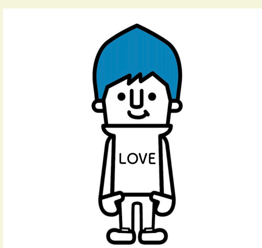
〜 ファーストサークル全課題解説フェス 〜

PUSH_SWAP 編

本題の前に
軽く自己紹介



- エラブ タカヤ(ID:terabu、愛称:ラブ)
- Piscine:2022-08 / 入学:2022-10
- 校舎出現率：週 5
- 職業：無職
- 特記事項：
  自分から声をかけるのは苦手ですが、
  声をかけてくれると喜びます

本題

前半

1. 課題の概要
2. 何から始めるのか
3. クリアまでの道筋

後半

push_swap で考える分割統治法(再帰)

前半

1. 課題の概要
2. 何から始めるのか
3. クリアまでの道筋

→ これらをまとめた資料が既に存在しています

push_swap を 理解するためのスライド

created by nafuka さん

クリアまでの道筋（補足）

基本方針
小さく始めよう！

何はともあれ
2 個のケースから

1. ３個ケースの補足
2. ６個以下ケースの補足
3. ７個以上ケースについて
4. 入力チェックを忘れずに

1️⃣ ３個ケースの補足

6 パターン

- 1 2 3 -> x (ソート済み)
- 1 3 2 -> 2 1 3 (rra) -> 1 2 3 (sa)
- 2 1 3 -> 1 2 3 (sa)
- 2 3 1 -> 1 2 3 (rra)
- 3 1 2 -> 1 2 3 (ra)
- 3 2 1 -> 2 1 3 (ra) -> 1 2 3 (sa)

2️⃣ ６個以下ケースの補足

なぜ６個？

- レビューで局所最適化が求められる
- ３個ケースが流用できる丁度良い数

3️⃣ ７個以上ケースについて

データ構造とアルゴリズムを選択

データ構造の種類

- 配列
- 単方向リスト
- 双方向リスト(循環・非循環)

push_swap で考えるデータ構造

created by terabu

アルゴリズムの種類

- 基数ソート
- クイックソート(系)
- 独自アルゴリズム(push_swap 特化型)

クイックソート「系」とした理由

- 人によって実装・分割の仕方が微妙に違う
- 課題の要件上、本来のクイックソートとは異なるものになる
    - 領域が２つのみ
    - 命令の制約

クイックソートを選択する場合は
ソート前に座標圧縮をするのがオススメ！

座標圧縮とは

それぞれの要素が

「全体で何番目に小さいか」

を求めていく作業

例

```
8, 100, 33, 12, 6, 1211
```

↓

```
1, 4, 3, 2, 0, 5
```

なぜ必要？

ピボット(分割の軸)を

決めるのが楽になる

クイックソート（分割統治法）の手法
→ 後半でお話しします

4️⃣ 入力チェックを忘れずに

- 引数が 0, 1 個
- 整数外（1 a 2）
- int 外（2147483648 -2147483649）
- ソート済み（1 2 3）
- 重複している（3 2 2 1）
- etc...

入力チェックの考慮漏れが原因で
落とされるケースがほとんどです

（課題をちゃんと読みましょう！）

前半のまとめ

- 「push_swap を 理解するためのスライド」を読もう！
- 小さく始めてみよう！

前半は以上となります
ありがとうございました！

後半

push_swap で考える分割統治法(再帰)

