each of the $s+1$ type B items of U in one of the $s+1$ A buckets delimited by items in S. (Each placement takes $\log(s+1)$ comparisons.) This is identical to step V1.

P2′: It sorts each of the $s+1$ A buckets (containing type B items) by insertion sort. This is identical to step V2.

P3′: It places each of the t type C items of U at the right end of one of the $s+1$ A buckets in $A \cup B$ (which already contain type B items in sorted order). (Each placement is done by a binary search over the A items, taking $\log(s+1)$ comparisons.)

P4′: For each of the $s+1$ A buckets, it places the type C items in their correct position by insertion sort (at the start of this step, each such A bucket contains both type B and type C items, with type B items in sorted order, and type C items at the right end).

It remains to determine the relative costs of steps V3 and V4 and of steps P3′ and P4′. Step V3 performs one more comparison for each type C item, for a total excess of t comparisons (this is clear for $s = 2^k - 1$, and needs a straightforward calculation otherwise). In step V4, the only comparisons are between type C items within the same AB bucket, whereas in step P4′ there are, in addition, comparisons between type C items in distinct AB buckets (but in the same A bucket), and also comparisons between type B and type C items in the same A bucket, called BC comparisons. We confine our attention to BC comparisons and show that there are at least t such comparisons, on the average; it would then follow that on average the partition sort dominates its variant.

If it were the case that one type B item ($s+1$ in number) went into each of the $s+1$ A buckets, resulting in an *equipartition*, it is evident that the number of BC comparisons would be exactly t. We now argue that this is in fact a minimum, when considering comparisons on the average.

Given the AB sequence with equipartition (i.e., with one B item in each A bucket) consider perturbing it to a different sequence. For any such non-trivial perturbation, there is at least one A bucket which holds a group of $r+1 > 1$ type B items (and which contains $r+2$ AB buckets).

Consider an A bucket with multiple type B items in the perturbed sequence. The r excess type B items must have been drawn from (originally) singleton A buckets (which originally contained two AB buckets) and thus there are r empty A buckets (which now contain only one AB bucket). An arbitrary type C item could go into any of the $2s+2$ AB buckets with equal probability. The number of BC comparisons that this type C item undergoes, on the average, increases by:

$$\frac{1}{2s+2}\left[-2.(r+1) + (r+2).\left(\frac{r+3}{2} - \frac{1}{r+2}\right)\right] > 0$$

for $r \geq 1$ as a consequence of the perturbation.

Finally, we note that in the perturbed sequence, for each A bucket with $r+1 > 1$ type B items, the number of BC comparisons an arbitrary type C item undergoes increases, on the average, by the above quantity.

Thus, the equipartition configuration minimises the number of BC comparisons, and consequently, the basic partition sort dominates its variant.

Q. E. D.

Setting $\gamma = 2$ to exhibit a bound, from dominance and Theorem 2 we have:

Theorem 3 *If all input orderings are equally likely, Chen's algorithm, with* $(\lambda, \gamma) = (2, 2)$*, performs at most* $n \operatorname{plog} n - \left(\frac{1}{2} + \ln 2\right) n + O(\log n)$ *comparisons on the average, independently of* μ.

3.2 Operation Counts

Theorem 4 *Chen's algorithm performs* $\Theta(\frac{\mu}{\lambda} \log \lambda \; n \log n)$ *comparisons in the worst case, independently of* γ.

Theorem 5 *Chen's algorithm performs* $\Theta(n\frac{\log^2 n}{\log^2 \frac{\mu}{\lambda}})$ *exchanges in the worst case.*

Theorem 6 *If all input orderings are equally likely, Chen's algorithm with* $\lambda(1+\epsilon) \leq \mu$ *performs* $\Theta(\gamma\, n \log n)$ *exchanges on the average.*

4 Empirical Studies

We implemented and compared the performance of quicksort, the partition sort and Chen's algorithm. We measured the running times T_n and counted comparisons C_n and data moves M_n, for various input sizes n.

We implemented quicksort in two different ways. The first largely follows Sedgewick [Sed78]; it uses a best-of-three strategy for selecting the pivot [Hoa62, Sin69], and sorts small subproblems (of size less than the insertion cutover) using insertion sort, but unlike Sedgewick, performs the insertion sorts as they arise (locally), rather than in one final pass (globally). Our experiments showed that for large input sizes, the local implementation yielded a speedup. The second implementation is similar except that it uses Tukey's 'ninther', the median of the medians of three samples (each of three items), as the pivot; this quasi-best-of-nine version due to Bentley and McIlroy [BM93] was observed to be about 3% faster than the best-of-three version.

For the partition sort and Chen's algorithm, the block swaps (of blocks $\{x\} \cup S_R$ and U_L) are performed using Chen's optimised implementation [Che96].

Best results for the partition sort were obtained with γ about 128. The average size of the bucket is then quite large and we found it to be significantly more efficient to sort them with quicksort. Moderate variations of γ had little impact on performance.

Again, with Chen's algorithm, we added a cutover to quicksort on moderate sized subproblems.[4] Our best performance arose with $\lambda = 2$, $\mu = 128$, and $\gamma = 128$, and a quicksort cutover of about 500. Again, moderate variation of μ and γ had little effect on the performance.

We compared top-down and bottom-up drivers in our experiments, and found that top-down drivers tend to have smoother behaviour. We have therefore chosen to use top-down drivers in our implementations.

For our algorithms, we resort to standard hand optimisations, such as eliminating the tail recursions by branching (and other recursions by explicit stack management); we also inline the code for internal procedures.

We ran each algorithm on a variety of problem sizes. Each algorithm, for each problem size, was run on the same collection of 25 randomly generated permutations of integers (drawn from the uniform distribution). Running times for the best choices of parameters are shown in the table below.

[4] Without this cutover we were unable to duplicate Chen's experimental results [Che02]. He reported a 5-10% speedup compared to best-of-three quicksort on inputs of size up to 50,000.

