

# push\_swapを 理解するためのスライド

作成者:nfukada(nafuka11)

# おしながき

1. push\_swapとはどんな課題か
2. データ構造
3. アルゴリズム
4. push\_swapの実装

**push\_swap**とはどんな課題か

# **push\_swap**を一言で表すと

与えられた数値をソートするプログラムを作る課題

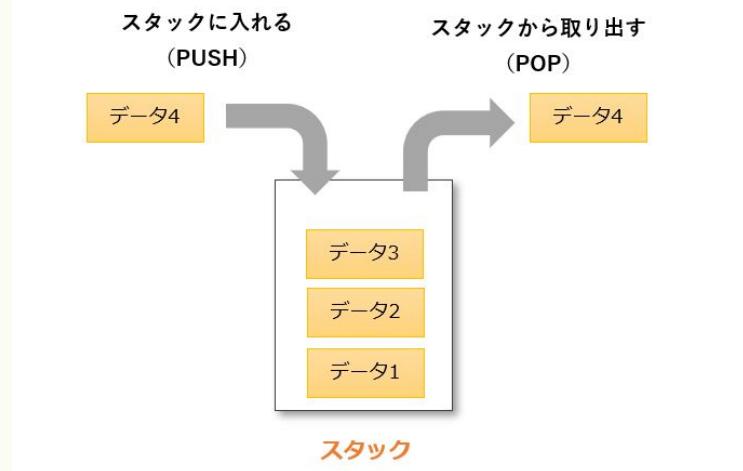
- スタックというデータ構造が2つあり、与えられた数値が入ってる
- 専用の命令でスタックを操作
- 2つのプログラムを作る

下線部について説明していきます

# データ構造のスタック

[wikipedia](#) より

スタックは、コンピュータで用いられる基本的なデータ構造の1つで、データを後入れ先出し(LIFO: Last In First Out; FILO: First In Last Out)の構造で保持するもの



出典: [Stack\(stack\)とは - ITを分かりやすく解説](#)

# push\_swapの スタック

- スタックaとbがある
- 最初はaに数値が入って  
いる
- 最終目標は、aに入った  
数値を昇順にソートする  
こと

初期状態（引数から与えられる）

a	b
[ 2] 3	
[ 0] 1	
[ 1] 2	



こうしたい

a	b
[ 0] 1	
[ 1] 2	
[ 2] 3	

# 専用の命令

全部で11個。大まかな系統は以下

- swap: スタックの先頭と次で値を交換
- push: 先頭の値を別のスタックへ移動
- rotate: スタックを回転
- reverse rotate: スタックを逆回転

詳細は、[Push Swap: The least amount of moves with two stacks | by Jamie Dawson | Medium](#) の図が分かりやすいです

# 2つのプログラム

push\_swap

- 引数から数値を複数受け取り、命令を使ってソートする
- 命令を標準出力に出力する

checker

- push\_swapの出力を確認する  
(= 標準入力から命令を読み取る)
- 標準出力にソートされたか出力
  - ソートされた:OK
  - ソートされない:KO

# プログラム実行例

```
# push_swapとcheckerの動作
```

```
$ ./push_swap 3 2 1
```

```
sa
```

```
rra
```

```
$ ./checker 3 2 1
```

```
(標準入力を受け付ける)
```

```
# push_swapとcheckerをパイプでつないで動作確認
```

```
$ ./push_swap 3 2 1 | ./checker 3 2 1
```

```
(OKかKOを表示する)
```

```
# 引数が数値でない or intの範囲を超えてたら Errorと表示
```

```
$ ./push_swap a
```

```
Error
```

```
$ ./push_swap 2147483648
```

```
Error
```

```
# 引数なしは何もしない
```

```
$ ./checker
```

```
$
```

# データ構造

# 今回使ったデータ構造：双方向循環リスト

双方向循環リスト = 連結リスト + 双方向 + 循環

- 連結リスト: 各要素が1つ後ろの要素のポインタを持ったデータ構造。可変のデータを持つ
- 双方向: 各要素が1つ前の要素を持つ
- 循環: 末尾の要素が先頭の要素を、先頭の要素が末尾の要素を参照できる