何を話すのか？

「push_swap を 理解するためのスライド」

分割統治法(クイックソート)を深掘り

何故話すのか？

- 自分が躓いたところ
- アルゴリズムのエッセンスが詰まっている

解説の前に

良いスコアを目指す内容ではありません🙇

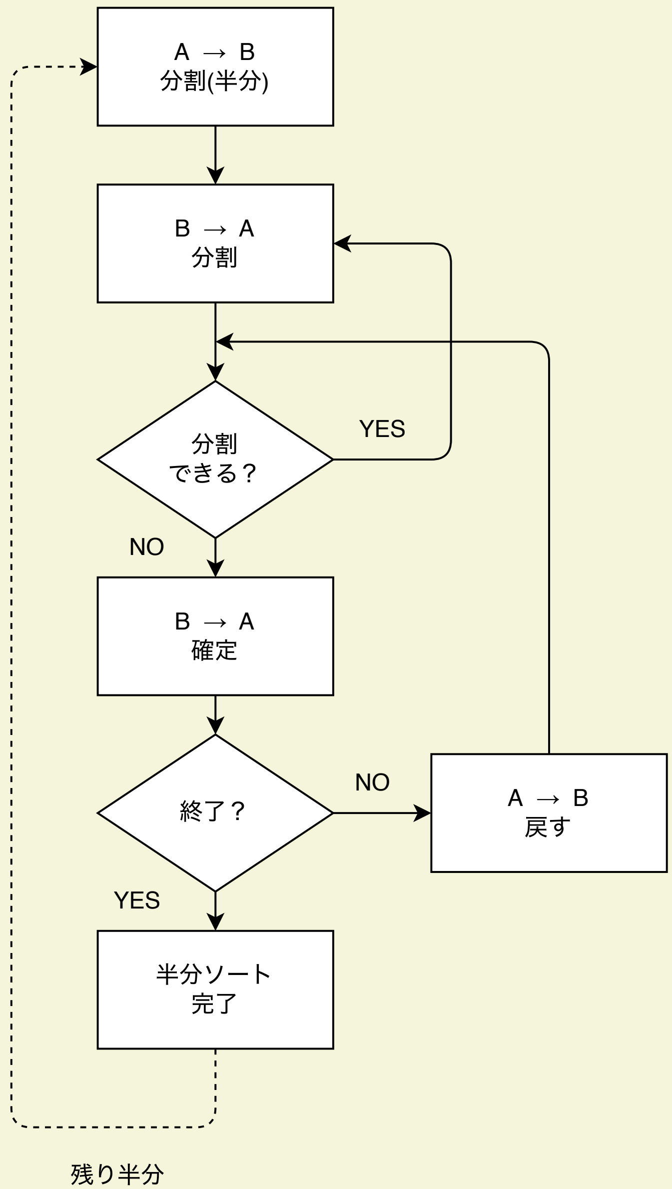
解説始めます！

解説の流れ

1. 処理の流れを考えよう！
2. 具体的な値で試してみよう！
3. 機能ごとに整理しよう！
4. 再帰部分＆パラメーターに注目しよう！
5. （時間があれば）コードを見てみよう！

