PUSH SWAP

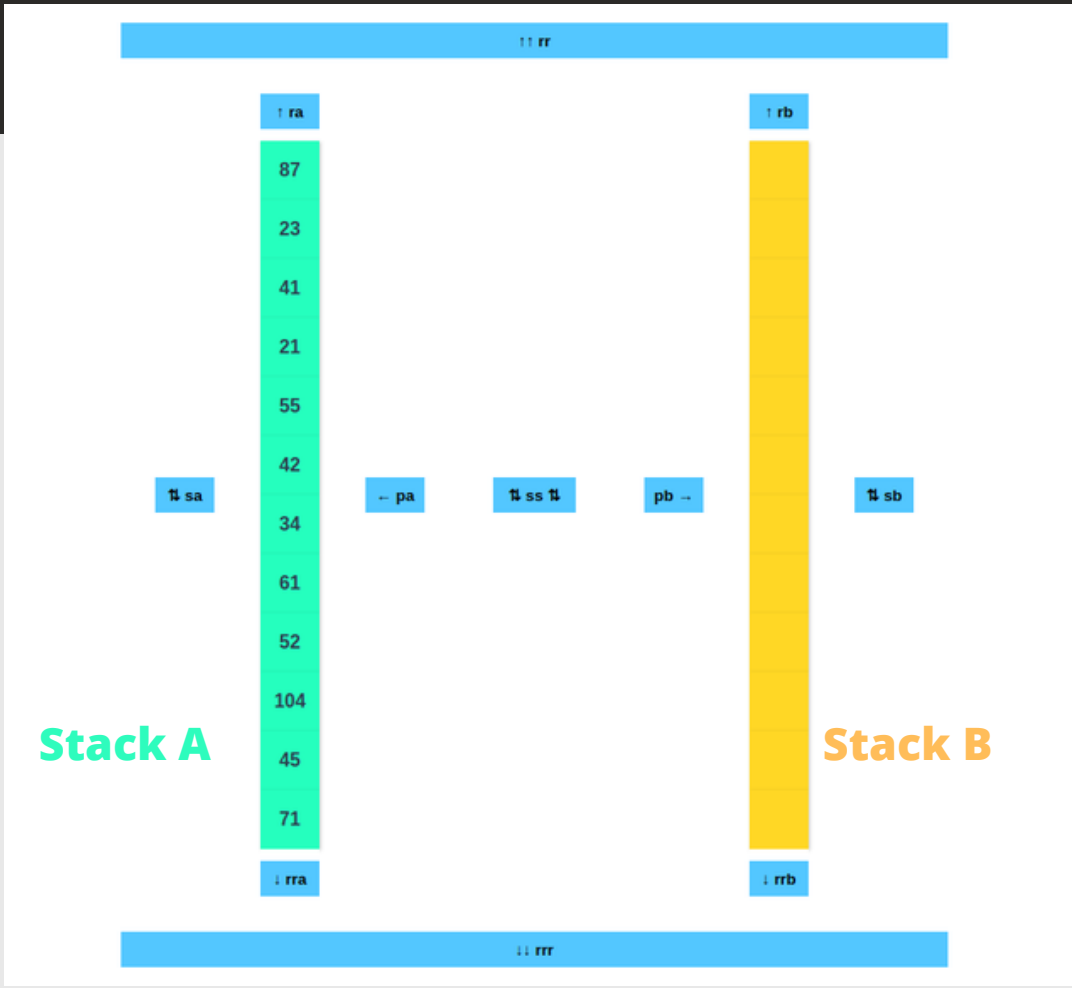
PUSH SWAP IS A PROJECT MEANT TO SORT A STACK OF INTEGERS USING THE LOWEST POSSIBLE NUMBER OF INSTRUCTIONS IN THE LOWEST POSSIBLE TIME.

Prepared by :
Athmane Naciri
Student at 1337

About

This project is one of my favorite projects.This project will make you sort data on a stack, with a limited set of instructions, using the lowest possible number of actions. To succeed you'll have to manipulate various types of algorithms and choose the most appropriate solution (out of many) for an optimized data sorting.

