Advanced Engineering Technology and Application
An International Journal

http://dx.doi.org/10.12785/aeta/040201

Simple Linear Sorting Algorithm for 123-Avoiding Permutations

*Satybaldiyev Yernaz** *and Orazbayev Sanzhar**

Kazakh British Technical University, Almaty, Kazakhstan

Received: 23 Feb. 2015, Revised: 24 Mar. 2015, Accepted: 25 Mar. 2015
Published online: 1 May 2015

Abstract: We present basic techniques on pattern avoiding permutations and provide simple linear sorting algorithm for 123-avoiding permutations. Also did some experiment on stack-based sorting algorithm by Donald Knuth and provide results achieved. Experiment is based on counting number of iterations needed for sorting any permutation by D. Knuth's algorithm.

Keywords: Sorting, Pattern avoiding, 123-avoiding permutations, Stack

1 Introduction

Sorting is a fundamental operation in computing and appears in almost every program. Nowadays many companies have large datasets, big-data and they do a lot of operations on them and the sorting is a part of them. So if we some how can improve sorting algorithm then we will make big impact on our performance. In next sections we will describe pattern avoiding permutations and give simple linear sorting algorithms for 123-avoiding permutations.

Many works on pattern avoiding permutations devoted for counting them and now there international conference devoted to this subject. And many works done on enumeration of them. For some papers, see [2,4,7,8,9, 10].One of the most important result is Stanley-Wilf conjecture, recently proven by Marcus and Tardos, which states that number of permutations π of length n that avoid a fixed pattern σ is at most C^n for some constant C(σ).

Recalling that an arbitrary permutation is known to take Ω(n log n) comparisons, because lg(n!) = Ω(n log n) comparisons are required to distinguish between the n! possible inputs. Now, suppose we want to sort a permutation that is known to avoid a fixed pattern . By the Stanley-Wilf conjecture, the same lower bound argument in this case can only yield a bound of lg(C^n) = Ω(n) here. And theoretically, it is become possible sorting pattern avoiding permutations faster than Ω(n log n).

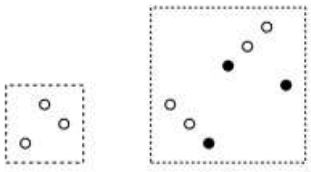


Fig. 1: The permutation (3, 2, 1, 5, 6, 7, 4) contains the pattern (1, 3, 2).

2 Pattern Avoiding Permutations

A permutation $\pi = (\pi_1, \pi_2, ..., \pi_n)$is said to contain a pattern $\sigma = (\sigma_1, \sigma_2, ..., \sigma_k)$ if contains a possibly non-contiguous subsequence $(\pi_1, \pi_2, ..., \pi_k)$ ordered in precisely the same way as σ. For example, (3, 2, 1, 5, 6, 7, 4) contains the pattern (1, 3, 2) since the subsequence (1, 5, 4) is ordered in the same way as (1, 3, 2). This is illustrated in Figure 1. If π does not contain σ, it is said to avoid σ.

Pattern avoiding permutations were firstly studied by Donald Knuth who provided linear algorithm for 231-avoiding permutations. His algorithm is based on single stack, he also proved that only 231-avoiding permutations can be sorted by single stack.[1]

Pattern-avoiding permutations arise naturally in a number of contexts. For example, the permutations

* Corresponding author e-mail: satybaldiyev.yernaz@gmail.com, sanzharorazbayev@gmail.com

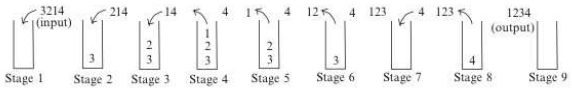


Fig. 2: Stack sorting the permutation 3214.

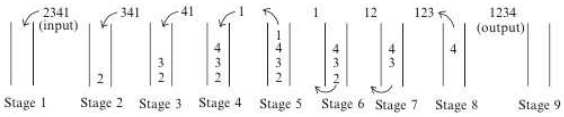


Fig. 3: Sorting the permutation 2341 with deque.

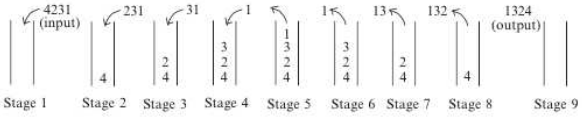


Fig. 4: Sorting the permutation 4231 with deque.

corresponding to riffle shuffling a deck of cards are precisely those that avoid the pattern (3, 2, 1).[3]

Study of the Tamari poset can be done by 132-avoiding permutations. And the Tamari poset is used in physics and algebra. For details, see [5].

Many work related for sorting pattern avoiding permutations did David Arthur. He proposed "fast-sorting" algorithm for some class of pattern avoiding permutations. His algorithm sorts input in O(n log log log n) time.

3 Sorting Industry

In this section we will illustrate how patterns can be used in the sorting industry.