1️⃣ 処理の流れを考えよう！

簡易フローチャート

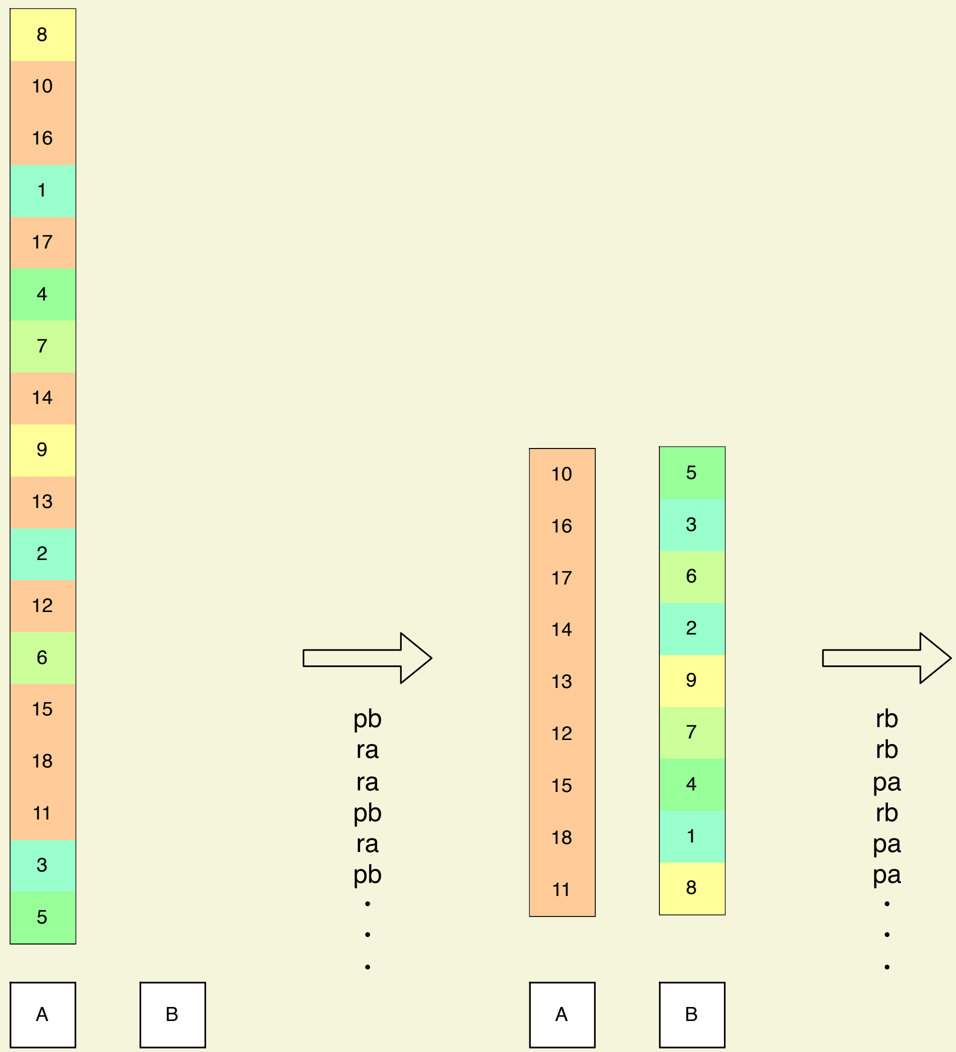


2️⃣ 具体的な値で試してみよう！

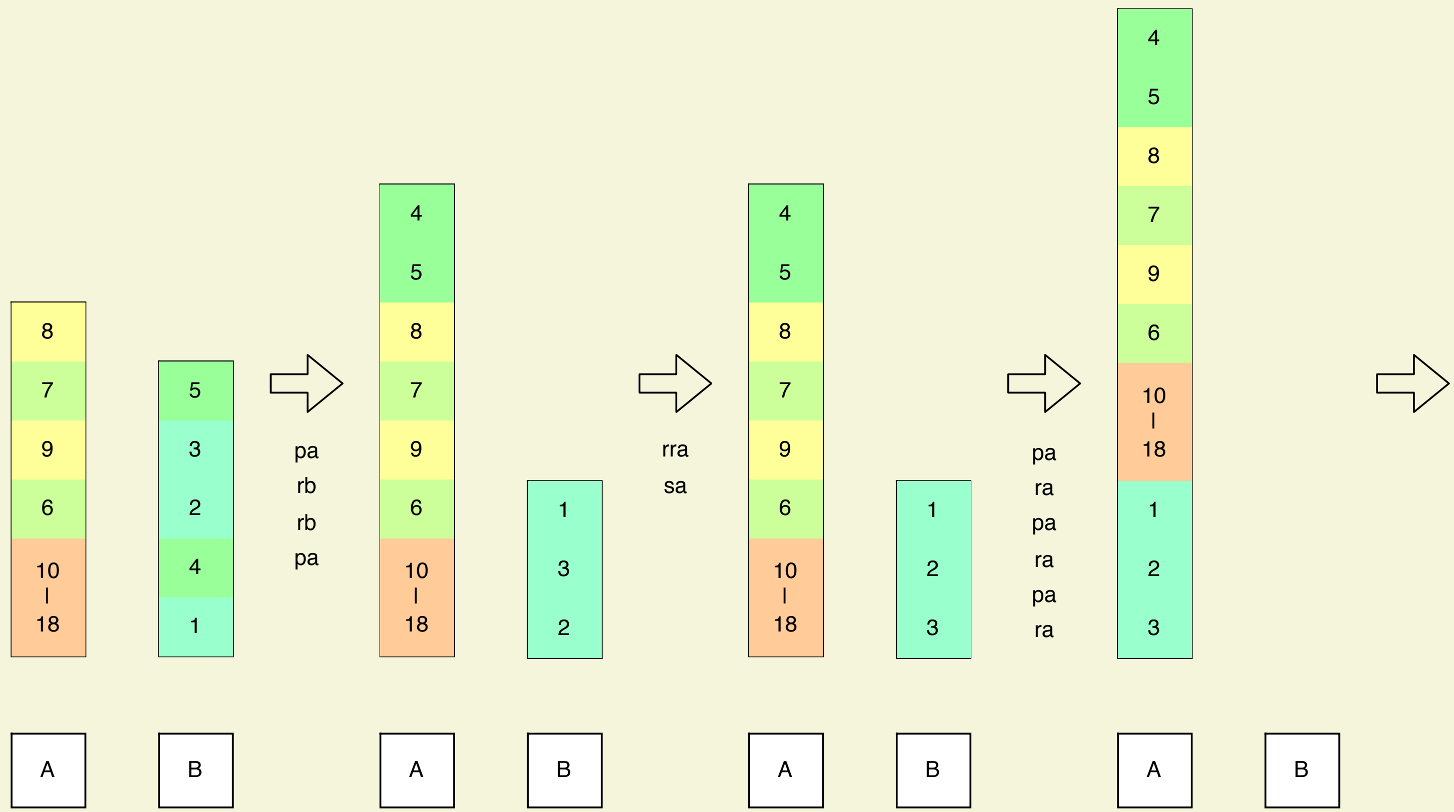
分割の条件

- 分割できる：B が４個以上の時
- 分割できない：B が３個以下の時

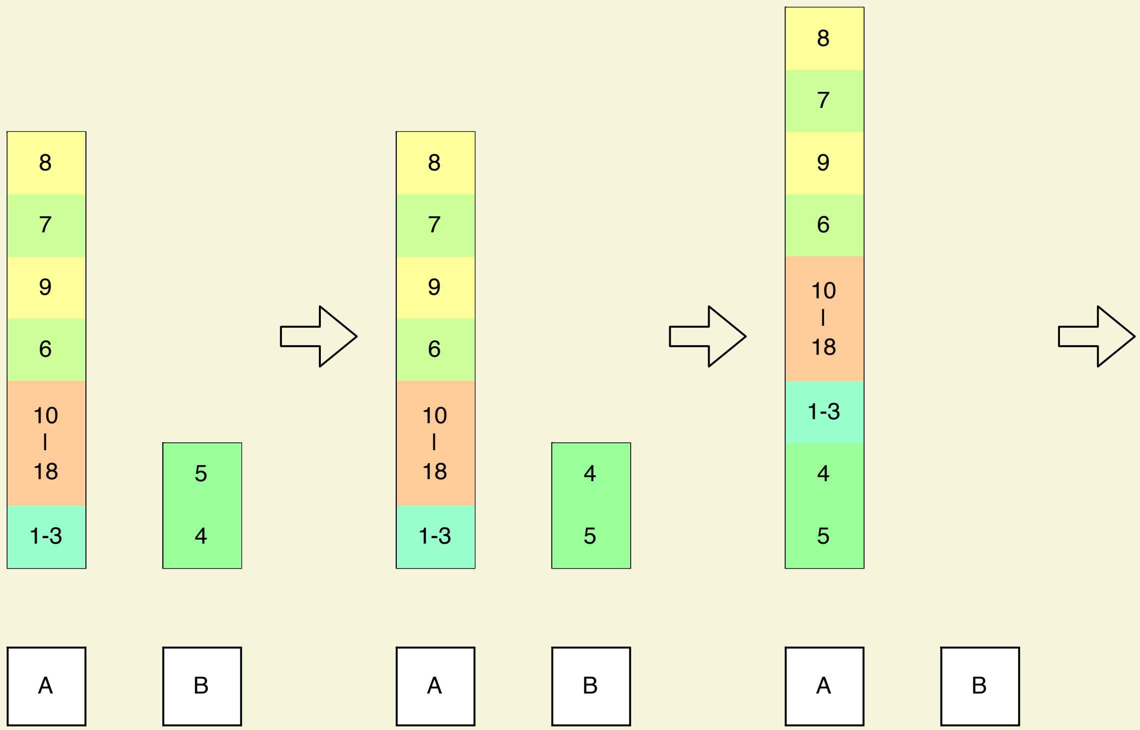
A の要素半分を B に移動



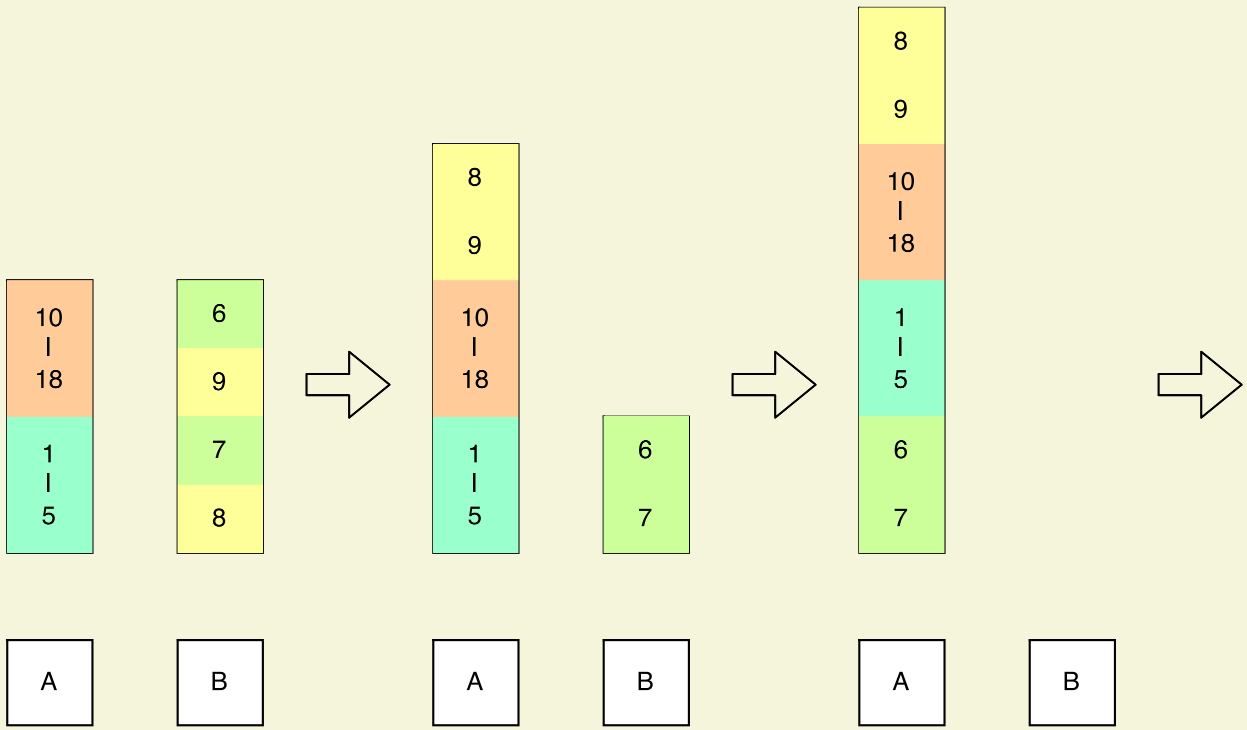
１ ２ ３をソートして確定



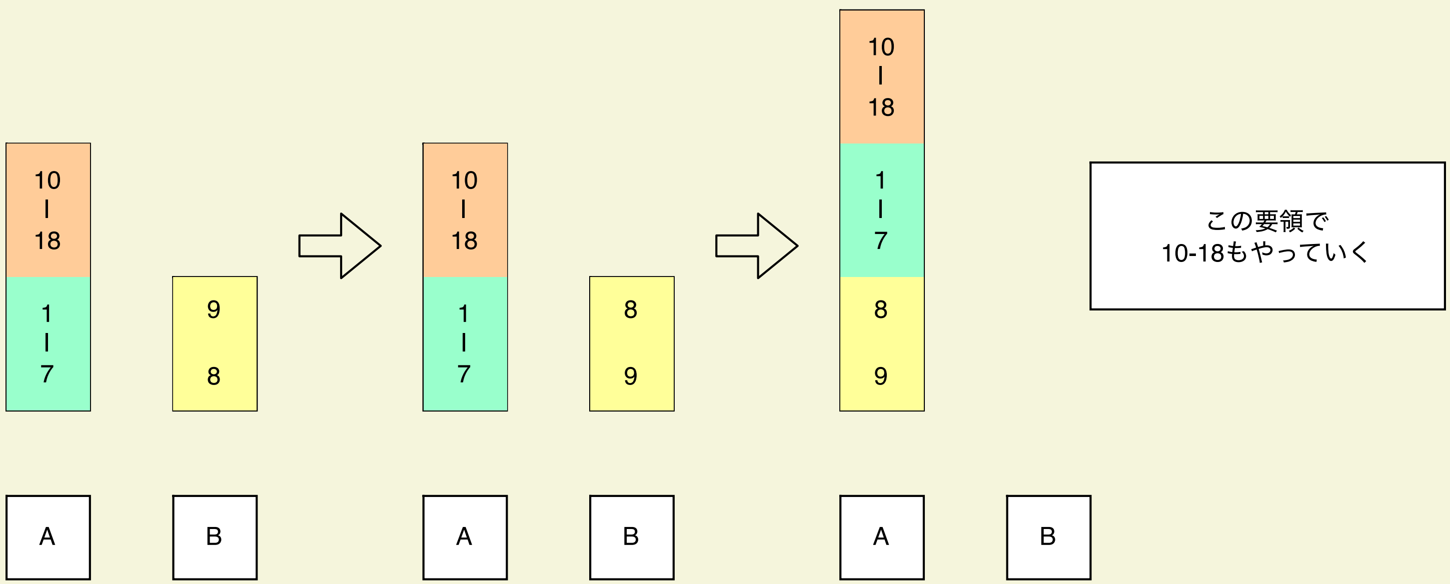
４ ５をソートして確定



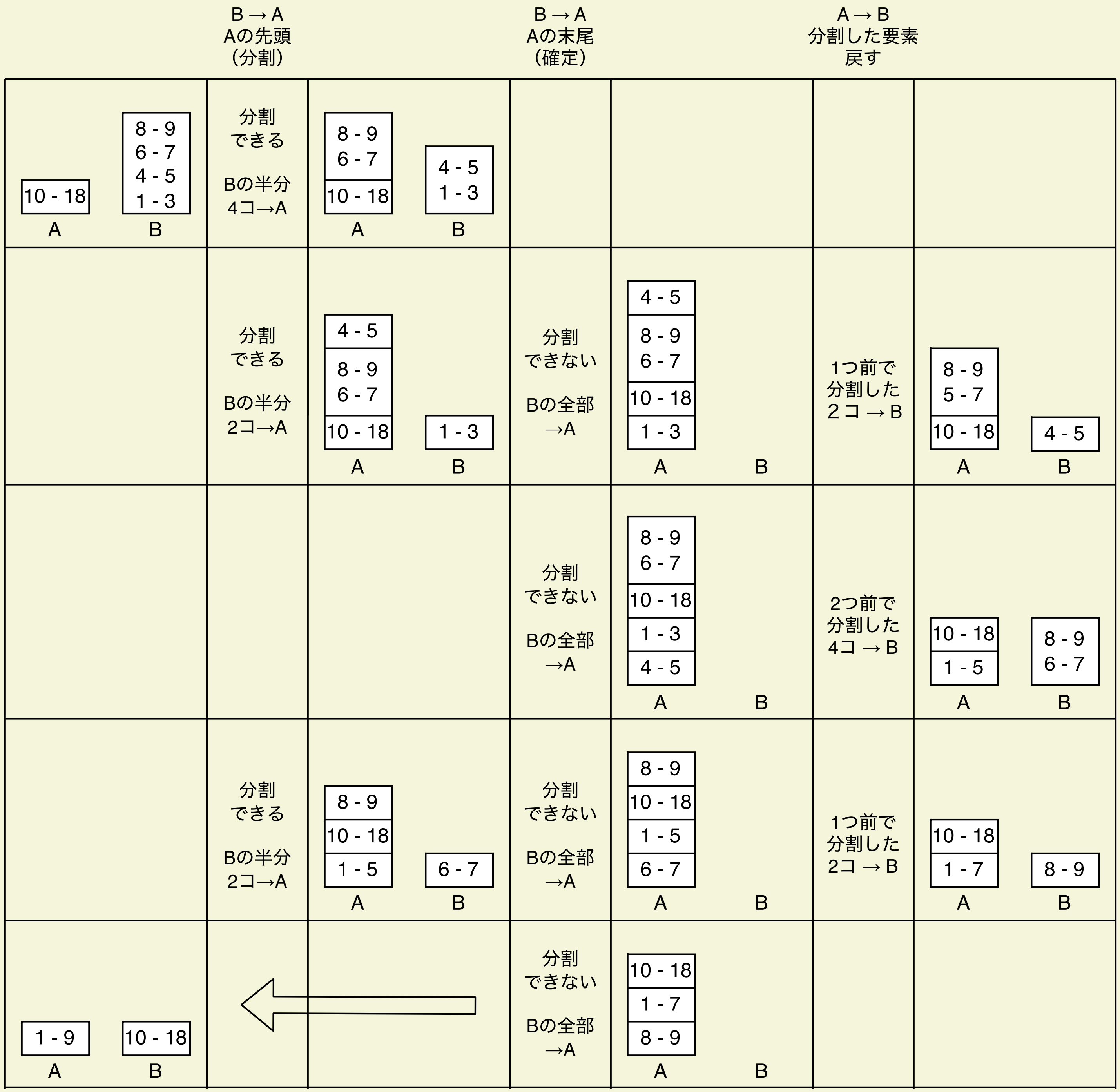
６ ７をソートして確定