# 双方向循環リストを使った理由

要素の追加削除を頻繁に行う → 連結リスト

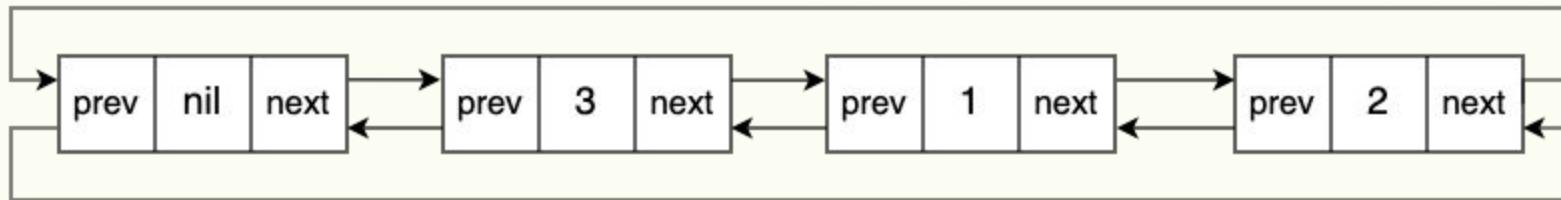
先頭と先頭の次、末尾にアクセスする必要あり → 双方向

回転、逆回転する → 循環

# 双方向循環リスト：図で表すと

1つの要素をノードという

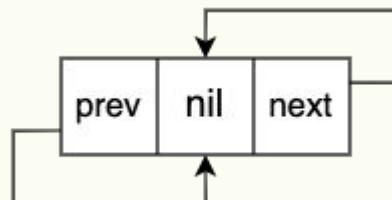
nilは番兵ノード



# 双方向循環リスト：番兵ノード

- 連結リストのNULLに相当するノード
- 番兵ノードがあることで、リストを先頭から見ていても無限ループにならない  
(番兵ノードがない場合、先頭の要素を覚えておく必要がある)

要素がないときは → のような感じになる



# アルゴリズム

# 分割統治法

そのままでは解決できない大きな問題を

小さな問題に分割して全て解決することで

大きな問題を解決する

# クイックソート

分割統治法を使った  
ソートアルゴリズム

- 基準値の大小でデータを分割  
(Partition)
- 分割したデータをさらに分割
- 分割を繰り返すことで最終的に  
データがソートされる



出典: [AIZU ONLINE JUDGE: ALDS1\\_6\\_C](#)

# push\_swapの実装

# **push\_swapの実装**

要素数3個以下、6個以下、7個以上で場合分け

6個以下はルールベース。7個以上はクイックソートする

共通の考え方

- 2つのスタックを先頭が繋がった配列のように考える

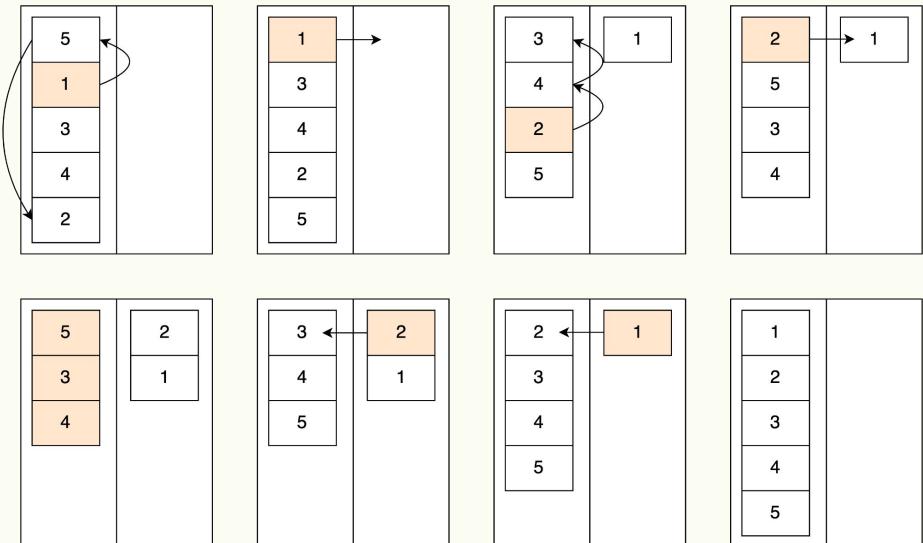
# **push\_swap**の流れ：3個以下のとき

- 1個なら何もしない
- 2個なら大小に応じてswap
- 3個なら大小に応じて5パターンの操作

参考：[Push Swap: The least amount of moves with two stacks | by Jamie Dawson | Medium](#) の「3 random numbers」

