

選択ソート (Selection Sort) 完全図解

初心者でもわかる、いちばんシンプルな並び替えアルゴリズム



一番小さいものを探して、左に詰めていく。

考え方は非常にシンプルです。

選択ソートの仕組みは、たったこれだけのルールで動いています。



3. 進める（「最小値が
確定した部分」を一つ
右へ進める）



1. 探す（リスト全体から
最小値を見つける）

2. 入れ替える（見つけた最小値と
、未整列部分の先頭を交換する）

実践：初期状態

実際のデータを使って動かしてみましょう。初期データはこの5つです。
まだ何も並び替えられていないため、すべて「未整列」の状態です。



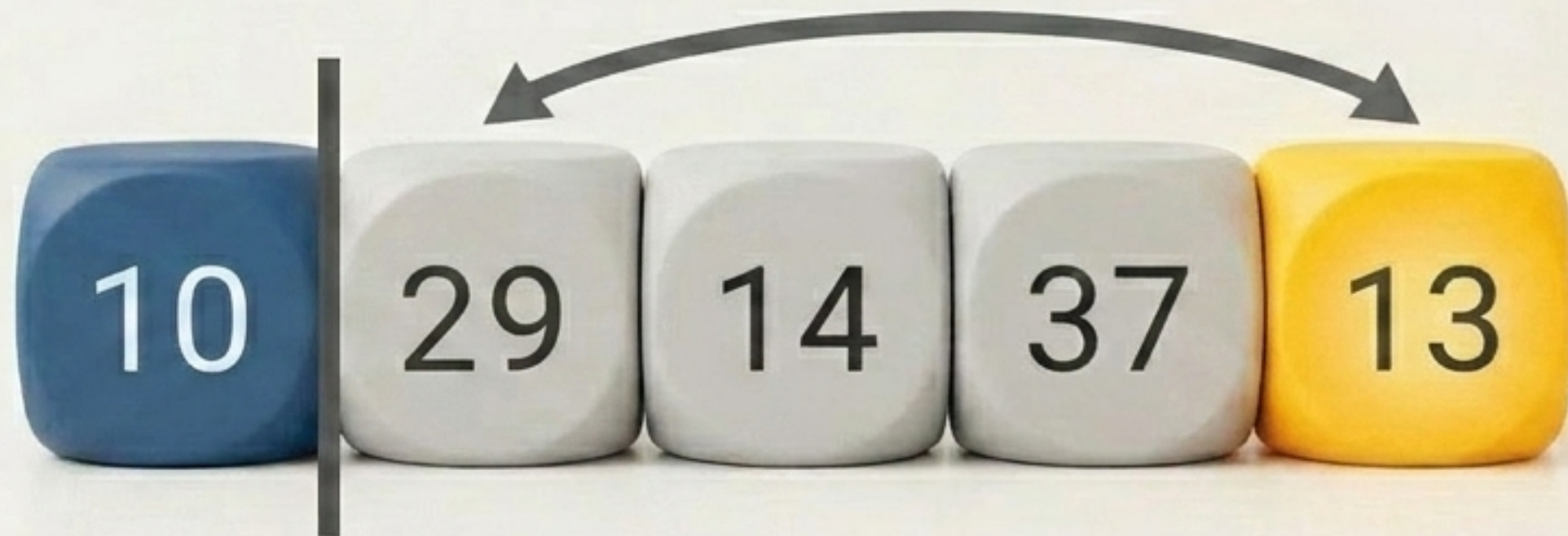
ステップ1：最初の最小値を探す

リスト全体の中から一番小さい「10」を見つけます。見つかった最小値と、未整列部分の先頭「29」を入れ替えます。



ステップ2：確定ラインを右へ進める

「10」が最小値として確定しました。確定ラインを右にズラし、残りの未整列部分から次の最小値「13」を探して、先頭の「29」と入れ替えます。



ステップ3&4：繰り返しの加速

確定ラインがどんどん右へ進みます。最小値がすでに未整列部分の先頭にある場合（14のケース）は、そのままの位置で確定します。



ソート完了

未整列の要素がなくなり、すべての並び替えが完了しました。これが選択ソートの動きの全貌です。



```
data = [29, 10, 14, 37, 13]  
# ソート後: [10, 13, 14, 29, 37]
```

動きをコードに翻訳する

先ほどのブロックの動きは、Pythonコードではこのように表現されます。外側のループが「境界線の移動」、内側のループが「最小値の探索」を担当しています。

```
def selection_sort(arr):
```

```
    n = len(arr)
```

```
    for i in range(n):
```

```
        min_idx = i
```

```
        for j in range(i + 1, n):
```

```
            if arr[j] < arr[min_idx]:
```

```
                min_idx = j
```