８ ９をソートして確定



3️⃣ 機能ごとに整理しよう！


B → A
Aの先頭
（分割）
B → A
Aの末尾
（確定）
A → B
分割した要素
戻す
10 - 18
A
8 - 9
6 - 7
4 - 5
1 - 3
B
分割
できる
Bの半分
4コ→A
8 - 9
6 - 7
10 - 18
A
4 - 5
1 - 3
B
分割
できる
Bの半分
2コ→A
4 - 5
8 - 9
6 - 7
10 - 18
A
1 - 3
B
分割
できない
Bの全部
→A
4 - 5
8 - 9
6 - 7
10 - 18
1 - 3
A
B
1つ前で
分割した
2 コ → B
8 - 9
5 - 7
10 - 18
A
4 - 5
B
分割
できない
Bの全部
→A
8 - 9
6 - 7
10 - 18
1 - 3
4 - 5
A
B
2つ前で
分割した
4コ → B
10 - 18
1 - 5
A
8 - 9
6 - 7
B
分割
できる
Bの半分
2コ→A
8 - 9
10 - 18
1 - 5
A
6 - 7
B
分割
できない
Bの全部
→A
8 - 9
10 - 18
1 - 5
6 - 7
A
B
1つ前で
分割した
2コ → B
10 - 18
1 - 7
A
8 - 9
B
1 - 9
A
10 - 18
B
分割
できない
Bの全部
→A
10 - 18
1 - 7
8 - 9
A
B

ポイントは？

A → B で
何個戻せば良いか
↓
A に push した数量を
どうやって戻すか

例えば、16 → 8 → 4 → 2 で分割されたら
2 → 4 → 8 → 16 の順で戻す必要がある

※正確には 4、8、16 で戻したらそれぞれでさらに分割されるのでもっと複雑

解決方法

1. 再帰関数
2. スタック領域

今回は再帰関数で考えます

ところで皆さん