Running Times $\overline{T_n} \pm \sigma(T_n)$ (μs)				
n	Partition	Chen	Quasi-best-of-9	Best-of-3
20000	4906 ± 110	4878 ± 78	4840 ± 190	4868 ± 182
25000	6306 ± 107	6421 ± 185	6134 ± 183	6268 ± 183
30000	7652 ± 78	7863 ± 150	7493 ± 241	7805 ± 201
35000	9124 ± 200	9155 ± 217	8969 ± 315	9215 ± 252
40000	10577 ± 149	10689 ± 221	10406 ± 235	10601 ± 284
200000	65453 ± 764	67048 ± 7861	65294 ± 1467	87997 ± 6412
250000	84526 ± 1355	84551 ± 1269	84110 ± 914	86294 ± 2199
300000	103683 ± 354	104319 ± 3631	102909 ± 1264	105514 ± 2124
350000	123258 ± 1457	124211 ± 2542	122788 ± 1389	127300 ± 2900
400000	143115 ± 1024	144246 ± 2007	143029 ± 1622	147475 ± 2328
2000000	858194 ± 3038	862604 ± 5364	893321 ± 2779	897493 ± 6893
2500000	1093029 ± 2805	1100885 ± 4665	1107729 ± 12688	1148407 ± 10356
3000000	1344032 ± 9178	1348160 ± 9513	1360892 ± 2680	1404965 ± 11242
3500000	1583731 ± 5295	1596032 ± 8990	1602471 ± 9326	1663999 ± 9192
4000000	1833468 ± 4975	1845352 ± 9949	1854723 ± 12678	1921886 ± 2685

In our experiments, even for trials repeated only a few times, the deviations are orders of magnitude smaller than the average. This is most striking for the operation counts (not shown) where, with trials repeated 25 times, the deviations are typically below 0.1% of the averages for the new algorithms, and below 2% for the quicksorts.

The experiments were conducted on a Sun UltraSPARC-IIe processor with CPU speed 550 MHz and Cache size 512 KB. The programs, written in C, were run under Sun OS (kernel version Generic_112233-08) with `gcc` (version 3.3.3) compilation at `-O3` level. Broadly similar results were obtained on a Pentium running GNU/Linux.

5 Acknowledgements

We thank J. Ian Munro, Alan Siegel, and an anonymous referee, for their helpful comments.

Bibliography

[Ben00] Jon L. Bentley. *Programming Pearls*. Addison-Wesley, Reading, MA, second edition, 2000.

[BM93] Jon L. Bentley and M. Douglas McIlroy. Engineering a sort function. *Software Practice and Experience*, 23(11):1249–1265, November 1993.

[Che96] Jing-Chao Chen. Proportion split sort. *Nordic Journal of Computing*, 3(3):271–279, Fall 1996.

[Che02] Jing-Chao Chen. Proportion extend sort. *SIAM Journal on Computing*, 31(1):323–330, February 2002.

[Dut93] Ronald D. Dutton. Weak-heapsort. *BIT*, 33(3):372–381, 1993.

[EW00] Stefan Edelkamp and Ingo Wegener. On the performance of weak-heapsort. *Lecture Notes in Computer Science*, 1770:254–266, 2000.

[FG03] Gianni Franceschini and Viliam Geffert. An in-place sorting with $O(n \log n)$ comparisons and $O(n)$ moves. In *Forty fourth IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 242–250, Cambridge, Massachusetts, 11–14 October 2003.

[FJ59] Lester R. Ford, Jr. and Selmer M. Johnson. A tournament problem. *The American Mathematical Monthly*, 66(5):387–389, May 1959.

[Flo64] Robert W. Floyd. ACM Algorithm 245: Treesort 3. *Communications of the ACM*, 7(12):701, December 1964.

[Hoa61] C. A. R. Hoare. ACM Algorithm 63: Partition, ACM Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321–322, July 1961.

[Hoa62] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, April 1962.

[Knu73] Donald E. Knuth [Ervin]. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts, USA, 1973.

[KPT96] Jyrki Katajainen, Tomi Pasanen, and Jukka Teuhola. Practical in-place mergesort. *Nordic Journal of Computing*, 3(1):27–40, Spring 1996.

[LL97] Anthony LaMarca and Richard E. Ladner. The influence of caches on the performance of sorting. In *Eighth ACM-SIAM Symposium on Discrete Algorithms*, pages 370–379, New Orleans, Louisiana, 5–7 January 1997.

[Mus97] David R. Musser. Introspective sorting and selection algorithms. *Software Practice and Experience*, 27(8):983–993, August 1997.

[Rei85] Rüdiger Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM Journal on Computing*, 14(2):396–409, May 1985.

[Rei92] Klaus Reinhardt. Sorting in-place with a worst case complexity of $n \log n - 1.3n + o(\log n)$ comparisons and $\epsilon n \log n + o(1)$ transports. In *Third International Symposium Algorithms and Computation*, pages 489–498, Nagoya, Japan, 16–18 December 1992.

[Sed78] Robert Sedgewick. Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857, October 1978.

[Sin69] R. C. Singleton. An efficient algorithm for sorting with minimal storage. *Communications of the ACM*, 12(3):185–187, March 1969.

[Weg93] Ingo Wegener. Bottom-up heapsort, a new variant of heapsort, beating, on an average, quicksort (if n is not very small). *Theoretical Computer Science*, 118(1):81–98, 13 September 1993.

[Wil64] J. W. J. Williams. ACM Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, June 1964.

A Addendum to Section 2

A.1 Proof of Lemma 2

An arbitrary item a in U could enter any of the $s + 1$ buckets taking either i or $i + 1$ comparisons. For $2h$ of these buckets, it will require $i + 1$ comparisons; for the remaining $s + 1 - 2h$ buckets it will need only i comparisons.

A.2 Proof of Lemma 3

We begin with the case where n is odd.

$$\begin{aligned}
\sum_{l=2}^{\frac{n+1}{2}} \frac{\binom{n-l-2}{\frac{n-5}{2}}}{\binom{n}{\frac{n-1}{2}}} \binom{l}{2} &= \sum_{l=2}^{\frac{n+1}{2}} \frac{\frac{n-1}{2}(\frac{n-1}{2}-1)(\frac{n+1}{2})(\frac{n+1}{2}-1)\cdots(\frac{n+1}{2}-l+1)}{n(n-1)(n-2)(n-3)\cdots(n-l-1)} \binom{l}{2} \\
&= \sum_{l=2}^{\frac{n+1}{2}} \frac{\frac{n-1}{2}(\frac{n-1}{2}-1)}{n(n-1)} \frac{(\frac{n+1}{2})(\frac{n+1}{2}-1)\cdots(\frac{n+1}{2}-l+1)}{(n-2)((n-2)-1)\cdots((n-2)-l+1)} \binom{l}{2} \\
&\leq \sum_{l=2}^{\frac{n+1}{2}} \frac{\frac{n-1}{2}(\frac{n-1}{2}-1)}{n(n-1)} \left[\frac{n+1}{2(n-2)}\right]^l \binom{l}{2} \\
&\leq \left(\frac{1}{2}-\frac{1}{2n}\right)\left(\frac{1}{2}-\frac{1}{n-1}\right) \frac{1}{2} \left[\sum_{l=1}^{\infty} l^2 r^l - \sum_{l=1}^{\infty} l r^l\right] \qquad \text{(where } r = \tfrac{n+1}{2(n-2)}\text{)} \\
&= \left(\frac{1}{2}-\frac{1}{2n}\right)\left(\frac{1}{2}-\frac{1}{n-1}\right) \frac{1}{2} \frac{2r^2}{(1-r)^3}
\end{aligned}$$

Since

$$r = \frac{n+1}{2(n-2)} = \frac{1}{2} + \frac{3}{2(n-2)} = \frac{1}{2} + \frac{3}{2n} + O\left(\frac{1}{n^2}\right)$$

we can see that the above is bounded by $\frac{1}{2} + \frac{12}{n} + O\left(\frac{1}{n^2}\right)$.