You have two stacks called Stack A and Stack B. Stack A is given a random list of unorganized numbers. You must take the random list of numbers in Stack A and sort them so that Stack A is organized from smallest to largest.



- A website that helps understand the implementation of actions: https://2g2uk.csb.app/

Mission and vision

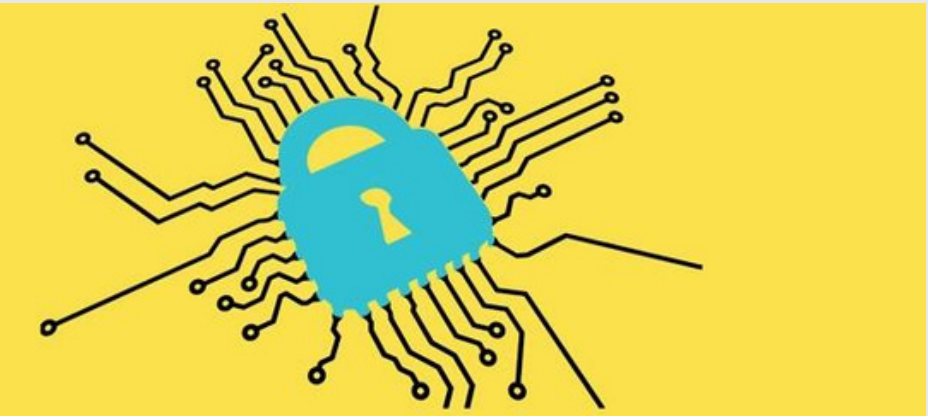
After reading a lot about this project, the hardest part is not writing the code, the hard part is finding the correct algorithm to sort the numbers into the smallest possible actions .

To do this I choose to use linked lists because it will be easier than arrays and after reading the algorithm you will understand Why (?) .

To make this document clear, let's break it down into parts

The Magic ALGo

To explain further if the user enters less than 6 digits, we need to separate these cases so that we can reduce the actions as much as possible.



what is the

Easiest way

to Do that ?

If Stack Size == 1

How Can you Sort One number ? exactly you can't

- The program Should display nothing (0 instruction) .

If Stack Size == 2

- Check if The Stack Sorted If Not swap Stack a .

If Stack Size == 3

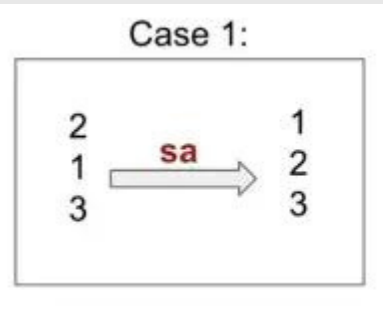
To explain this part, i must first tell you how numbers are stored in order to know how to use them ...

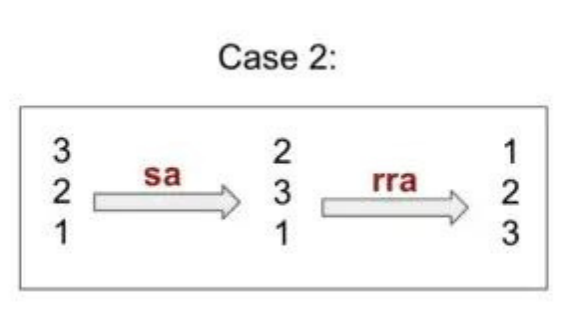
i Declare the following structure, so that I can use the data to store the number and position to know its rank among other numbers and finally the next pointing to the next node.