# push\_swapの流れ：6個以下のとき

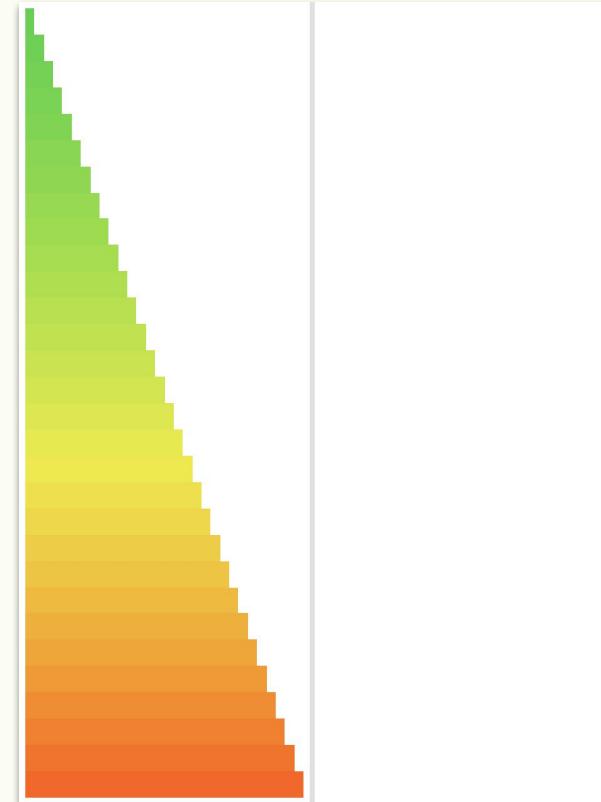
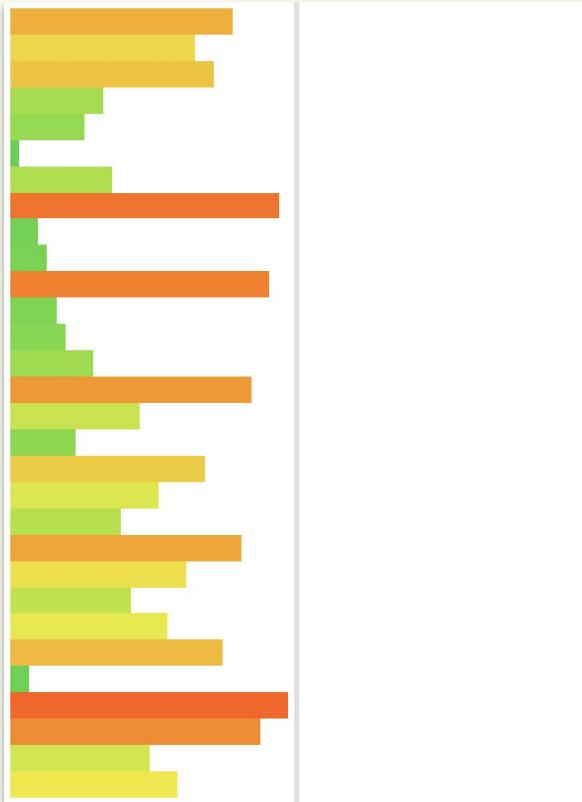
1. aが3個になるまで  
小さい数からbにpush
2. 残ったaを3個のルールでソート
3. bをaにpushして戻す