The other case is similar.

A.3 Proof of Lemma 4

We have, for $n > 1$:

$$C(n) \leq \left\lceil \frac{n}{2} \right\rceil i + 2h + \frac{1}{2}n + O(1) + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

where $\lfloor \frac{n}{2} \rfloor + 1 = 2^i + h$ with $0 \leq h < 2^i$.

Let the binary representation of n be $b_{i+1}b_i\cdots b_1b_0$. Since $C(1)=0$, we see that:

$$\begin{aligned}
C(n) &\le \left(\left\lceil\frac{n}{2}\right\rceil i + \left\lceil\frac{1}{2}\left\lfloor\frac{n}{2}\right\rfloor\right\rceil(i-1) + \left\lceil\frac{1}{2}\left\lfloor\frac{n}{4}\right\rfloor\right\rceil(i-2)+\cdots\right) + 4h + n + O(\log n)\\
&= \left[\frac{n}{2}+\frac{b_0}{2}\right] i + \left[\left(\frac{n}{4}-\frac{b_0}{4}\right)+\frac{b_1}{2}\right](i-1)\\
&\quad + \left[\left(\frac{n}{8}-\frac{b_0}{8}-\frac{b_1}{4}\right)+\frac{b_2}{2}\right](i-2)+\cdots\\
&\quad + 4h + n + O(\log n)\\
&= \left(\frac{n}{2}i+\frac{n}{4}(i-1)+\frac{n}{8}(i-2)+\cdots\right)\\
&\quad + b_0\left[\frac{i}{2}-\left(\frac{i-1}{4}+\frac{i-2}{8}+\cdots\right)\right]\\
&\quad + b_1\left[\frac{i-1}{2}-\left(\frac{i-2}{4}+\frac{i-3}{8}+\cdots\right)\right]\\
&\quad \cdots\\
&\quad + 4h + n + O(\log n)\\
&= n\left\lfloor\log\left(\frac{n+1}{2}\right)\right\rfloor - n + O(b_0+b_1+\cdots+b_{i+1}) + 4h + n + O(\log n)\\
&\le n\lfloor\log(n+1)\rfloor - n + 4h + O(\log n)\\
&\le n\,\mathrm{plog}\,(n+1) - n + O(\log n)
\end{aligned}$$

A.4 An Improved Average Case Bound

The following elementary observations regarding standard insertion sort are easily verified.

1. The kth item being inserted performs:

$$\frac{1}{k}\left[(k-1)+\sum_{i=1}^{k-1} i\right] = \frac{k+1}{2}-\frac{1}{k}$$

comparisons on the average, where $1\le k\le l$, provided that sentinels are not used.

2. Sorting l items consumes:

$$\frac{l+3}{4}-\frac{H_l}{l}$$

comparisons per item, on the average.

Using arguments similar to those used before, we obtain that, for $n>1$:

$$C(n)\le\left\lceil\frac{n}{2}\right\rceil\left(i+\frac{2h}{\lfloor\frac{n}{2}\rfloor+1}\right)+n\left[\sum_{l=2}^{\lceil\frac{n}{2}\rceil}\frac{\binom{n-l-2}{\lfloor\frac{n}{2}\rfloor-2}}{\binom{n}{\lfloor\frac{n}{2}\rfloor}}\left(\frac{l(l+3)}{4}-H_l\right)\right]+C\left(\left\lfloor\frac{n}{2}\right\rfloor\right)$$

where $\lfloor\frac{n}{2}\rfloor+1=2^i+h$ and $0\le h<2^i$, and that:

$$\sum_{l=2}^{\frac{n+1}{2}}\frac{\binom{n-l-2}{\lfloor\frac{n}{2}\rfloor-2}}{\binom{n}{\lfloor\frac{n}{2}\rfloor}}\left(\frac{l(l+3)}{4}-H_l\right)\le\frac{5}{8}-\frac{1}{4}\sum_{l=2}^{\infty}H_l\left(\frac{1}{2}\right)^l+O\left(\frac{1}{n}\right)$$

To justify the use of bound (2) above for sorting l items in recursive calls to partition sort, we need that, for some constant c:

$$l \log l - l + O(\log l) \leq I(l)$$

for l such that $c \log n < l \leq \lceil \frac{n}{2} \rceil$. Clearly this holds for a large enough constant $c > 0$.

Lemma 7

$$\sum_{l=2}^{\infty} H_l \left(\frac{1}{2}\right)^l = 2\ln 2 - \frac{1}{2}$$

Proof:

$$\begin{aligned}
\sum_{i=2}^{\infty} H_i \left(\frac{1}{2}\right)^i &= \sum_{i=2}^{\infty} \frac{1}{2^i} \sum_{j=1}^{i} \frac{1}{j} \\
&= \sum_{i=2}^{\infty} \frac{1}{2^i} \left[1 + \sum_{j=2}^{i} \frac{1}{j}\right] \\
&= \sum_{i=2}^{\infty} \frac{1}{2^i} + \sum_{i=2}^{\infty} \sum_{j=2}^{i} \frac{1}{j2^i} \\
&= \frac{1}{2} + \sum_{j=2}^{\infty} \frac{1}{j} \sum_{i=j}^{\infty} \frac{1}{2^i} \\
&= \frac{1}{2} + 2 \sum_{j=2}^{\infty} \frac{1}{j} \left(\frac{1}{2}\right)^j \\
&= \frac{1}{2} + 2\left[-\ln\left(1 - \frac{1}{2}\right) - \frac{1}{2}\right] \\
&= 2\ln 2 - \frac{1}{2}
\end{aligned}$$

Q. E. D.

This yields, for $n > 1$:

$$C(n) \leq \left\lceil \frac{n}{2} \right\rceil \left(i + \frac{2h}{\lfloor \frac{n}{2} \rfloor + 1}\right) + \left(\frac{3}{4} - \frac{1}{2}\ln 2\right) n + C\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

Since $C(1) = 0$, we have Theorem 2.

For larger γ, one may, with a similar argument, obtain a bound of $n \log n + O(\gamma n)$ comparisons and a comparable bound on operations, on the average. We note that, for small increases in γ, while the number of comparisons increases, the number of exchanges decreases. Our experiments indicate that the best tradeoff occurs at $\gamma > 2$. We did not analyse this further, as other factors such as caching effects and the relative costs of different operations could make meaningful results somewhat machine dependent.