確定ラインを右に進める



未整列部分の先頭を
「暫定の最小値」とする



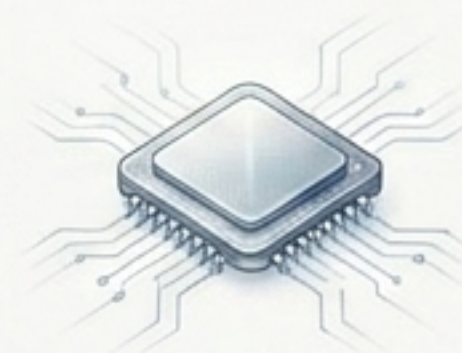
残りの要素から
「本当の最小値」を探す

性能評価ダッシュボード



時間計算量: $O(n^2)$

どんなデータ状態（既に整列されていても、逆順でも）であっても、必ず最後まで比較を行うため、データが増えると時間がかかります。

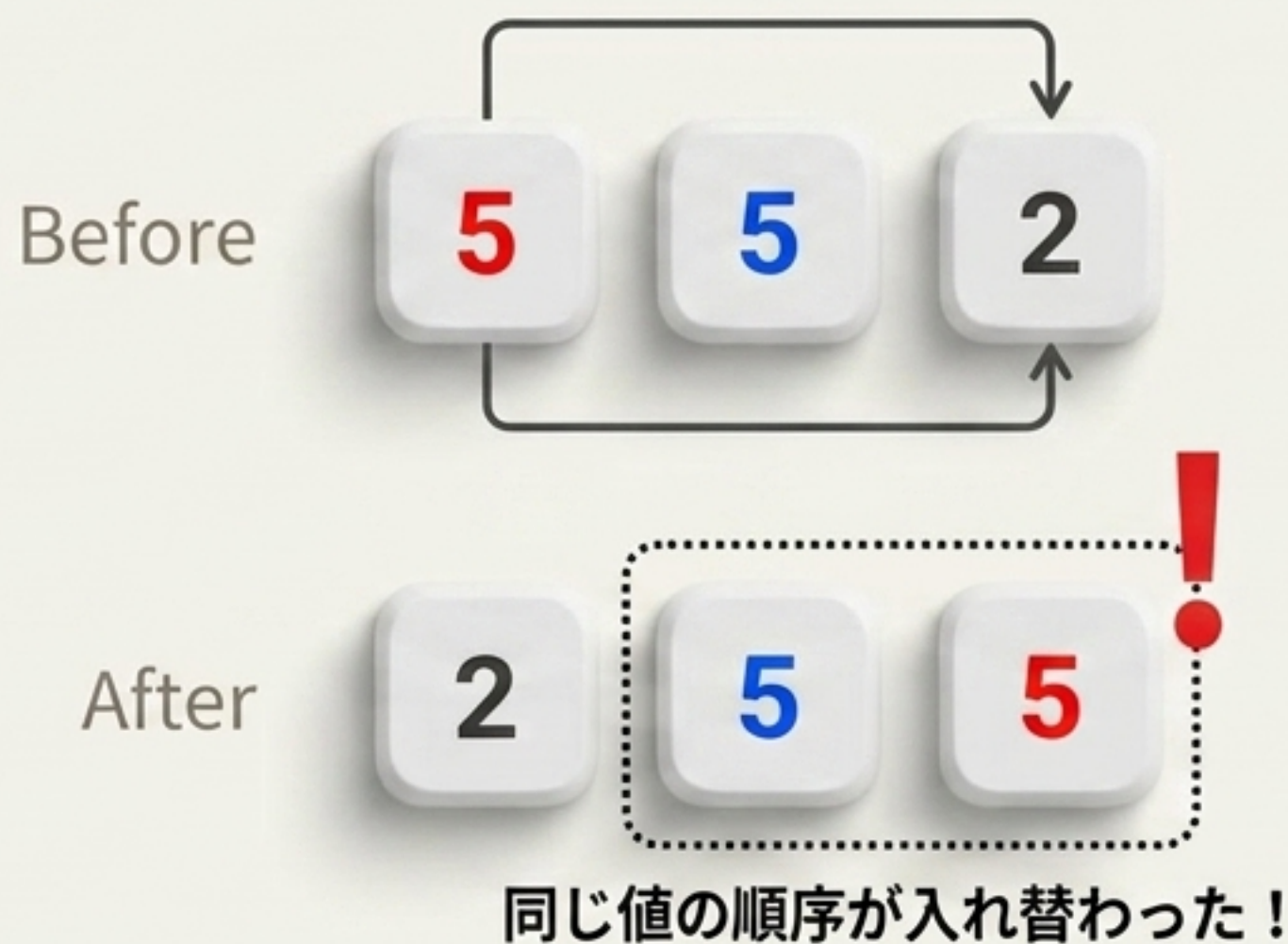


空間計算量: $O(1)$

仕組みが簡単のため、追加のメモリ領域をほぼ使いません（省スペース）。

注意点：「不安定なソート」である理由

選択ソートは「不安定（Unstable）」なアルゴリズムです。離れた場所にある要素同士を大きく入れ替えるため、同じ値を持つ要素の「元々の前後関係」が崩れる可能性があります。



まとめ：選択ソートの要点

- **直感的な動き**：まだ並び替えていない部分から一番小さいものを探し、左に詰めていく。
- **特徴**：仕組みもコードも直感的で非常にシンプル。
- **パフォーマンス**：速度は遅い（時間計算量 $O(n^2)$ ）が、メモリは消費しない（空間計算量 $O(1)$ ）。
- **注意点**：同じ値の要素の前後関係が入れ替わる可能性がある（不安定なソート）。

