Bounds for Cut-and-Paste Sorting of Permutations

Daniel Cranston* Hal Sudborough† Douglas B. West‡

March 3, 2005

Abstract

We consider the problem of determining the maximum number of moves required to sort a permutation of $[n]$ using cut-and-paste operations, in which a segment is cut out and then pasted into the remaining string, possibly reversed. Every permutation of $[n]$ can be transformed to the identity in at most $\lceil 2n/3 \rceil$ such moves, and some permutations require at least $\lfloor n/2 \rfloor$ moves.

1 Introduction

The problem of sorting a list of numbers is so fundamental that it has been studied under many computational models. Some of these models require sorting "in place", where elements are moved around within an array of fixed length using various allowed operations.

A well-studied example is that of sorting using reversals of substrings. This is motivated by applications in measuring the evolutionary distance between genomes of different species [6]. The restricted case in which the reversed substring must be an initial portion of the permutation is the famous "pancake problem" [5, 8]. One can also give each integer a sign to denote the orientation of the gene [9, 6, 7]; in the case of prefix reversal, this becomes the "burnt pancake problem" [2].

Sorting by block transpositions has also been considered [4, 10]. Moving one block past another is a reasonable operation when the permutation is stored using a linked list. A restriction of this operation is inserting the head element between two later elements [1]. Some work has also been done in which both reversals and transpositions are allowed [11].

In this paper, we consider unsigned permutations and a more powerful operation that incorporates both block reversal and block transposition. A *(cut-and-paste) move* consists of cutting a substring out of the permutation, possibly reversing it, and then pasting it back

*University of Illinois, dcransto@uiuc.edu

†University of Texas—Dallas, hal@utdallas.edu

‡University of Illinois, west@math.uiuc.edu. Work supported in part by the NSA under Award No. MDA904-03-1-0037.

into the permutation between two adjacent integers. The storage model that supports this conveniently is a doubly-linked list.

We study permutations of $[n]$, where $[n] = \{1, \ldots, n\}$. A natural measure of how close a permutation is to the identity permutation is the number of pairs $\{i, i+1\}$ that occur consecutively (in either order). Call such pairs *adjacencies*. At most three adjacencies are created at each move (involving the two ends of the moved substring and the element(s) that formerly were next to the substring. For $n \geq 4$, there are permutations with no adjacencies, but the identity permutation has $n+1$ adjacencies (counting the front and back), so there are permutations of $[n]$ that require at least $\lceil (n+1)/3 \rceil$ moves to sort.

By a simple insertion sort, every permutation can be sorted in at most $n-1$ moves. Indeed, since every list of n distinct numbers has a monotone sublist of length at least $\sqrt{n}$ [3], we can insert the remaining elements one at a time into a longest monotone sequence, reversing the full list at the end if necessary, to sort in at most $n - \sqrt{n} + 1$ moves.

Thus trivially we have $\lceil (n+1)/3 \rceil \leq f(n) \leq n - \sqrt{n} + 1$, where $f(n)$ is the worst-case number of cut-and-paste moves needed to sort a permutation of $[n]$. In Section 2, we prove that $f(n) \geq \lfloor n/2 \rfloor$, obtaining many permutations of $[n]$ that require at least $\lfloor n/2 \rfloor$ moves to sort. In Section 3, we obtain $f(n) \leq \lceil 2n/3 \rceil + 1$ by presenting an algorithm that sorts any permutation of $[n]$ using at most $\lceil 2n/3 \rceil + 1$ moves.

2 The Lower Bound

We write a permutation of $[n]$ as a list of numbers within brackets, without commas; for example, $\pi = [\pi_1 \cdots \pi_n]$. For discussion of the lower bound, we prepend $\pi_0 = 0$ and postpend $\pi_{n+1} = n+1$ to a permutation π. A move is performed using three (not necessarily distinct) *cut point* indices i such that π_i and π_{i+1} become separated by the move. These cut points partition π into four disjoint substrings and yield three possible moves: the string between two of the cut points is inserted at the third cut point, possibly reversed. Given cut points i, j, k with $0 \leq i \leq j \leq k \leq n$, the results of the three legal moves on π are

$$\begin{aligned}
&\left[\pi_0 \cdots \pi_i \quad \pi_{j+1} \cdots \pi_k \quad \pi_{i+1} \cdots \pi_j \quad \pi_{k+1} \cdots \pi_{n+1}\right], \\
&\left[\pi_0 \cdots \pi_i \quad \pi_k \cdots \pi_{j+1} \quad \pi_{i+1} \cdots \pi_j \quad \pi_{k+1} \cdots \pi_{n+1}\right], \\
&\left[\pi_0 \cdots \pi_i \quad \pi_{j+1} \cdots \pi_k \quad \pi_j \cdots \pi_{i+1} \quad \pi_{k+1} \cdots \pi_{n+1}\right].
\end{aligned}$$

To reverse a string in place, let $i = j$ in the second form or $j = k$ in the third form.