3.1 Stack sortable permutations

A stack is a last-in first-out linear sorting device with push and pop operations (also known as insert and remove operations). In other words, a stack is a container for a linear sequence (in our case, for a permutation) that one is allowed to change by inserting new items (one at a time) at its tail and by removing tail items (again, one at a time). Initially the stack is empty and then a sequence of insertions interleaved with removals is made. Thus an input permutation is transformed thereby into an output permutation.

The greedy algorithm we are interested in for stack sorting a permutation $\pi = (\pi_1, \pi_2, ..., \pi_n)$ works as follows. We start with pushing π_1 onto the stack. Next, if $\pi_2 < \pi_1$ then we push π_2 onto the stack to be on top of π_1;on the other hand, if $\pi_2 > \pi_1$, we pop π_1 off and let π_2 enter the stack. More generally, suppose, at some point, the letters $\pi_1, \pi_2, ..., \pi_i$ have all been added to the stack (some of them could be still in the stack, others have been popped off), so we are reading π_{i+1}. We push π_{i+1} onto the stack if and only if π_{i+1} is less than the top element of the stack (which is easily seen to be π_i). Otherwise, we pop elements off the stack, one by one, until π_{i+1} is less than the top remaining stack element and then we push π_{i+1} onto the stack. When no more elements remain to be pushed onto the stack, we pop off all elements of the stack until it is empty. This produces a permutation S(π) as output.[6] Look for illustration, Figure 2.

Above greedy algorithm is Knut's algorithm and sorts only 231-avoiding permutations.

3.2 Sorting by Deque

Deque is a linear collection that supports element insertion and removal at both ends. The name deque is short for "double ended queue" and is usually pronounced "deck". A successful sorting of a permutation requires the existence of a sequence involving the allowed operations that leads to the increasing permutation. Of course, we now have more possibilities to sort a permutation.For example, the permutation 2341 requires three stacks in series to be sorted while it can be sorted with a single deque as shown in Figure 3.

On the other hand, not all permutations can be sorted with deque as shown in Figure 4.

4 Symmetry of Patterns

4.1 Reversing

Let $\sigma = (\sigma_1, \sigma_2, ..., \sigma_s)$be an arbitrary pattern. Then, we define the reverse pattern r(σ) to be $(\sigma_s, \sigma_{s1}, ..., \sigma_1)$, if we can sort σ-avoiding permutation then we can sort r(σ)-avoiding permutation.

Ex. 5 4 2 3 1 $--> $ 1 3 2 4 5

Proof. Simply reverse input permutation to obtain an σ-avoiding permutation and then sort.

4.2 Taking Complement

Let $\sigma = (\sigma_1, \sigma_2, ..., \sigma_s)$be an arbitrary pattern. Then, we define the complement pattern $\bar{\sigma}$ to be $(s+1-\sigma_1, s+1-\sigma_2, ..., s+1-\sigma_s)$, if we can sort σ-avoiding permutation then we can sort $\bar{\sigma}$-avoiding permutation.

Ex. 5 4 2 3 1 $--> $ 1 2 4 3 5

Proof. Take complement of input permutation to obtain an σ-avoiding permutation then sort.

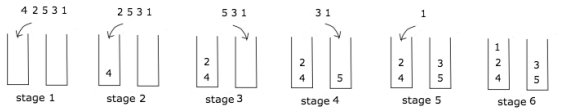


Fig. 5: Illustration of algorithm for input (4 2 5 3 1).

4.3 Classification of Permutations with Length 3

There will left only two classes (1, 2, 3) and (2, 3, 1) others will be derivatives of these two by above symmetries or combination of them.

1.(1, 3, 2) - reverse(2, 3, 1)

2.(2, 1, 3) - complement(2, 3, 1)

3.(3, 1, 2) - reverse(complement(2, 3, 1))

4.(3, 2, 1) - reverse(1, 2, 3)

Donald Knuth provided simple one stack algorithm for (2, 3, 1), below we will describe our algorithm for (1, 2, 3).

5 Simple Algorithm for 123-avoiding Permutations

Algorithm use 2 stacks and in each stack we keep numbers in descending order. So, for each number - x:

1.If stack1.isEmpty() or stack1[top] $>$ x, put x to stack1 and go to next number
2.If stack2.isEmpty() or stack2[top] $>$ x, put x to stack2 and go to next number
3.Alert that input is not 123-avoiding and break

At the end we will finish with 2 stack sorted in descending order and we just merge these stacks to obtain sorted sequence. Example:
input - [4 2 5 3 1]
stack1 - [4 2 1]
stack2 - [5 3]

Proof.

–If algorithm reaches step-3 it means that input is not 123-avoiding because stack2[top] is less than the last number and there is a number in stack1 which is less than stack2[top]. These three numbers form (1, 2, 3) pattern.
–if algorithm doesn't reach step-3, we will finish with 2 stacks sorted in descending order.