A.5 A Bottom-up Variant of the Partition Sort

We consider a bottom-up variant of the partition sort which is useful for analysing Chen's algorithm: in turn we perform an insertion sort on the first $\gamma - 1$ items, followed by a series of sort completions of the form $(\gamma^k - 1, \gamma^{k+1} - \gamma^k)$, for $1 \leq k < \lfloor \log_\gamma(n+1) \rfloor$, finishing with a sort completion of the form $(\hat{s}, \hat{u}) = (\gamma^{\lfloor \log_\gamma(n+1) \rfloor} - 1, n - \gamma^{\lfloor \log_\gamma(n+1) \rfloor} + 1)$ on the whole array; thus initial calls are generated with sizes which are essentially powers of the parameter γ.

We can show that the bounds derived previously apply here too.

A.6 Analysis of the Bottom-up Variant

We exhibit this only for the average case comparison count, with $\gamma = 2$, since the other bounds entail similar arguments.

The cost of sorting the array of size $\hat{s}$ is analysed as before, since it is the same algorithm viewed bottom-up (rather than top-down), apart from the cost of the initial insertion sort. This yields a comparison count of $\hat{s}\log(\hat{s}+1) - 1.193\hat{s} + O(\log\hat{s})$ on the average.

The final sort completion requires at most $\hat{u}\log(\hat{s}+1)$ comparisons for the multiway partitioning step, and $2\left(\frac{3}{4} - \frac{1}{2}\ln 2\right)\hat{u} = O(n)$ comparisons for the sorting of buckets, on the average.

Thus, the overall comparison count is $n\,\mathrm{p}\log n - 1.193\hat{s} + O(\log n)$ on the average.

A.7 Notes

Remark 2 *The strategy of improving the average case behaviour of quicksort by choosing the median of a small sample as the pivot has some commonality with the multiway partitioning in the partition sort.*

Remark 3 *But for the fact that we sort large buckets recursively, the partition sort would perform* $\Theta(n^2)$ *operations in the worst case, independently of* γ*.*

Remark 4 *To detect substantially sorted inputs, which trigger worst case behaviour in partition sort, we might proceed as follows. During the initial sort of a set of size about* $\sqrt{n}$*, if the top level application of the multiway partitioning procedure yields one or more very large buckets, then we heuristically assume the input is bad. This would then cause a switch to an alternative sort for the buckets; perhaps a randomization of the set at hand, perhaps a switch to heapsort or insertion sort. A more cautious approach is to keep checking for undue bucket sizes in each application of the multiway partitioning procedure.*

Remark 5 *The method for detecting substantially sorted inputs is comparable to the optimisation to quicksort [Mus97], which improves the worst case number of comparisons to* $\Theta(n\log n)$*, while essentially preserving the average case comparison bound of quicksort; the optimisation consists of switching to heapsort for quicksort subproblems which are found to have been partitioned more than* $\Theta(\log n)$ *times. In either optimisation, we note that* $\Theta(n\log n)$ *operations are done before bad inputs are isolated.*

B Addendum to Section 3

B.1 Worst Case Analysis of Comparisons

We analyse the algorithm for the case $(\lambda,\mu,\gamma) = (2,2,2)$ in detail.

We observe that the algorithm performs a series of sort completions of the form $(s,u) = (2^k - 1, 2^k)$, with $1 \le k \le \lceil \log\frac{n+1}{2}\rceil$, in turn, followed by a final completion of the form $(2^{\lceil \log\frac{n+1}{2}\rceil} - 1, n - 2^{\lceil \log\frac{n+1}{2}\rceil} + 1)$. Each completion (except the first) uses the result of the preceding completion as its sorted subarray.

We now analyse the worst case comparison cost of Chen's sort completion.

Suppose there is a call (s,u) with $s \ge u-1$, i.e., a balanced input. (Observe that this includes all the top level calls mentioned in the previous paragraph). We wish to prove that after a constant number of subcalls, all subproblems $(\tilde{s},\tilde{u})$ generated at the fringe (of the call tree with (s,u) as the root) will regain the balanced property, namely $\tilde{s} \ge \tilde{u} - 1$.

Let (s_1,u_1) and (s_2,u_2) denote the two subcalls of (s,u). Note that $s_1 = s_2 = \frac{s-1}{2}$. We consider the case in which at least one of the subcalls has lost the balanced property (without loss of generality, the call (s_1,u_1), with $s_1 < u_1 - 1$). It follows that $s_2 \ge u_2 - 1$, i.e., that (s_1,u_1) is the only subcall to have lost this property.

The call (s_1,u_1) first generates a call of the form $(\frac{s-1}{2},\frac{s+1}{2})$ (which has the balanced property); this increases the sorted set to size $s_1' = 2s_1 + 1 = s$, and reduces the unsorted set to size $u_1' = u_1 - \frac{s+1}{2}$; now the call $(s_1',u_1') = (s, u_1 - \frac{s+1}{2})$ is generated.

The subcalls of the call (s_1', u_1') are of the form $(\frac{s-1}{2}, u_3)$ and $(\frac{s-1}{2}, u_4)$, with $u_3 + u_4 = u_1 - \frac{s+1}{2}$, and thus $\frac{s-1}{2} \geq u_3 - 1$ and $\frac{s-1}{2} \geq u_4 - 1$, showing the balanced property for the subcalls at the fringe of the call tree at this stage. Further, in all these subcalls, the size of the sorted subarray has essentially halved relative to the root.

We now prove, for balanced inputs of size (s, u), a bound on the worst case number of comparisons, $B(s, u)$. Clearly, $B(1, 1) = 1$, $B(1, 2) = 3$, and

$$
\begin{aligned}
B(s,u) &= \max \begin{cases} u + B\left(\frac{s-1}{2}, u_1\right) + B\left(\frac{s-1}{2}, u_2\right) & \text{if } s_1 \geq u_1 - 1 \\ u + u_1' + B\left(\frac{s-1}{2}, \frac{s+1}{2}\right) + B\left(\frac{s-1}{2}, u_2\right) & \\ \quad + B\left(\frac{s-1}{2}, u_3\right) + B\left(\frac{s-1}{2}, u_4\right) & \text{otherwise} \end{cases} \\
&\leq \frac{3}{2}u + B\left(\frac{s-1}{2}, \frac{s+1}{2}\right) + B\left(\frac{s-1}{2}, u_2\right) + B\left(\frac{s-1}{2}, u_3\right) + B\left(\frac{s-1}{2}, u_4\right)
\end{aligned}
$$

Since $\frac{s+1}{2} + u_2 + u_3 + u_4 = u$, it follows that $B(s, u) \leq \frac{3}{2} u \log(s + 1)$.