The trivial lower bound of $(n+1)/3$ was obtained by considering adjacencies. To improve this bound, define a *parity adjacency* to be a pair of consecutive values in π having opposite parity. With 0 and $n+1$ fixed at the ends, the identity permutation has $n+1$ parity adjacencies. The key is to show that each move increases the number of parity adjacencies by at most 2.

Theorem 1. *$f(n) \geq \lfloor n/2 \rfloor$ for every positive integer n.*

Proof. A permutation with all even values before all odd values has one parity adjacency if n is even, two if n is odd. Since the identity permutation has $n+1$ parity adjacencies (counting the ends), it thus suffices to show that each move increases the number of parity adjacencies by at most 2.

Consider a move that inserts $\pi_k \cdots \pi_l$ between π_m and π_{m+1}, reversed or not. The three newly consecutive pairs of values are $\{\pi_{k-1}\pi_{l+1}, \pi_m\pi_k, \pi_l\pi_{m+1}\}$ if the block is not reversed, $\{\pi_{k-1}\pi_{l+1}, \pi_m\pi_l, \pi_k\pi_{m+1}\}$ if it is reversed. If the number of parity adjacencies increases by 3, then each new pair is a new parity adjacency and no parity adjacencies were destroyed.

The six values in the new pairs and broken pairs form a cycle of pairs that must alternate between opposite parity ($\not\equiv$) and equal parity ($\equiv$). We show these requirements below, depending on whether the moving block is reversed.

$$\pi_{k-1} \not\equiv \pi_{l+1} \equiv \pi_l \not\equiv \pi_{m+1} \equiv \pi_m \not\equiv \pi_k \equiv \pi_{k-1},$$
$$\pi_{k-1} \not\equiv \pi_{l+1} \equiv \pi_l \not\equiv \pi_m \equiv \pi_{m+1} \not\equiv \pi_k \equiv \pi_{k-1},$$

In each case, two of the three pairs of equal parity must have the same equal parity, but they appear on the cycle linked by a pair that gives them opposite parity. This contradiction prevents the number of parity adjacencies from increasing by 3. □

The proof in fact shows that $n/2$ moves may be needed to reach the entire set of permutations that, like the identity, alternate between odd and even values.

3 Upper Bound

For the upper bound, we no longer append 0 and $n+1$ at the ends of the permutation. Instead, we adopt the convention that the values n and 1 are consecutive. In particular, adding 1 to the value n produces the value 1; all computations with values are modulo n.

A *block* in a permutation is a maximal substring of (at least two) consecutive values in consecutive positions. A block is *increasing* if its second value is one more than its first. An increasing block can reach n and continue with 1. A permutation that consists of a single increasing block can be sorted in one move, shifting the initial part ending at n to the end.

An element in no block is a *singleton*. When singletons i and $i+1$ are made consecutive, they automatically form an adjacency. When blocks ending at i and $i+1$ are made consecutive, they produce a larger block only if oriented properly. Hence somehow moves that reduce the number of blocks are more valuable than moves that combine singletons.

We capture this phenomenon by giving higher weight to blocks than to singletons in measuring the "non-sortedness" of a permutation. The proof of the upper bound is constructive, providing an algorithm to sort permutations of $[n]$ with at most $\lceil 2n/3 \rceil + 1$ moves. We note that the algorithm for the pancake problem in [5] also treats blocks and singletons differently, but not via a weight function.

Theorem 2. *$f(n) \le \lceil 2n/3 \rceil + 1$, for every positive integer n.*

Proof. Let the *weight* of a permutation be

$$(number\ of\ blocks) + (2/3)(number\ of\ singletons).$$

Every permutation consisting of a single increasing block, including the identity permutation, has weight 1. The maximum weight of a permutation of $[n]$ is $2n/3$, achieved by permutations having no blocks. The *gain* of a move is the amount by which it reduces the weight.

Using at most one move, we establish an increasing block at the beginning of the permutation. We show that when such a block exists, we can gain at least 1 in one move or gain at least 2 in two moves, while maintaining that condition. We thus reach a permutation with weight 1 in at most $1 + \lceil 2n/3 \rceil - 1$ steps. One more move may be needed to sort the single increasing block at the end.

In each move, the string we cut out is a union of full blocks and singletons, never part of a block. Also, we never paste this string into the interior of a block. That is, we never break adjacencies, and the numbers of singletons and blocks can change only by creating adjacencies. Creating a block from two singletons gains 1/3. Absorbing a singleton into an existing block gains 2/3 (this is an *absorbing move*). Creating one block by combining two blocks gains 1. In all moves we ignore the possible gain resulting from closing the gap left by the extracted string, counting only the gain from pasting it elsewhere.