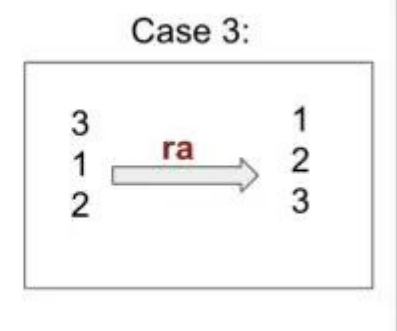
```
typedef struct node
{
    int         data;
    int         position;
    struct node *next;
}t_list;
```

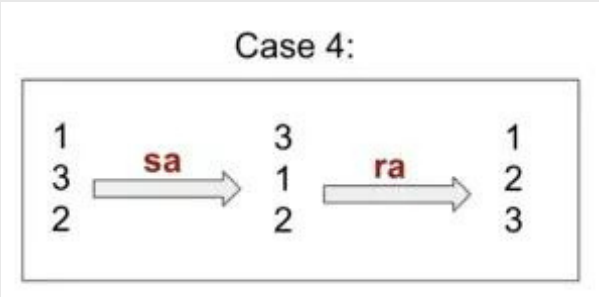
Now, We Do have a full stack of random numbers waiting for us to sort it .

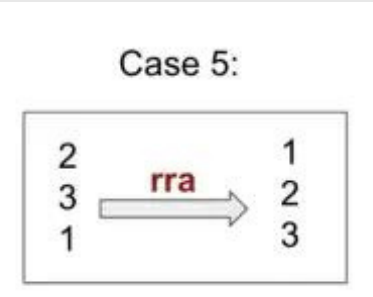
Those are the 5 Cases Of three random Numbers :











- How can we implement That in our Code ? Easily we compare The position of The three nodes and then we choose the right actions to DO !

Check the Code for More understanding .

If Stack Size == 4

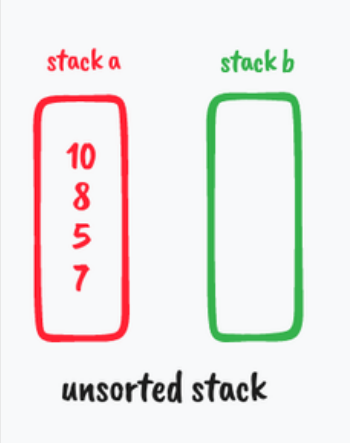
- In this case, your first step is to find the smallest number on the stack ; You can use position on that... and what I mean by "find" is the index of the number on the stack in another word is it on the top half or the bottom half ?

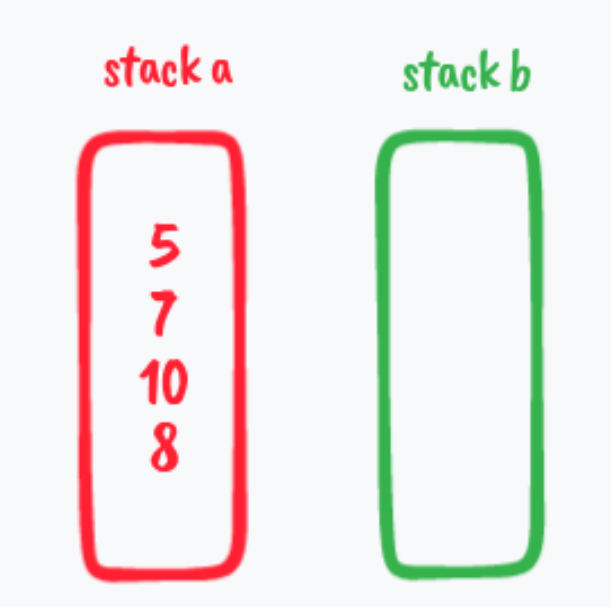
You may ask why do I need to know that ?

➡ So you will know which actions to use ...

- If You find the smallest number on the top half you will use ra and use rra if not .
- Now you have three numbers in stack a and one number(the smallest one) in stack b . sort the first three As I explained earlier and push the number to stack a.