再帰関数は苦手ですか？

ハーイ！

🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️
🙋‍♀️🙋‍♂️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️
🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️🙋‍♂️🙋‍♀️

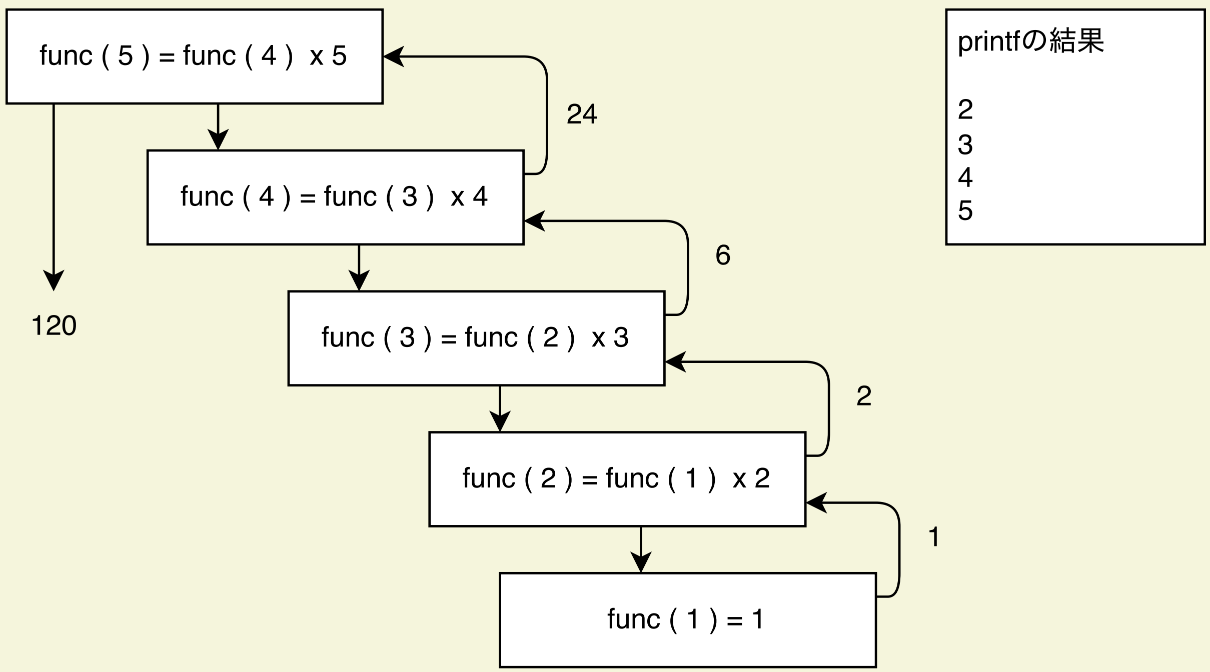
再帰のおさらい

再帰関数の例

```
int func(int N)
{
    int A;

    if (N == 1)
        return (1);
    A = func(N - 1) * N; // 再帰呼び出し
    printf("%d\n", N);
    return (A);
}
```