Let $C(n)$ be the worst case comparison cost for Chen's algorithm for inputs of size n. Let $\hat{s} = 2^{\lceil \log \frac{n+1}{2} \rceil} - 1$. Clearly, $C(1) = 0$, and, for $\hat{s} > 1$:

$$
\begin{aligned}
C(\hat{s}) &= B\left(\frac{\hat{s}-1}{2}, \frac{\hat{s}+1}{2}\right) + C\left(\frac{\hat{s}-1}{2}\right) \\
&\leq \frac{3}{2}\frac{\hat{s}+1}{2} \log\left(\frac{\hat{s}+1}{2}\right) + C\left(\frac{\hat{s}-1}{2}\right)
\end{aligned}
$$

Hence, $C(\hat{s}) \leq \frac{3}{2}(\hat{s} + 1)\log(\hat{s} + 1) - 3\hat{s} + O(\log \hat{s})$.

For arbitrary n, we need to consider the final sort completion $(\hat{s}, \hat{u})$. This gives:

$$
\begin{aligned}
C(n) &\leq \frac{3}{2}(\hat{s}+1)\log(\hat{s}+1) - 3\hat{s} + O(\log(\hat{s}+1)) + B\left(\hat{s}, \hat{u}\right) \\
&\leq \frac{3}{2}(\hat{s}+1)\log(\hat{s}+1) - 3\hat{s} + O(\log(\hat{s}+1)) + \frac{3}{2}\hat{u}\log(\hat{s}+1) \\
&\leq \frac{3}{2}n\left\lceil \log \frac{n+1}{2} \right\rceil - 3\hat{s} + O(\log n)
\end{aligned}
$$

It is easy to construct an input meeting this bound, i.e., this bound is tight. Thus, we have shown:

Lemma 8 *Chen's algorithm, with* $(\lambda, \mu, \gamma) = (2, 2, 2)$*, performs*

$$
\frac{3}{2}n\left\lceil \log \frac{n+1}{2} \right\rceil - 3.2^{\lceil \log \frac{n+1}{2} \rceil} + O(\log n)
$$

comparisons in the worst case.

The argument generalises for all larger λ, μ, and γ. The bound for $B(s, u)$ may be obtained as follows.

To begin, at most one subcall (s_1, u_1) of a call (s, u) having the balanced property $(\mu - 1)(s + 1) \geq u$ could become unbalanced. If it is unbalanced, as $u_1 \leq u$:

$$
(\mu - 1)(s_1 + 1) < u_1 \leq u \leq (\mu - 1)(s + 1) = (\mu - 1)(2s_1 + 2) = 2(\mu - 1)(s_1 + 1)
$$

and hence $(s_1, u_1) = (s_1, \Theta(u))$.

For $\lambda = \mu$, recalling that λ is a power of 2, we use an argument similar to the one above. This gives, for $s > 1$:

$$
B(s, u) \leq u + B(s_2, u_2) + B\left(s_1, (\lambda - 1)(s_1 + 1)\right) + \log\lambda\ (\lambda - 1)(s_1 + 1) + \sum_{k=3}^{\lambda+2} B\left(s_k, u_k\right)
$$

where $\sum_{k=2}^{\lambda+2} u_k = u - (\lambda-1)(s_1+1)$, $s_k = s_1$, and all (s_k, u_k) are balanced, for $k \geq 2$. Thus $B(s,u) \leq \log\lambda\ u \log(s+1)$. For arbitrary γ, this yields $C(n) = \Theta(\log\lambda\ n \log n)$, since the bound is easily seen to be tight.

For $\lambda < \mu$, if $\mu - 1$ is an integer multiple of $\lambda - 1$, we obtain:

$$B(s,u) \leq \frac{\mu-1}{\lambda-1} \log\lambda\ \Theta(u) + \sum_{k=1}^{\lambda^{\frac{\mu-1}{\lambda-1}}} B(s_k, u_k)$$

where $\sum u_k = u$, $s_k = s_1$, and all (s_k, u_k) are balanced. To obtain this, suppose the initial halving results in an unbalanced call (s_1, u_1). The halvings proceed as in the case $\lambda = \mu$. The one change is that, following these halvings, at most one call at the fringe may be unbalanced, and the process can iterate. But in each iteration, the size of the largest unsorted set decreases by at least $(\lambda-1)(s_1+1)$ and thus there are at most $\frac{\mu-1}{\lambda-1}$ iterations. Thus $B(s,u) \leq \frac{\mu-1}{\lambda-1} \log\lambda\ u \log(s+1)$. For arbitrary γ, this yields $C(n) = \Theta(\frac{\mu-1}{\lambda-1} \log\lambda\ n \log n)$, since again it is not hard to show the tightness of the bound.

Finally we can generalise this to all $2 \leq \lambda < \mu$, again obtaining a bound $\Theta(\frac{\mu-1}{\lambda-1} \log\lambda\ n \log n)$ for $C(n)$. We have shown Theorem 4.

B.2 Worst Case Analysis of Exchanges

We analyse the case $(\lambda, \mu, \gamma) = (2, 2, 2)$ in detail.

Since the number of partition exchanges is bounded by the number of comparisons, it suffices to analyse the exchanges occurring in block swaps, which we call *block exchanges*.

B.2.1 An upper bound

First we analyse top level calls of the form (m, ν), where $m = 2^k - 1$ and $k \geq 1$.

Let $D(s,u)$ be the worst case number of block exchanges for calls of the form (s,u).

Consider the call tree with (m,ν) as root, truncated at those calls which occur due to doubling, i.e., calls of the form $(s, s+1)$ generated by an unbalanced parent call (s,u). We call these calls (which are always balanced) *doubling calls*, and we call the remaining calls (s,u) with the balanced property, *non-doubling calls*. (No block exchanges occur within unbalanced calls exclusive of inner calls, and hence they play no role in the analysis.)

The non-doubling calls (s,u) may be partitioned into collections based on the values of s; the possible values are m, $\frac{m-1}{2}$, $\frac{m-3}{4}$, $\cdots$. We say an item in a sorted subarray is *stale* with respect to the truncated call tree if it has already been in the sorted subarray of a doubling call at a leaf of the truncated call tree; otherwise it is *fresh*. For each collection, the number of fresh items in the S sets is at least the number of stale items. Further, for each collection, the fresh items in each call are distinct and hence number $O(m+\nu)$. Each non-doubling call (s,u) performs at most $\frac{s}{2}$ block exchanges. Thus the total number of block exchanges occurring within all non-doubling calls is at most $\Theta((m+\nu)\log m)$.

The unsorted sets in the doubling calls are disjoint, thus the sum of the sizes of the unsorted sets for the doubling calls is at most ν.

Clearly, $D(1,1) = 1$, $D(1,2) = 2$, and, for $m > 1$:

$$D(m,\nu) \leq \Theta((m+\nu)\log m) + \sum_k D(s_k, s_k+1)$$

where $s_k \leq \frac{m-1}{2}$ and $\sum_k (s_k + 1) \leq \nu$. Hence $D(m,\nu) = O((m+\nu)\log^2 m)$.