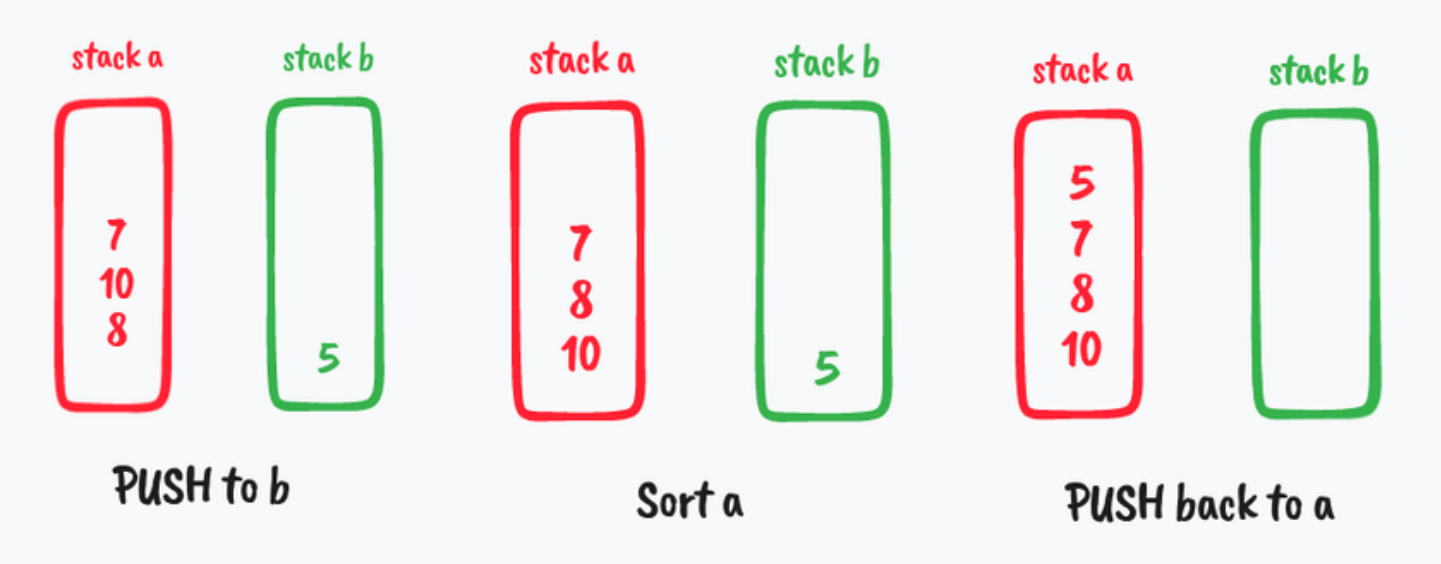
Example :





The smallest number is 5, in the bottom half we use **rra** until we make it to the top of the stack .

Finally :



If Stack Size == 5

- In this case, we will use the same algorithm as the last. The only difference is instead of pushing one number, you'll push two numbers (the smallest one and the second smallest one) so you can sort stack A and push back the other two numbers.

Étape 2

- **This is going to be hard to understand, you think. No, just keep your attention and focus !!!**

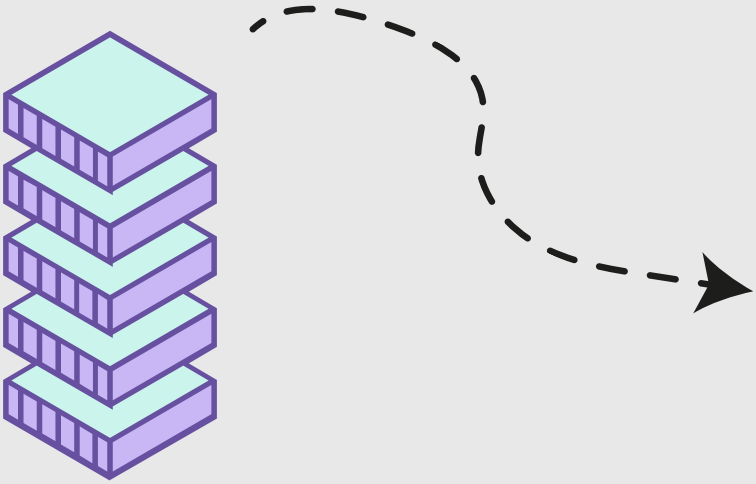
From now on we can't select each number by itself, to make it easier to work, we will divide the stack into chunks so that we can push chunk by chunk until all the stack is emptied, this is not a random action we need to push the number into an exact scheme, to make it understandable here is "**100**" as an example:

