Stable Quick Sort

A sorting algorithm is stable, if the relative order of elements that are "equal" are maintained.

To demonstrate this, consider a list of pairs of integers, [(1, 2), (2, 3), (2, 1), (3, 2)]

If we want to sort this list such that tuples with lesser first elements occur first, but among tuples with the same first value, we are not bothered with the order of the second value, for example both (2, 3) and (2, 1) can occur in any order and are considered "equal" to each other.

If we run **any** stable sorting algorithm with the comparator defined as above, the relative order of the elements with the same first value will remain unchanged. For the above example, a stable sorting algorithm will not change the above array given, however an unstable sorting algorithm could print [(1, 2), (2, 1), (2, 3), (3, 2)] instead.

We know that the some implementations of quick sort are not stable, specifically the one given in the question QCKSRT1 is not stable.

Here is an attempt at a modification of quick sort that is intended to be stable.

In the above code (quick sort), the line for choosing the pivot is left blank. If we choose the pivot to be the first value of the array (i.e $arr[l]$

), then the sorting algorithm will be stable, however if it is randomised, then it won't be.

You are to help fill the gap in the above algorithm while making it randomised.

Given an array A

(of size n), of pairs of integers, print (in increasing order) those indices (numbered from 0 to $n-1$

) that if chosen as a pivot for the above modified quick sort algorithm, results in a stable sorting algorithm.

Note that the if the output of your program is a list of indices called $candidates$

then choosing the pivot as $arr[candidates[rand(0,len(candidates)-1)]]$

would result in a version of quick sort that is both randomised and stable.

You are given a solution template, note that the input and output is taken care of in the template and you only need to correct the function $choosepivot$

in the above template.

Input

Note that you can directly fill in the $choosepivot$

function given in the solution template, so you do not have to worry about this section.

This is an explanation of how input is taken in the solution template.

There are two lines, the first line contains a single integer n

, the size of the array.

The second line contains $2n$

space separated integers, the $2ith$ and $(2i+1)th$ numbers denote the ith

tuple.

Output

Note that you can directly fill in the $choosepivot$

function given in the solution template, so you do not have to worry about this section.

This is an explanation of how output is done in the given solution template.

In the first line print m

, the number of valid choices of indices for the pivot in the first iteration of quick sort of this array

In the second line print those indices (space separated) in increasing order.

Constraints

$1 \leq n \leq 1000$

$-109 \leq$

every integer given ≤ 109

Sample Input

```
4
1 2 2 3 2 1 3 2
```

Sample Output

```
3
0 1 3
```

Explanation

The given array is $\{(1,2),(2,3),(2,1),(3,2)\}$

If we choose the index 2, then the partitioned array after the first iteration of quick sort becomes $\{(1,2),(2,1),(2,3),(3,2)\}$

which is affects the stability, since the order of $(2,3)$ and $(2,1)$ is changed.