Sorting a sequence by swapping adjacent elements using minimum swaps

Asked 6 years, 3 months ago Active 4 years, 10 months ago Viewed 11k times

16

13

We have an unsorted sequence of N numbers (1, 2, 3, 4, ... N). We can sort the whole sequence by swapping adjacent elements in a specific order. Given a sequence, how do I compute the minimum possible swaps required to sort the sequence.

As an example, consider the sequence {4, 2, 5, 3, 1}.

The best way to sort this is using 7 swaps in the following order

1. Swap 3, 1: {4, 2, 5, 1, 3}
2. Swap 5, 1: {4, 2, 1, 5, 3}
3. Swap 4, 2: {2, 4, 1, 5, 3}
4. Swap 4, 1: {2, 1, 4, 5, 3}
5. Swap 2, 1: {1, 2, 4, 5, 3}
6. Swap 5, 3: {1, 2, 4, 3, 5}
7. Swap 3, 4: {1, 2, 3, 4, 5}

A greedy algorithm did not prove fruitful. A counterexample was easy to construct. The next obvious choice for approaching the solution was dynamic programming.

Say we have an unsorted sequence: {A1, A2, ...Ai, A(i+1), ..., An}. We know the minimum number of swaps required to sort the sequence {Ai, A(i+1), ..., An} is Min[Ai, A(i+1), ..., An}. The problem is finding Min[A(i-1), Ai, ..., An].

But I was not able to prove the validity of this solution. That is often the case with me. When I think I've solved the problem, the best I can do is obtain an 'intuitive' proof for it. I'm in high school and have no formal training in algorithmcs as such. I do it purely out of interest.

Is there rigorous mathematical notation that this problem can be converted into and proved formally? Can this notation be extended to other problems? How? I would appreciate it if it could be presented in a form comprehensible to a high-school-student.

algorithm

asked Jan 8 '14 at 8:11
Gerard
511 2 5 13

Can you give an example where any sequence of swaps fails, that is it is sub-optimal? In other words, in your example array, what can be a sequence of swaps that has a length more than 7? – Abhishek Bansal Jan 8 '14 at 9:06

2 Answers

Active Oldest Votes

28

This is a classical algorithm problem. The minimum number if swaps is equal to the number of inversions in the array. If we have index i and index j such that $a_i > a_j$ and i < j then this is called an inversion. Let's prove this statement! I will need a few lemmas on the way:

Lemma 1: If there is no inversion of two **adjacent** elements then the array is sorted.
Proof: Let's assume that no two adjacent elements form an inversion. This means that $a_i <= a_{i+1}$ for all i in the interval [0, n-1]. As `<=` is transitive this will mean that the array is sorted.

Lemma 2: A single swap of two adjacent elements will reduce the total number of inversions in the array by at most 1.
Proof: when we swap two adjacent elements a_i and a_{i+1} their relative position with respect to all the other elements in the array will remain unchanged. That is for all elements that were after a_{i+1}, they will still be after a_{i+1} and for all elements before a_i, they will still be before the a_i. This also means that if a_i or a_{i+1} formed an inversion with an element a_j then, they will still form an inversion with it after the swap. Therefor if we swap a_i and a_{i+1} we will affect only inversions that these two elements used to form. As two elements may participate in no more than one inversion we have also proved the lemma.

Lemma 3: We need to perform at least NI swaps of adjacent elements in order to sort the array where NI is the number of inversions in the array
Proof: In a sorted array there are no inversions. Also according to lemma 2, a single swap can reduce the number of inversions by at most one. Thus we need to perform at least as many swaps as is the number of inversions.

according to lemma 1, the array will be sorted and we are done.
Otherwise there is at least one pair of adjacent elements that form an inversion. We can swap them and thus reduce the total number of inversions by exactly once. We can continue performing this operation exactly NI times.

Now I have proven my statement from the beginning of the answer.

The only question left is how to count the number of inversions in a given array. You can do that using a slight modification of merge sort where you accumulate the inversions in the merge phase. You can have a look at this answer for details on how to implement that. The overall complexity of the algorithm is `O(n*log(n))` .

edited May 23 '17 at 12:02
Community ♦
1 1

answered Jan 8 '14 at 8:22
Ivaylo Strandjev
60.5k 12 98 152

Nice! This was a proof using words. Is there any formal mathematical notation in which the problem can be expressed? Mathematics makes the problem much easier to digest and manipulate. – Gerard Jan 8 '14 at 15:36

2 Inversion is a mathematical term and most of my answer is in fact mathematical. I have used mathematical notations where appropriate. – Ivaylo Strandjev Jan 8 '14 at 15:55

1 For more information on counting inversions: geeksforgeeks.org/counting-inversions – JMS Feb 10 '14 at 23:52

1

Thanks @Ivaylo Strandjev's explanation, to make the answer more complete, here is Java implementation:

```
// http://stackoverflow.com/questions/20990127/sorting-a-sequence-by-swapping-adjacent-
elements-using-minimum-swaps
// The minimum number if swaps is equal to the number of inversions in the array
public static long sortWithSwap(int [] a) {
    return invCount(a, 0, a.length-1);
}

private static long invCount(int[] a, int left, int right) {
    if(left >= right)   return 0;
    int mid = left + (right-left)/2;
    long cnt = invCount(a, left, mid) + invCount(a, mid+1, right);
    cnt += merge(a, left, mid, right);
    return cnt;
}

private static long merge(int[] a, int left, int mid, int right) {
    long cnt = 0;
    int i = left, j = mid+1, k = left;
    int[] b = new int[a.length];
    while(i<=mid && j<=right) {
        if(a[i] <= a[j])    b[k++] = a[i++];
        else {
            b[k++] = a[j++];
    while(i <= mid) {
        b[k++] = a[i++];
    }
    while(j <= right) {
        b[k++] = a[j++];
    }

    for(i=left; i<=right; i++)  a[i] = b[i];
    return cnt;
}
```

answered Jun 4 '15 at 8:18

spiralmoon
2,096 1 17 24