Steps :

- The stack size is 100 and since we start the position at 0 means we have a range of numbers between 0-99, the first thing we do is divide the stack by 5. Why exactly five? Because we pray 5 times a day . kidding ! Because when we divide by a larger number, the number of pieces will get bigger and that means more actions and we don't need that. Some how numbers less than 200 give exactly correct actions when divided by 5.

- Now what we need to do is check the position of the first node in the stack if it is in the first chunk which ranged from 0-19, if not then rotate a until you find one, now you have a node with a position in that range.

Chunk 0 -19 Chunk 20-39 Chunk 40-59 Chunk 60-79 Chunk 80-99



Chunk 0 -19

- As before, check if the position of the number is in the top half of the chunk or in the bottom half; If the number in the top half pushes it to stack b if it is in the bottom push it to stack b and rotate it until all numbers in that chunk are pushed to b and move to the next chunk. Repeat the same way until stack A is empty. You may ask why are we doing all of this ? Great question... so the numbers takes that schema in stack b and that's pretty necessary, so the way we push them back to stack a doesn't take a bunch of actions.

The scheme of numbers in stack B will be much like this

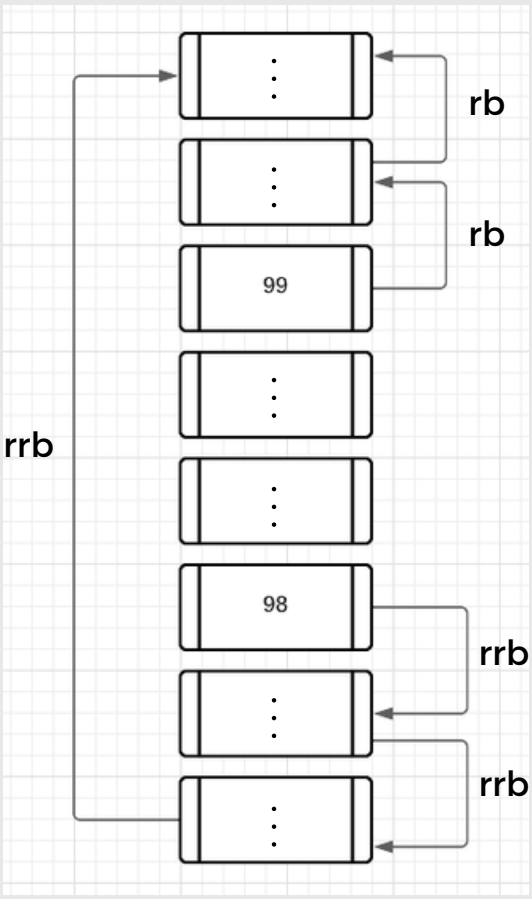
→ Your stack 'a' is empty and you have a full stack 'b' of numbers shaped as above .

Now what is the way to push them back ?

How that scheme we've created can help us categorize fewer actions ?

Easy, in the Scheme if you notice that the biggest numbers are stored on the sides, so your task is to find the biggest number (99th position in this case) and the second biggest number (98th position in this case) and check which one of them near to the top of the stack.

- if the biggest one is the nearest one push him to stack a . first of all what do i mean by nearest ?



The largest number at position 99 in this case needs 2 actions on top, while the other number needs 3 actions .

- If the second biggest number is closest, push it to stack a and go to find the largest number in the stack again, then push it to a and swap a.

in this way you will sort your stack in a smart way with a Reasonable number of actions

Étape 3

- this part is not different at all from the previous part. The only difference is that instead of dividing by 5, divide by 10, because the number will increase and we will need more chunks so that there are not many numbers in one chunk, and this will make the movements more.

This was the end of the document

THANK YOU !

https://github.com/athmanenaciri/push-swap