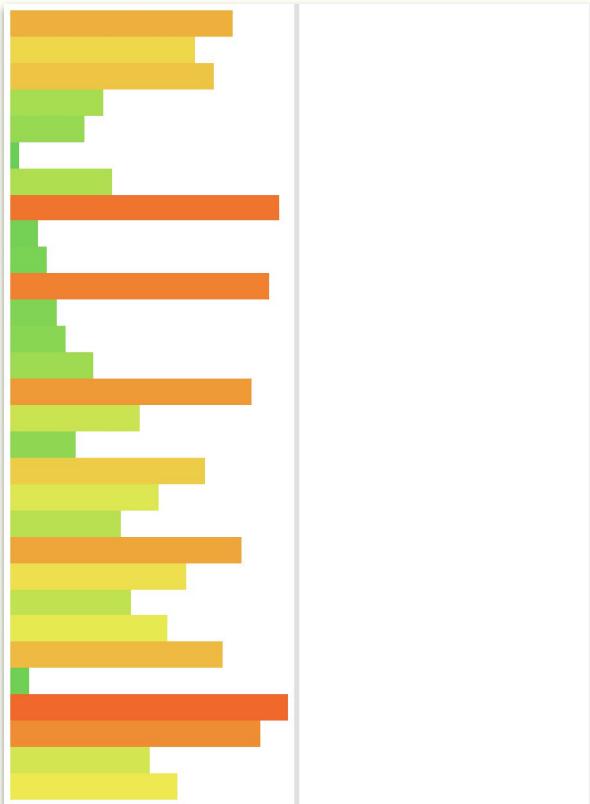
# **push\_swapの流れ：7個以上のとき**

aの末尾へ小さい順に数字を挿入し、ソート済み扱いにしていく

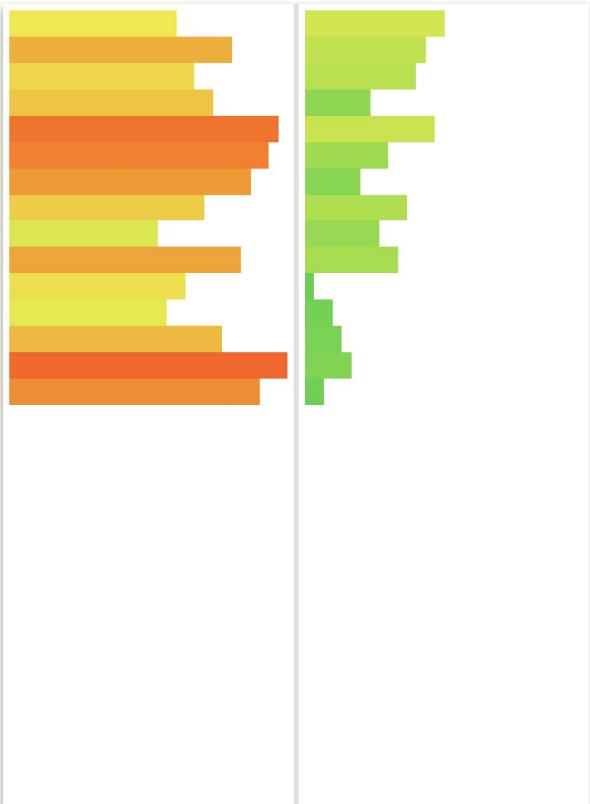
1. 初期状態。aに数値がある
2. aの数値を大小で分け、bにpushする
3. bが分割できなくなるまで、bの数値を大小で分け、aにpushす
4. 残ったbの数値を小さい順にaの末尾へ移動する
5. aをbへpushし、bの分割を繰り返す