処理の流れと printf の結果

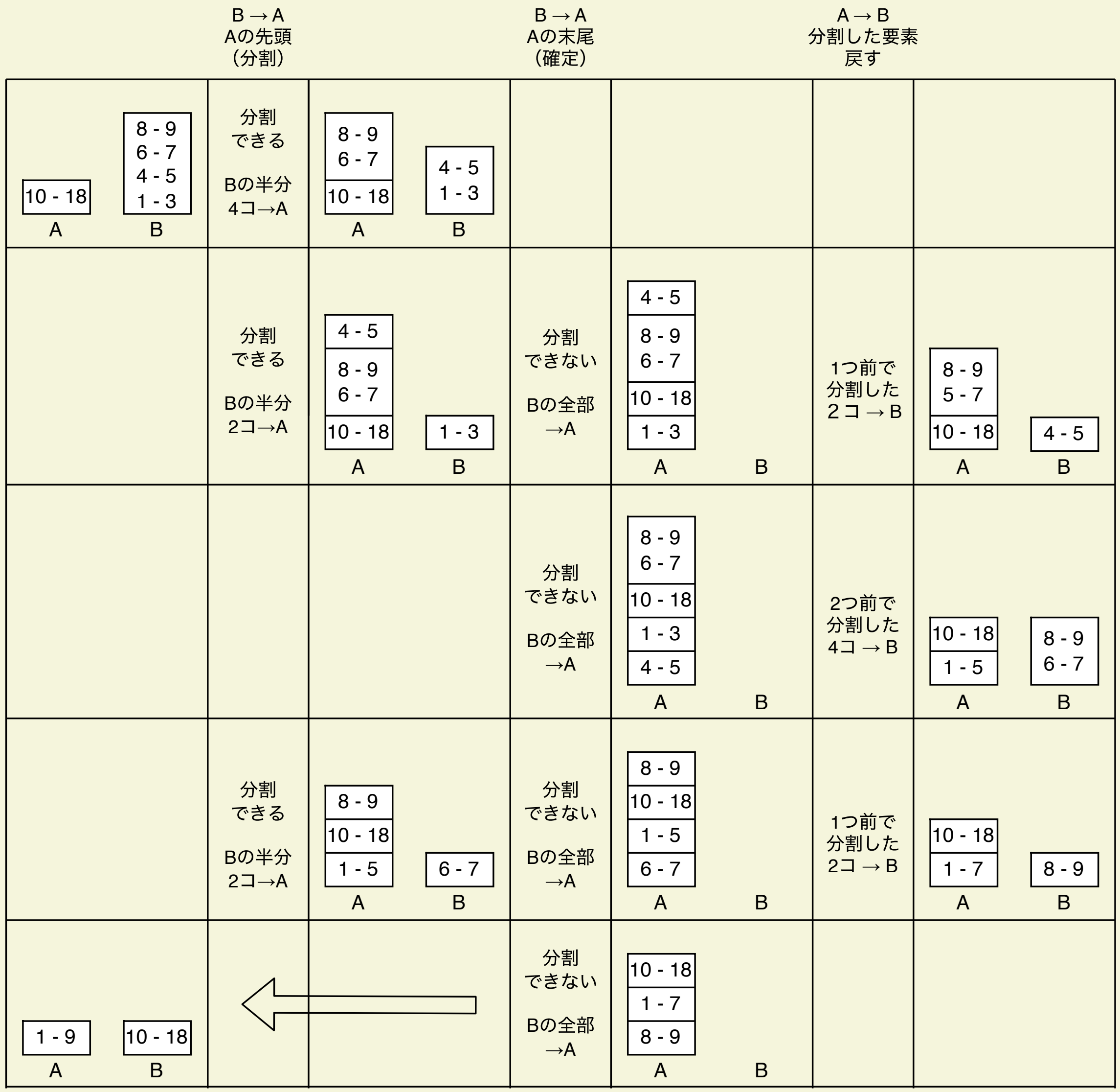


「戻る」という性質を活用して
A にいくつ push したか記録する

どう実現させるか？

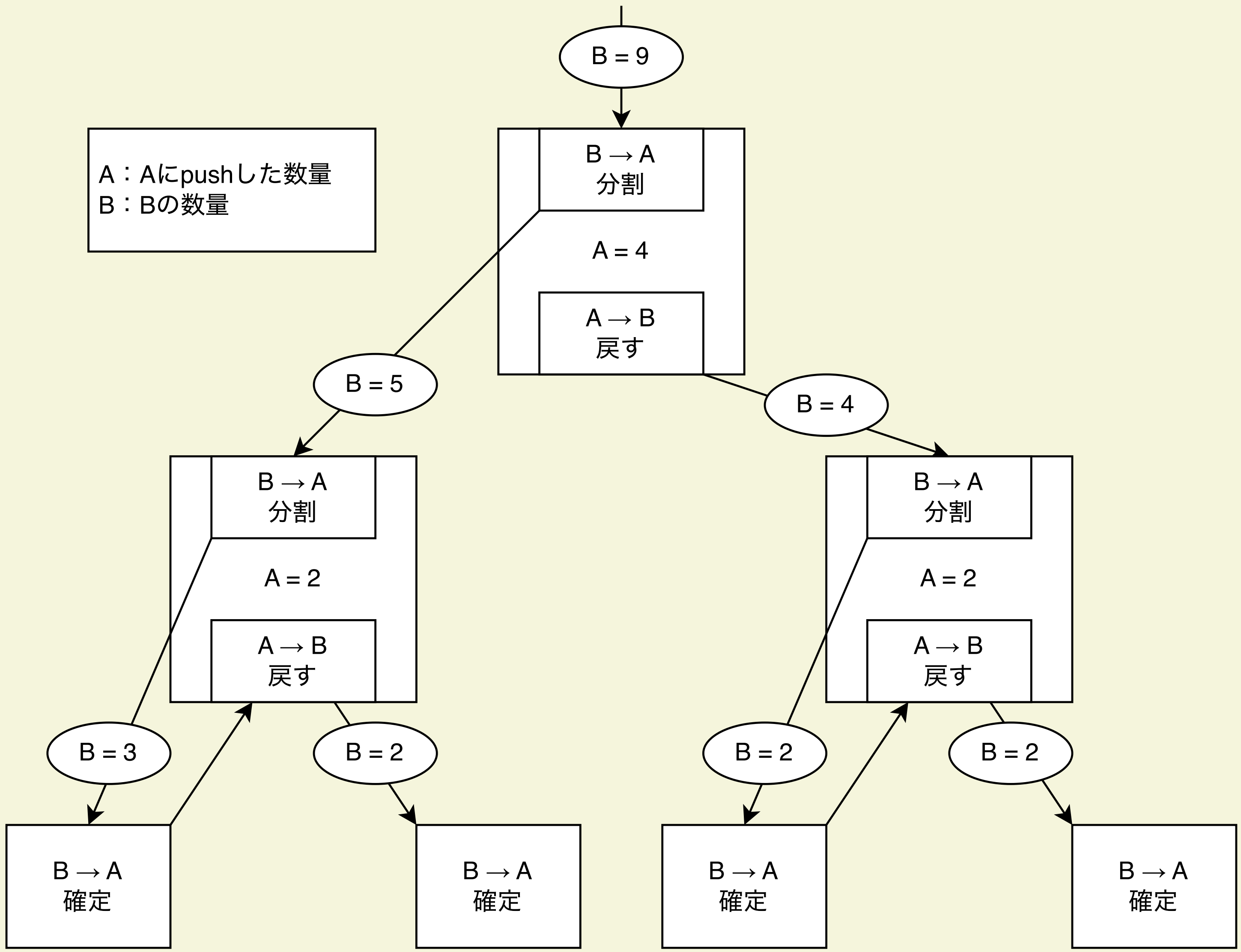
B → A(分割)と A → B(戻す)という処理を
1 つの関数にまとめて再帰させる

（再喝）



B → A
Aの先頭
（分割）
B → A
Aの末尾
（確定）
A → B
分割した要素
戻す
10 - 18
A
8 - 9
6 - 7
4 - 5
1 - 3
B
分割
できる
Bの半分
4コ→A
8 - 9
6 - 7
10 - 18
A
4 - 5
1 - 3
B
分割
できる
Bの半分
2コ→A
4 - 5
8 - 9
6 - 7
10 - 18
A
1 - 3
B
分割
できない
Bの全部
→A
4 - 5
8 - 9
6 - 7
10 - 18
1 - 3
A
B
1つ前で
分割した
2 コ → B
8 - 9
5 - 7
10 - 18
A
4 - 5
B
分割
できない
Bの全部
→A
8 - 9
6 - 7
10 - 18
1 - 3
4 - 5
A
B
2つ前で
分割した
4コ → B
10 - 18
1 - 5
A
8 - 9
6 - 7
B
