Editorial: Sorting

Subtask1

In subtask 1, Bob do not change sequence s, thus any method that can sort the sequence in increasing order will be a solution.

A bubble sort or insertion sort may cost at most $O(n^2)$ swaps. A merge sort or quick sort may cost at most $O(n\mathbf{log}n)$ swaps.

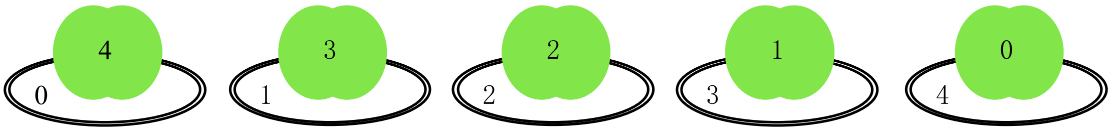
The optimized solutions is selection sort. Since the sequence contains 0, 1, ..., n-1 exactly once, we can swap value i into position i one by one (i=0, 1, ..., n-1). This costs at most n-1 swaps. It can be proved that the number of swaps is minimized.

Subtask2

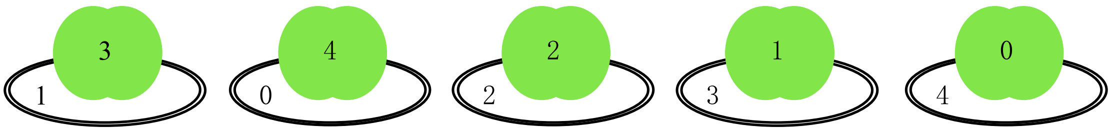
In subtask 2, Bob will swap positions 0 and 1, which makes the problem more complicated. However, the solution does not change much. We can swap value i into position i one by one from the rightmost position to the leftmost one (i=n-1, n-2, ..., 1, 0). The number of swaps is also minimized.

Subtask3

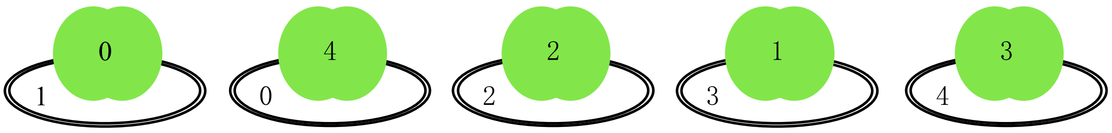
In subtask 3, we distinguish Alice and Bob's swaps. Let a[0]=0, a[1]=1, ..., a[n-1]=n-1 be n plates in a queue, and b[0]=s[0], b[1]=s[1], ..., b[n-1]=s[n-1] be n apples on plates 0, 1, ..., n-1. Then s[i]=b[a[i]].



If Bob swap two positions x and y, it can be viewed as two plates are swapped. For example, in the above case, if Bob swaps numbers at positions 0 and 1, the result sequence will be:



If Alice swap two positions p and q, it can be viewed as two apples s[p] and s[q] are swapped. For example, if Alice swaps numbers are positions 0 and 4, it can be viewed as apples s[0]=3 and s[4]=0 are swapped:



Now Bob swaps two plates in each round and Alice swaps two apples. Since they operates on difference objects, the sequence of operations can be separated. Let *t* be the number of rounds Alice takes. It can be viewed as Bob first swap *t* pairs of plates, then Alice swap *t* pairs of apples. Since Alice can sort all apples in increasing order in less than *n* swaps, the minimum number of rounds is less than *n*.

Solution: One solution for this problem would be enumerate *t* from 0 to *n*-1, and check whether Alice can sort all apples within *t* swaps after Bob swap first *t* pairs of plates. This costs $O(n^2)$ time and will get timeout for some test cases.

Binary Search: For this problem, if (p_0, q_0), (p_1, q_1), …, (p_{t-1}, q_{t-1}) be a solution for Alice in *t* swaps, then (p_0, q_0), (p_1, q_1), …, (p_{t-1}, q_{t-1}), (x_t, y_t), …, (x_{e-1}, y_{e-1}) must be a solution for Alice in *e* swaps ($e > t$). Which means that if there is a feasible solution in *t* swaps, then there must be feasible solutions in more than *t* swaps. Which makes the problem solvable using binary search. The time complexity reduces to $O(n\mathbf{log}n)$ and full mark is expected.

Implementation: The implementation of this problem may be a little bit error prone. It is recommended that programs first find out pairs of numbers (i.e. pairs of apples) to swap, then transform numbers into their positions. An inversed array that maps each number to its position is helpful.

Comments

The $O(n^2)$ time solution is essentially updating a set of cycles under splicing (edge insertion / deletion) operations. Some care is needed in test data making to distinguish it from $O(n\mathbf{log}n)$ time solutions, perhaps the bounds can be raised to ease this process.

This solution can also be made to run in $O(n^{1.5})$ or $O(n\mathbf{log}n)$ time using BBST-like data structures (e.g. http://contest.felk.cvut.cz/11cerc/solved/r/). This is way more work than the intended solution though.