We now bound $E(n)$, the worst case number of block exchanges for Chen's algorithm. Let $\hat{s} = 2^{\lceil \log \frac{n+1}{2} \rceil} -$

1 and $\hat{u} = n - \hat{s}$. Clearly, $E(1) = 0$, and, for $\hat{s} > 1$:

$$\begin{aligned} E(\hat{s}) &= D\left(\frac{\hat{s}-1}{2}, \frac{\hat{s}+1}{2}\right) + E\left(\frac{\hat{s}-1}{2}\right) \\ &\le O(\hat{s}\log^2\hat{s}) + E\left(\frac{\hat{s}-1}{2}\right) \end{aligned}$$

yielding $E(\hat{s}) = O(\hat{s}\log^2\hat{s})$. Clearly, for $\hat{u} \le \hat{s}$, $D(\hat{s}, \hat{u}) = O(\hat{s}\log^2\hat{s})$, yielding $E(n) = O(n\log^2 n)$.

The argument is unchanged for the case $\lambda = \mu$ (recall that λ is a power of 2). If $\lambda < \mu$, then, using similar arguments, we get:

$$D(m, \nu) \le \Theta(\nu \log m) + \sum_k D(s_k, (\lambda - 1)(s_k + 1))$$

where $s_k \le \frac{m-1}{2}$, $s_k < \frac{\lambda-1}{\mu-1}(m+1) - 1$, and $\sum_k (\lambda-1)(s_k+1) \le \nu$. Hence $D(m, \nu) = O(\nu \frac{\log^2 m}{\log^2 \frac{\mu}{\lambda}})$.

B.2.2 An input meeting the upper bound

We now consider a particular input to the algorithm which shows that this bound is tight.

Consider a call $(m, m+1)$ (where $m = 2^k - 1$ and $k \ge 1$) which generates (s_1, u_1) and (s_2, u_2) such that $u_1 = m + 1 - \sqrt{m+1}$ and $u_2 = \sqrt{m+1}$.

The balanced call (s_2, u_2) can be made to perform $\frac{m}{4}\log(m+1)$ block exchanges, by having the unsorted array split evenly in subsequent calls.

In the call $(s_1, u_1) = (\frac{m-1}{2}, m + 1 - \sqrt{m+1})$, since $s_1 < u_1 - 1$, s_1 doubles by means of a call of the form $(\frac{m-1}{2}, \frac{m+1}{2})$, to yield $s_1' = 2s_1 + 1 = m$ and $u_1' = u_1 - \frac{m+1}{2} = \frac{m+1}{2} - \sqrt{m+1}$.

The call (s_1', u_1') can be made to generate subcalls $(\frac{m-1}{2}, \frac{m+1}{2} - \sqrt{m+1})$ and $(\frac{m-1}{2}, 0)$, (with no block exchanges being expended).

The call $(\frac{m-1}{2}, \frac{m+1}{2} - \sqrt{m+1})$ can be made to generate subcalls $(\frac{m-3}{4}, \frac{m+1}{2} - \sqrt{m+1})$ and $(\frac{m-3}{4}, 0)$, (with no block exchanges being expended).

The call $(\frac{m-3}{4}, \frac{m+1}{2} - \sqrt{m+1})$ can be made to generate a series of calls similar to that generated by (s_1, u_1), and to repeat essentially the same pattern, i.e., a doubling call followed by two halvings, yielding calls of the form $(\frac{m-2^i+1}{2^i}, \frac{m+1}{2^{i-1}} - \sqrt{m+1})$, for $i = 1, 2, \cdots, \frac{1}{2}\log(m+1)$.

If $D(m, m+1)$ is the number of block exchanges for such a call $(m, m+1)$, then, for $m > 1$:

$$\begin{aligned} D(m, m+1) &\ge \frac{m}{4}\log(m+1) + D\left(\frac{m-1}{2}, \frac{m+1}{2}\right) + D\left(\frac{m-3}{4}, \frac{m+1}{4}\right) + \\ &\quad \cdots + D(\sqrt{m+1} - 1, \sqrt{m+1}) \\ &\ge \frac{m}{4}\log(m+1) - m + D\left(\frac{m-1}{2}, \frac{m+1}{2}\right) + D\left(\frac{m-3}{4}, \frac{m+1}{4}\right) + \\ &\quad \cdots + D(1, 2) \end{aligned}$$

Thus, $D(m, m) = \Theta(m\log^2 m)$ and we have:

Lemma 9 *Chen's algorithm, with* $(\lambda, \mu, \gamma) = (2, 2, 2)$*, performs* $\Theta(m\log^2 m)$ *exchanges in the worst case.*

The construction can be generalised when λ and $\frac{\mu-1}{\lambda-1}$ are powers of 2.

We start with a call of the form $(m, (\mu-1)(m+1))$, which is used to generate the call $(\frac{m+1}{2} - 1, (\mu - 1)(m+1))$. There is then an expansion yielding a new call $(\lambda\frac{m+1}{2} - 1, [2(\mu-1) - (\lambda-1)]\frac{m+1}{2})$. The halving here creates two subcalls, a terminal subproblem $(\lambda\frac{m+1}{4} - 1, \sqrt{m+1})$ and a continuing subproblem $(\lambda\frac{m+1}{4} - 1, (2\mu - \lambda - 1)\frac{m+1}{2} - \sqrt{m+1})$. For λ a power of 2, the latter problem, by repeated halving, becomes $(\frac{m+1}{2} - 1, (2\mu - \lambda - 1)\frac{m+1}{2} - \sqrt{m+1})$ which induces an expansion. The process is iterated, each time generating a terminal subproblem of the same size and continuing subproblems with successively smaller unsorted subarrays, until a subproblem $(\frac{m+1}{2}-1, [2(\mu-1)-\frac{\mu-1}{\lambda-1}(\lambda-1)]\frac{m+1}{2}-\frac{\mu-1}{\lambda-1}\sqrt{m+1})$ is generated. But this is $(\frac{m+1}{2}-1, 2(\mu-1)\frac{m+1}{4}-\frac{\mu-1}{\lambda-1}\sqrt{m+1})$, from which we generate $(\frac{m+1}{4}-1, 2(\mu-1)\frac{m+1}{4}-\frac{\mu-1}{\lambda-1}\sqrt{m+1})$. We now continue the above process, generating continuing subproblems but no more terminal subproblems. Each terminal subproblem can be made to perform $\Theta(\lambda\ m\log m)$ block exchanges. Each expansion $(\frac{m+1}{2^i}-1, (\lambda-1)\frac{m+1}{2^i})$, by repeated halving, generates a subproblem $(\frac{m+1}{2^i}\frac{\lambda-1}{\mu-1}-1, (\lambda-1)\frac{m+1}{2^i})$, which has the same form as the initial problem. We obtain the recurrence:

$$\begin{aligned}D(m, (\mu-1)(m+1)) \geq \Theta\Bigg(&\lambda\frac{\mu-1}{\lambda-1}\ m\log m\\&+\frac{\mu-1}{\lambda-1}D\left(\frac{\lambda-1}{\mu-1}\ \frac{m+1}{2}-1, (\lambda-1)\frac{m+1}{2}\right)\\&+\frac{\mu-1}{\lambda-1}D\left(\frac{\lambda-1}{\mu-1}\ \frac{m+1}{4}-1, (\lambda-1)\frac{m+1}{4}\right)\\&+\cdots\\&+\frac{\mu-1}{\lambda-1}D(2\,\frac{\sqrt{m+1}}{\lambda}-1, 2\,\frac{\mu-1}{\lambda}\ \sqrt{m+1}\Bigg)\end{aligned}$$