分割
できる
Bの半分
2コ→A
8 - 9
10 - 18
1 - 5
A
6 - 7
B
分割
できない
Bの全部
→A
8 - 9
10 - 18
1 - 5
6 - 7
A
B
1つ前で
分割した
2コ → B
10 - 18
1 - 7
A
8 - 9
B
1 - 9
A
10 - 18
B
分割
できない
Bの全部
→A
10 - 18
1 - 7
8 - 9
A
B

4️⃣ 再帰＆パラメーターに注目しよう！


A：Aにpushした数量
B：Bの数量
B = 9
B → A
分割
A = 4
A → B
戻す
B = 5
B = 4
B → A
分割
A = 2
A → B
戻す
B → A
分割
A = 2
A → B
戻す
B = 3
B = 2
B = 2
B = 2
B → A
確定
B → A
確定
B → A
確定
B → A
確定

5️⃣ （時間があれば）コードを見てみよう！

後半のまとめ

分割統治法の実現方法について
ざっくりした処理フロー → コード化まで

今回、一番伝えたかったこと

いろんな切り口で考えてみよう！

- 処理の流れは？
- 具体的な値を入れてみたらどう動く？
- 抽象化・共通化できる部分はある？
- パラメーターはどのように変化する？

質疑応答
(...時間があれば)

ご清聴ありがとうございました！