6 Generalization

Above simple linear algorithm for 123-avoiding permutations can be easily generalized for (12..k)-avoiding permutations that will use k-1 stacks, and at the end we will get k-1 sorted stacks and will merge them. But for k we must choose value smaller than log(n) to ensure efficiency of algorithm.

7 Experiment with Donald Knuth's Algorithm

What if we apply repeatedly Knuth's algorithm until we get sorted array? How many iterations/runs it will need? What is average, minimum, maximum of iterations? To answer these questions we implemented Knuth's algorithm and got some results.

7.1 How Implemented?

We generated all possible permutations of fixed length and apply the above algorithm to each of them.
Just recursively generated all possible permutations.

```
public void genPerm(int pos) {
  if (pos == n) {
    stackSort(a);
    return;
  }
  for (int i=0; i<n; i++) if (!used[i])
      {
    a[pos] = i;
    used[i] = true;
    genPerm(pos+1);
    used[i] = false;
    a[pos] = -1;
  }
}
```

And apply above algorithm until we get sorted array. At the end, number of iterations is stored in variable *steps*.

```
public void stackSort(int[] b) {
  int[] a = b.clone();
  int steps = 0;
  System.err.print(Arrays.toString(a));
  while (!isSorted(a)) {
    a = apply(a);
    steps++;
  }
  total+=steps;
   max = Math.max(max, steps);
  System.err.println(" " + steps);
}
```

Main stack-based algorithm.

```
public int[] apply(int[] a) {
     int[] st = new int[n];
     int top = 0;
     int[] b = new int[n];
     int cur = 0;
     for (int i=0; i<n; i++) {
       while (top > 0 && a[i] > st[top-1])
           {
         b[cur++] = st[top-1];
         top--;
       }
       st[top++] = a[i];
     }

     while (top > 0) {
       b[cur++] = st[top-1];
       top--;
     }
     return b;
  }
```

7.2 Results

Table 1: Results of experiment on Knuth's algorithm

Len	Max	Max inputs	Avg	# Total inputs
1	0	1	0	1
2	1	1	0.5	2
3	2	1	1.0	6
4	3	2	1.45	24
5	4	6	1.93	120
6	5	24	2.44	720
7	6	120	2.97	5040
8	7	720	3.52	40320
9	8	5040	4.09	362880
10	9	40320	4.67	3628800
11	10	362880	5.26	39916800
12	11	3628800	5.87	479001600

Surprizingly:

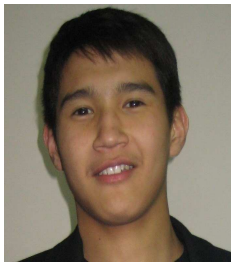
–Minimum iterations is easy - equal to 1
–Maximum iterations is 1 less than input length
–Number of inputs with maximum iterations is equal to Factorial(input length - 2)
–Average number of iterations increases approximately by 0.6

8 Conclusion

We described pattern avoiding permutations. Provided simple linear sorting algorithm for 123-avoiding permutations and did some experiment with Knuth's algorithm for sorting 231-avoiding permutations. Pattern-avoiding inputs problem is a new field and there is a great deal of opportunity for future work.[3]

References

[1] Donald E. Knuth, The art of computer programming, Addison-Wesley vol. 1, 1973.
[2] Sergi, Elizalde, Statistics on Pattern-avoiding Permutations, MIT. 2004
[3] David Arthur, Fast Sorting and Pattern-Avoiding Permutations, Stanford University
[4] Joel Brewster Lewis, Alternating, pattern-avoiding permutations, Feb 27, 2009.
[5] Marcelo Aguiar and Frank Sottile, strcuture of the loday-ronco Hopf algebra of trees
[6] Kitaev S. Patterns in permutations and words, Springer 2011
[7] R. Arratia On the StanleyWilf conjecture for the number of permutations avoiding a given pattern
[8] M. Bna Permutations avoiding certain patterns: The case of length 4 and some generalizations
[9] D. Marinov, R. Radoii Counting 1324-avoiding permutations
[10] F. R. K. Chung, R. L. Graham, V. E. Hoggatt, Jr., and M. Kleiman, The number of Baxter permutations, J. Combin. Theory Ser. A 24 (1978), no. 3, 382394.



Yernaz Satybaldiyev received the bachelor degree in Information Systems at Kazakh-British Technical University in Almaty. He has several awards in olympiads like Republican Olympiad in Informatics, IOI, ACM-ICPC, Russian Code Cup and Cotlin Challenge. His main research interests are: data structures, combinatorics, graph theory, game theory and number theory.

Sanzhar Orazbayev received the bachelor degree in Computer Science and Software at Kazakh-British Technical University in Almaty. He has several awards in IMO, ACM-ICPC and Imagine Cup. His main research interests are: combinatorics, sorting algorithms, game theory and number theory.