push_swap

Sort data on a stack (or like data stricture), with a limited set of instructions, using the lowest possible number of actions.

Intro

The Push_swap project is a very simple and highly effective algorithm project: data will need to be sorted. You have at your disposal a set of int values, 2 stacks (or like data strictures) and a set of instructions to manipulate both stacks.

In a language of your choice, write 2 programs:

- The first, named checker which takes integer arguments and reads instructions – it's up to you how to accept this input (standard output, csv, etc.). Once read, checker executes them and displays OK if integers are sorted. Otherwise, it will display KO.

- The second one called push_swap which calculates and displays the smallest program using push_swap instructions that sorts integer arguments received.

Goals

To write a sorting algorithm is always a very important step in a coder's life, because it's often the first encounter with the concept of complexity.

Sorting algorithms, and their complexities are part of the classic questions discussed during job interviews. It's probably a good time to look at these concepts because you'll have to face them at one point.

Sorting values is simple. To sort them the fastest way possible is less simple.

Rules

1. The game is composed of 2 stacks (or like data strictures in the language of your choice) named **a** and **b**
2. To start with:
   - **a** contains a random number of either positive or negative numbers without any duplicates
   - **b** is empty
3. The goal is to sort numbers in ascending order into stack a
4. To do this you have the following operations at your disposal:

sa	swap a - swap the first 2 elements at the top of stack a. Do nothing if there is only one or no elements.
sb	swap b - swap the first 2 elements at the top of stack b. Do nothing if there is only one or no elements.
ss	sa and sb at the same time
pa	push a - take the first element at the top of b and put it at the top of a. Do nothing if b is empty.
pb	push b - take the first element at the top of a and put it at the top of b. Do nothing if a is empty.
ra	rotate a - shift up all elements of stack a by 1. The first element becomes the last one.
rb	rotate b - shift up all elements of stack b by 1. The first element becomes the last one.
rr	ra and rb at the same time
rra	reverse rotate a - shift down all elements of stack a by 1. The last element becomes the first one.
rrb	reverse rotate b - shift down all elements of stack b by 1. The last element becomes the first one.
rrr	rra and rrb at the same time

Example:

To illustrate the effect of some of these instructions, let's sort a random list of integers. In this example, we'll consider that both stacks are growing from the right.

Stack at start:

stack **a**	stack **b**
2	
1	
3	
6	
5	
8	

Instruction/s: **sa**

stack **a**	stack **b**
1	
2	
3	
6	
5	
8	

Instruction/s: **pb pb pb**

stack **a**	stack **b**
6	3
5	2
8	1

Instruction/s: **ra rb**

stack **a**	stack **b**
5	2
8	1
6	3

Instruction/s: **rra rrb**

stack **a**	stack **b**
6	3
5	2
8	1

Instruction/s: **sa**

stack **a**	stack **b**
5	3
6	2
8	1

Instruction/s: **pa pa pa**

stack **a**	stack **b**
1	
2	
3	
5	
6	
8	

This example sorts integers from a in 12 instructions. Can you do better?

checker

- You have to write a program named checker, which will get as an argument a formatted list of integers. The first argument should be at the top of the stack (be careful about the order).
- Checker will then wait for instructions.
- If after executing those instructions, stack **a** is actually sorted and **b** is empty, then checker must display “OK”. In every other case, checker must display “KO”.
- Handle errors. Errors include, for example: some arguments are not integers, some arguments are bigger than an integer, there are duplicates, an instruction does not exist and/or is incorrectly formatted.

> Thanks to the checker program, you will be able to check if the list of instructions you’ll generate with the program push_swap is actually sorting the stack properly.

Examples:

List of ints: 3 2 1 0
Instructions: rra pb sa rra pa
Output: OK

List of ints: 3 2 1 0
Instructions: sa rra pb
Output: KO

List of ints: 3 two 1
Output: Error

push_swap

- You have to write a program named push_swap which will receive as an argument the stack **a** formatted as a list of integers. The first argument should be at the top of the stack (be careful about the order).

- The program must display the smallest list of instructions possible to sort the stack **a**, the smallest number being at the top.

- The goal is to sort the stack with the minimum possible number of operations. During defence we'll compare the number of instructions your program found with a maximum number of operation tolerated. If your program either displays a list too big or if the list isn't sorted properly – your program fails.

- Handle errors. Errors include, for example: some arguments are not integers, some arguments are bigger than an integer, there are duplicates, an instruction does not exist and/or is incorrectly formatted.

Examples:

List of ints: 2 1 3 6 5 8
Output: sa pb pb pb sa pa pa pa

List of ints: 3 two 1
Output: Error

> If your checker program displays KO when fed the list of instructions, it means that your push_swap came up with a list of instructions that doesn't sort the list.