Thus $D(m, (\mu-1)(m+1)) \geq \Theta(\mu\ m\frac{\log^2 m}{\log^2\frac{\mu}{\lambda}})$ and we have Theorem 5.

The construction extends to all other values $\lambda < \mu$ also.

Remark 6 *One nice feature of Chen's algorithm is that it runs in* $\Theta(n\log n)$ *operations on natural orderings such as sorted and reverse sorted inputs, that are bad for quicksort and partition sort. Its worst case behaviour, however, arises on near sorted inputs. Specifically, imagine an input sorted apart from* k *arbitrarily placed items. In the worst case, Chen's algorithm will take* $\Theta(n\log n\log k)$ *time on such an input. Thus it is plausible that Chen's algorithm has the potential for somewhat slow operation on occasion in practice.*

B.3 Average Case Analysis of Exchanges

We analyse the algorithm for the case $\lambda(1+\epsilon) \leq \mu$, where $\epsilon > 0$ is a constant.

We analyse top level calls of the form $(m, (\gamma-1)(m+1))$. Consider the call tree with $(m, (\gamma-1)(m+1))$ as the root, truncated at unbalanced calls, i.e., calls of the form (s, u) with $(\mu-1)(s+1) < u \leq 2(\mu-1)(s+1)$.

Again, it suffices to analyse the block exchanges. Further it suffices to analyse block exchanges occurring within unbalanced calls (s, u), since the total number of block exchanges occurring within the non-truncated portion of the call tree is at most $\Theta(\gamma\ m\log m)$.

We note that balanced calls occurring within the unbalanced call (s, u), but outside inner unbalanced calls (s', u'), expend at most $\Theta(u\log s)$ exchanges. Since unbalanced calls (s, u) are disjoint within $(m, (\gamma-1)(m+1))$, the total number of block exchanges occurring within unbalanced calls (s, u), but outside inner unbalanced calls (s', u'), is at most $\Theta(\gamma\ m\log m)$.

Thus, we seek to bound, within an unbalanced call of the form (s, u), the total number of block exchanges done on average by inner unbalanced calls (s', u') with no intermediate unbalanced call ancestors.

Consider the unbalanced call (s, u). It generates an expanding call $C_1 = (s, (\lambda-1)(s+1))$, followed by a balanced call $C_2 = (\lambda(s+1)-1, u-(\lambda-1)(s+1))$. An unbalanced call (s', u') inside (s, u) must be inside C_1 or C_2. We shall show that the probability that a particular call (s', u') occurs is $\operatorname{negl}(s')$ in the following sense[5]: let $l = s'+u'$; then the probability that there is a call (s', u') for any given set of l contiguous items (in the sorted order) among the $s+u$ items in the unbalanced call (s, u) is $\operatorname{negl}(s')$. We set $u' = (\nu-1)(s'+1)$, and so $\mu < \nu \leq 2\mu-1$.

Lemma 10 *The probability that a particular call* (s', u') *occurs inside* C_1 *is* $\operatorname{negl}(s')$.

[5] $\operatorname{negl}(x) \stackrel{\text{def}}{=} o\left(\frac{1}{x^{O(1)}}\right)$.

Proof: Given a segment of size l items in the sorted order, the probability that it gives rise to a call of the form $(s', u') = (s', (\nu-1)(s'+1))$ (where $\mu < \nu \le 2\mu - 1$) is:

$$
\begin{aligned}
f(l,\nu) &\stackrel{\text{def}}{=} \frac{\binom{l}{\frac{l+1}{\nu}-1}\binom{\lambda(s+1)-1-l}{s-(\frac{l+1}{\nu}-1)}}{\binom{\lambda(s+1)-1}{s}} \\
&= g(s',\nu) \stackrel{\text{def}}{=} \frac{\binom{\nu(s'+1)-1}{s'}\binom{\lambda(s+1)-\nu(s'+1)}{s-s'}}{\binom{\lambda(s+1)-1}{s}}
\end{aligned}
$$

We define $f_1(x) \simeq f_2(x)$ if and only if $\frac{f_1(x)}{f_2(x)} = x^{\pm O(1)}$.

We use Stirling's approximation and obtain:

$$
\begin{aligned}
g(s',\nu) &\simeq \frac{\left(\nu+\frac{\nu-1}{s'}\right)^{\nu(s'+1)-1}}{\left[(\nu-1)+\frac{\nu-1}{s'}\right]^{(\nu-1)(s'+1)}} \cdot \frac{\left[(\lambda-1)+\frac{\lambda-1}{s}\right]^{(\lambda-1)(s+1)}}{\left(\lambda+\frac{\lambda-1}{s}\right)^{\lambda(s+1)-1}} \cdot \\
&\quad \frac{\left(\lambda-\nu\frac{s'}{s}+\frac{\lambda-\nu}{s}\right)^{\lambda(s+1)-\nu(s'+1)}}{\left(1-\frac{s'}{s}\right)^{s-s'}\left[(\lambda-1)-(\nu-1)\frac{s'}{s}+\frac{\lambda-\nu}{s}\right]^{(\lambda-1)(s+1)-(\nu-1)(s'+1)}} \\
&\simeq \frac{\nu^{\nu s'}}{(\nu-1)^{(\nu-1)s'}} \cdot \frac{(\lambda-1)^{(\nu-1)s'}}{\lambda^{\nu s'}} \cdot \frac{\left(1-\frac{\nu s'}{\lambda s}\right)^{\lambda s-\nu s'}}{\left(1-\frac{s'}{s}\right)^{s-s'}\left(1-\frac{\nu-1}{\lambda-1}\frac{s'}{s}\right)^{(\lambda-1)s-(\nu-1)s'}} \\
&\sim \left[\frac{\nu}{\nu-1}\right]^{(\nu-1)s'} \cdot \left[\frac{\nu}{\lambda}\right]^{s'} \cdot \left[\frac{\lambda-1}{\lambda}\right]^{(\nu-1)s'} \cdot \frac{e^{-\frac{\nu s'}{\lambda s}(\lambda s-\nu s')}}{e^{-\frac{s'}{s}(s-s')}e^{-\frac{\nu-1}{\lambda-1}\frac{s'}{s}((\lambda-1)s-(\nu-1)s')}} \\
&\sim \left[\frac{\nu}{\lambda} \cdot \left(\frac{\nu(\lambda-1)}{\lambda(\nu-1)}\right)^{\nu-1}\right]^{s'} . e^{\frac{s'^2}{s}\left(\frac{\nu^2}{\lambda}-\frac{(\nu-1)^2}{\lambda-1}-1\right)} \\
&= \left[\frac{\nu}{\lambda}\left(1-\frac{\frac{\nu}{\lambda}-1}{\nu-1}\right)^{\nu-1}\right]^{s'} . e^{-\frac{s'^2}{s}\frac{(\nu-\lambda)^2}{\lambda(\lambda-1)}} \\
&\le \left[\frac{\nu}{\lambda}e^{-(\frac{\nu}{\lambda}-1)}\right]^{s'} \ \mathrm{negl}\,(s') \\
&= \mathrm{negl}\,(s')
\end{aligned}
$$