If our permutation has i and $i + 1$ in distinct blocks, then cutting out one of these blocks and pasting it next to the other gains at least 1. We call this a *block move*. When a block move is available, we perform it, leaving (or augmenting) the increasing block at the beginning of the permutation.

A *bonus move* is one that gains two adjacencies by the insertion. That is, the cut string (without splitting a block) has i and j at its ends, and it is inserted between two consecutive elements whose values are next to i and next to j. If one of these four elements was in a block, then the move gains 1. If at least two of them were in blocks, then the gain is at least 4/3 (this is an *extra bonus move*). When we make two moves to gain 2, they will be an absorbing move and an extra bonus move.

Let l be the last value in the initial increasing block, and let $l' = l+1$. Let p be the value in the position following l, and let p' be a value immediately above or below p that is not located next to p (there are two choices for p' if p is singleton, one if p is in a block). We consider several cases based on the condition of these elements.

If l' is in a block, then a block move is available, so we may assume that l' is a singleton. If p' is a singleton, or if the string S with l' and p' as its end-elements contains the block that p' ends, then we cut S and insert it between l and p (if p' is at the left end of S, we reverse S before inserting); see figure 1. Since l is in a block, this is a bonus move that gains at least 1 (extra bonus if p' is in a block).

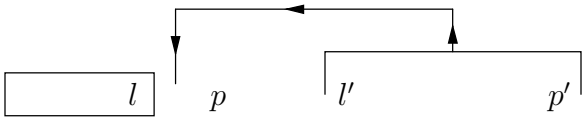

Fig. 1: A bonus move

Hence we may assume that p' is in a block B that is not contained in S. Let q be the end of B other than p'. Let q' be the value next to q (up or down) that is not in B. If q' or p is in a block, then a block move is available to combine B with the block containing q' or p, so we may assume that q' and p are singletons. See figure 2.

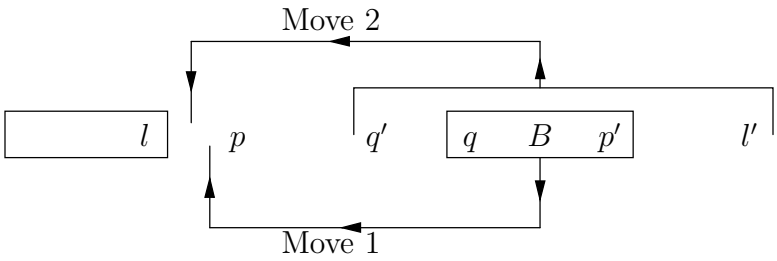

Fig. 2: An absorbing move followed by an extra bonus move

Now we make two moves. We first insert B between l and p, with p' next to p (if p' is at the left end of B, then we reverse B before inserting); this is an absorbing move, and it puts q next to l. We now insert the string from l' to q' between l and q (if q' is at the left end of this string, we reverse this string before inserting); since both l and q are now in blocks, this is an extra bonus move. As noted earlier, an absorbing move and an extra bonus move together gain (at least) 2. □

References

[1] M. Aigner and D. B. West, Sorting by insertion of leading elements. *J. Combin. Theory Ser. A* 45 (1987), 306–309.

[2] D. S. Cohen and M. Blum, On the problem of sorting burnt pancakes. *Discrete Appl. Math.* 61 (1995), 105–120.

[3] P. Erdős and G. Szekeres, A combinatorial problem in geometry. *Compos. Math.* 2 (1935), 464-470.

[4] Henrik Eriksson, Kimmo Eriksson, Johan Karlander, Lars Svensson, and Johan Wästlund, Sorting a bridge hand. *Discrete Math.* 241 (2001), 289–300.

[5] W. H. Gates and C. H. Papadimitriou, Bounds for sorting by prefix reversal. *Discrete Math.* 27 (1979), 47–57.

[6] Q.-P. Gu, S. Peng, and H. Sudborough, Approximating algorithms for genome rearrangements. *Proc. 7th Workshop on Genome Informatics* (1996).

[7] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Proc. 20th annual ACM Symposium on Theory of Computing* (1995) 178-189.

[8] M. H. Heydari and I. H. Sudborough, On the diameter of the pancake network. *J. Algorithms* 25 (1997), 67–94.

[9] H. Kamplan, R. Shamir, and R. E. Tarjan, Faster and simpler algorithm for sorting signed permutations by reversals. *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms* (1997), 344-351.

[10] Tzvika Hartman and Ron Shamir, A simpler 1.5-approximation algorithm for sorting by transpositions. *Proc. Symp. Combinatorial Pattern Matching* (2003), 156-169. 2003.

[11] M. T. Walter, Z. Dias, and J. Meidanis, Reversal and transposition distance of linear chromosomes. *String Processing and Information Retrieval: A South American Symposium* (1998), 96-102.