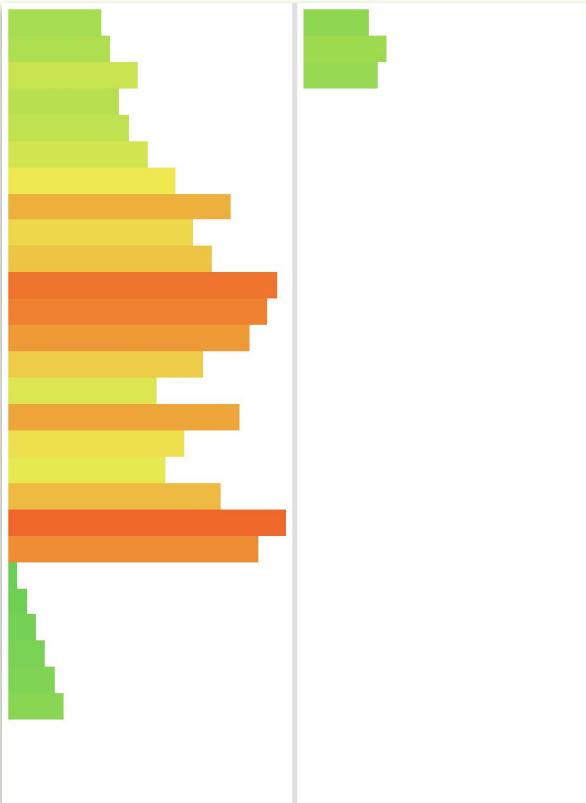
左の初期状態から右の状態にしたい



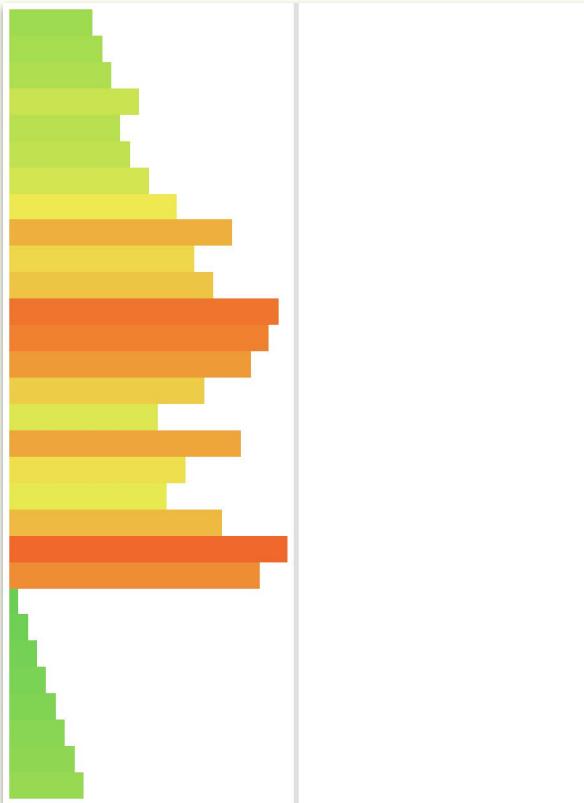
1. 初期状態



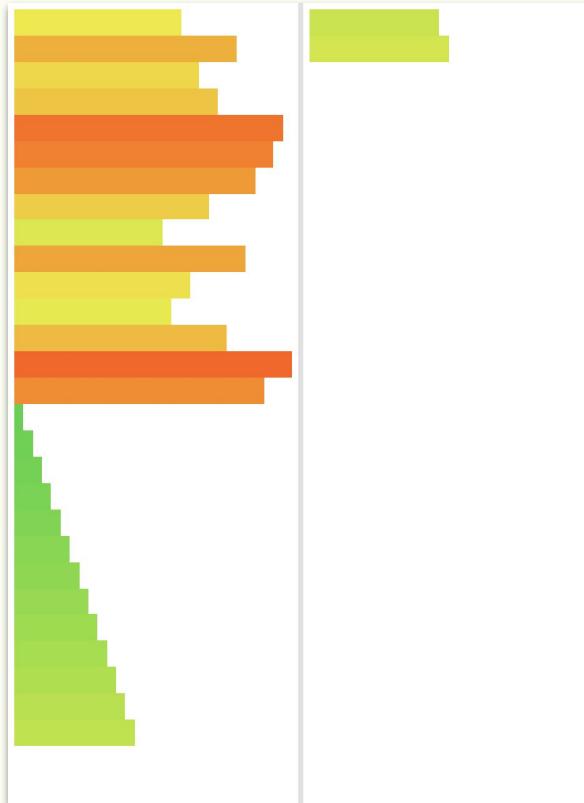
2. aを大小で分けbへpush



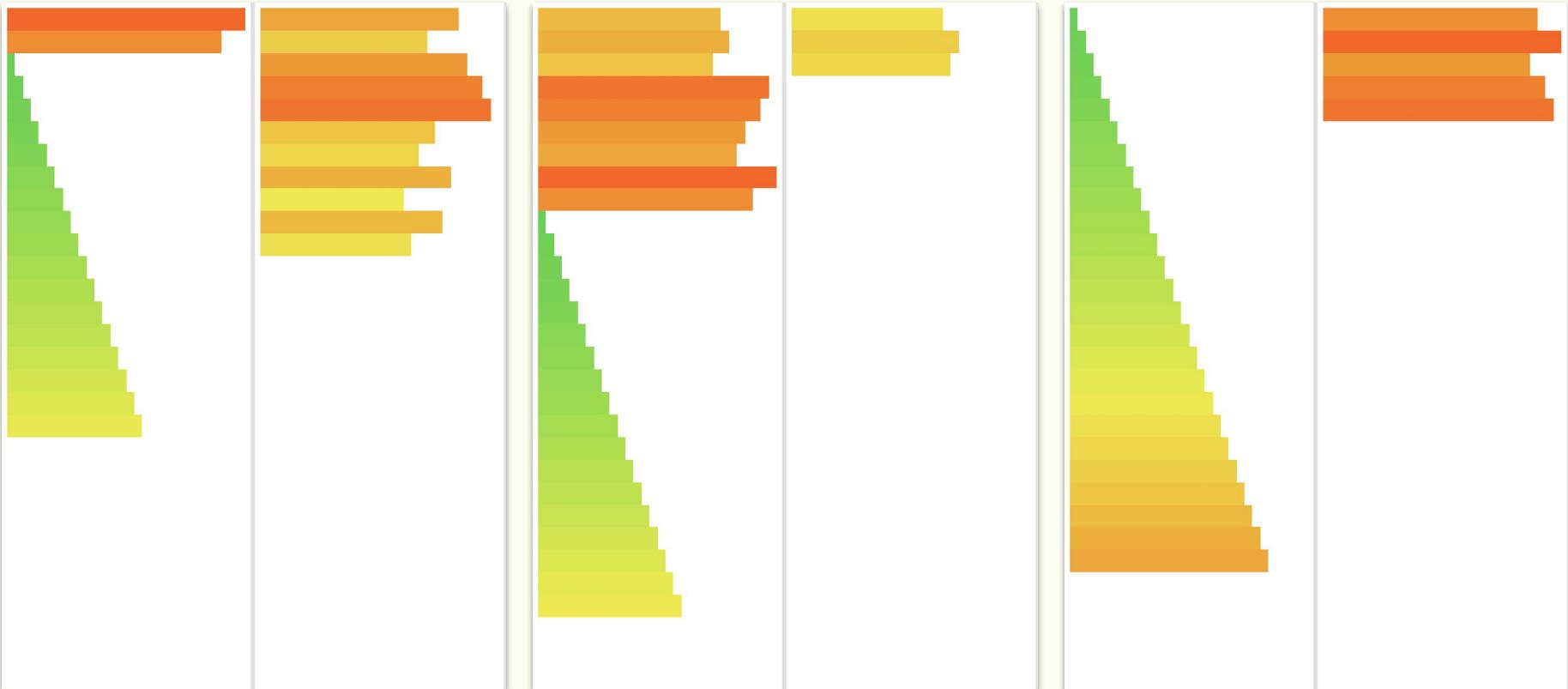
3. bを分割できなくなるまで  
aへpush



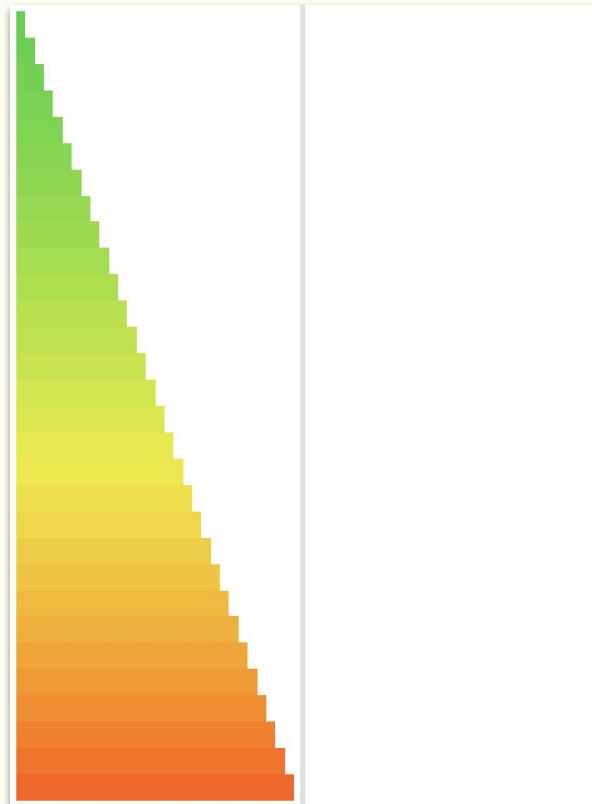
4. bをaの末尾へ



5. aからbへpushして戻す



bへpush, aへpush, aの末尾へ追加, を繰り返して.....



最終的に全てがソートされる

# 細かな最適化

- 分割するときに小さい数字を見つけたら、aの末尾へ
- まとめられる命令をまとめる
  - saとsbならssにする
  - paとpbなら削除する

# 参考書籍・URL

## アルゴリズム

- [『プログラミングコンテスト攻略のためのアルゴリズムとデータ構造』](#)
- [『アルゴリズム・イントロダクション 第3版 第1巻』](#)

## push\_swap

- [Push Swap: The least amount of moves with two stacks | by Jamie Dawson | Medium](#)

## ビジュアライザー

- [elijahkash/push\\_swap\\_gui: Implementation of push-swap \(42-school project\) on python with GUI.](#)
  - p.21 以降の画像として使用しました

# ありがとうございました！

不明点などありましたら、Discordにてお願いします