Q. E. D.

A similar argument shows that the probability of such a call inside C_2 is also $\mathrm{negl}\,(s')$.

Lemma 11 *The average number of exchanges performed by inner unbalanced calls is* $O(n)$.

Proof: Inside C_1 the number of distinct unbalanced calls (s', u') for each fixed value of s' is at most $[\lambda(s+1)-1](2\mu-1)s'$. The number of exchanges done in each call is at most $\frac{(2\mu s')^2}{2}$. Thus the average number of exchanges occurring inside C_1 is bounded by

$$
O\left(\sum_{s'<s}(\mu s')^3\ \lambda(s+1)\ \ \mathrm{negl}\,(s')\right) = O(\lambda s)
$$

Similarly, inside C_2 the average number of exchanges is $O(u-(\lambda-1)(s+1))$. **Q. E. D.**

We have shown Theorem 6.

B.4 Notes

Remark 7 *We comment on the slightly mysterious expansion law in Chen's algorithm, the conditions for the two cases, and the choice of the input sizes for the top level sort completions. We would like to consider an input unbalanced when* $s < \left\lfloor \frac{s+u}{\mu} \right\rfloor$. *We would, however, also like to guarantee that* s *is invariably odd, to facilitate even splitting about the median of* S. *This is ensured by the expansion law* $s' = \lambda(s+1) - 1$, *provided* λ *and* γ *are powers of* 2. *In order to enforce consistency between the meanings of* λ *and* μ, *we approximate the unbalanced property by* $\mu(s+1) - 1 < (s+u)$. *The top level sort completions are such that they satisfy the balanced property.*

Remark 8 *Binary insertion sort and quicksort are similar to extreme cases of Chen's family of algorithms, namely with* $(\lambda, \mu, \gamma) = (1 + \frac{1}{s+1}, 1 + \frac{1}{s+1}, n+1)$ *and* $(\lambda, \mu, \gamma) = (s+u, s+u, n+1)$ *respectively.*

C Addendum to Section 4

Running times were measured using the `gethrtime` function; pseudo-random numbers were generated using the C library function `random`.

Comparison Counts $\overline{C_n} \pm \sigma(C_n)$												
n	Partition			Chen			Quasi-best-of-9			Best-of-3		
20000	305546	±	1097	302150	±	1002	309561	±	3865	320564	±	5033
25000	390220	±	1657	386222	±	874	395630	±	5249	411489	±	5164
30000	476495	±	1784	471470	±	1148	484372	±	4555	501055	±	9000
35000	563975	±	2188	557102	±	1038	573429	±	6294	597101	±	11042
40000	652301	±	2443	644593	±	1671	665629	±	8360	690239	±	12494
200000	3717621	±	4381	3692109	±	4196	3836652	±	35762	3985199	±	55805
250000	4733552	±	6244	4691273	±	2794	4860636	±	29934	5113716	±	116504
300000	5764341	±	6101	5708190	±	3749	5903429	±	35349	6179035	±	70063
350000	6801929	±	5976	6742757	±	3793	6979691	±	52562	7353966	±	122661
400000	7838234	±	8193	7781411	±	4596	8061647	±	59833	8479922	±	147614
2000000	43876624	±	16728	43521794	±	6941	45377851	±	318857	47963454	±	758987
2500000	55689824	±	17973	55221048	±	10036	57690500	±	593030	60870331	±	863631
3000000	67550833	±	16611	67096293	±	10638	70239232	±	577638	73948489	±	1005341
3500000	79610371	±	18565	79048824	±	13626	82516974	±	446344	87170240	±	1343546
4000000	91750180	±	22272	91049235	±	14761	94998765	±	567395	100124345	±	1278447

Move Counts $\overline{M_n} \pm \sigma(M_n)$												
n	Partition			Chen			Quasi-best-of-9			Best-of-3		
20000	238620	±	486	240568	±	415	236998	±	861	234812	±	966
25000	304218	±	643	306521	±	632	302076	±	1065	298977	±	1111
30000	371405	±	696	374208	±	648	368445	±	1204	364994	±	1350
35000	438857	±	1000	442895	±	884	435344	±	1517	430722	±	1956
40000	507520	±	829	511538	±	714	502487	±	1670	497986	±	2422
200000	2889652	±	1702	2907694	±	2350	2850238	±	6599	2823552	±	8165
250000	3673333	±	2210	3703661	±	3230	3627016	±	6476	3585593	±	14476
300000	4468240	±	2344	4503491	±	2779	4412779	±	8142	4361515	±	12100
350000	5270988	±	2519	5306375	±	2389	5205095	±	10217	5135099	±	20834
400000	6083693	±	2689	6119882	±	2661	6001817	±	13614	5928101	±	25557
2000000	33935685	±	7805	34190863	±	6099	33396794	±	70902	32937251	±	128283
2500000	43016274	±	6590	43351769	±	6378	42313581	±	112009	41759063	±	145696
3000000	52230311	±	5860	52539053	±	6584	51320739	±	108955	50665544	±	164316
3500000	61528422	±	5947	61874851	±	7843	60499358	±	97468	59690058	±	211206
4000000	70904440	±	8625	71415113	±	13125	69735446	±	129138	68844177	±	200174

We remark that there does not seem to exist a theory for choosing in an objective manner the number of times the trials are to be repeated to achieve a specified degree of confidence. The difficulty is that we lack estimates for the standard deviations σ of the quantities being measured, and hence it is difficult to predict the standard deviation of the computed sample mean after a specified number of trials. In practice, however, this does not seem to entail any difficulty in estimating the accuracy of the computed sample mean.