

01-01 並び替えのアルゴリズムと実装

例えば、次のような表では、どのような仕組みで「昇順」「降順」への並び替えが行われるのだろうか。

送信者 ▼	件名 ▼	受信日時 ▼
宮下武	次回会合について	6/1/2017 10:05
川田美香	スケジュールのご確認	5/30/2017 18:01
野崎順子	昨日のお礼	5/30/2017 9:39
浅田酒店	ご注文の品について	5/29/2017 11:45
赤井直紀	Re: アルゴリズムの質問	5/25/2017 13:22
秋山裕子	ご協力のお願い	5/24/2017 12:57
浅田酒店	新酒入荷しました	5/21/2017 16:10
安達裕也	セミナーのお知らせ	5/20/2017 15:02

◇ 基本選択法（選択ソート）

- 最小選択法
- 最大選択法

【最小選択法の実装例】

1. 比較基準値を決める。
2. 最初の数値を比較基準値とし、n個の中から最も小さい数字を探し、それと入れ替える。
3. 次に、残りのn-1個の中から最も小さい数字を探し、それを2番目の数字と入れ替える。
4. この処理をn-1回繰り返す。

```
function minSelectSort(Array $array): Array
{
    // 比較基準値を固定し、それ以外の数値と比べていく。
    for($i = 0; $i < count($array)-1; $i++){

        // 比較基準値を仮の最小値として定義。
        $min = $array[$i];

        // 比較基準値の位置を定義
        $position = $i;

        // 比較基準値の位置以降で、数値を固定し、順番に評価していく。
        for($j = $position + 1; $j < count($array); $j++){

            // 比較基準値の位置以降に小さい数値があれば、比較基準値と最小値を更新。
            if($min > $array[$j]){
                $position = $j;
                $min = $array[$j];
            }
        }
    }
}
```

```

    }

    // 比較基準値の位置が更新されていなかった場合、親のfor文から抜ける。
    if($i == $position){
        break;
    }

    // 親のfor文の最小値を更新。
    $array[$i] = $min;

    // 次に2番目を比較基準値とし、同じ処理を繰り返していく。
}
return $array;
}

```

```

// 実際に試してみる。
$array = array(10,2,12,7,16,8,13)
$result = selectSort($array);
var_dump($result);

// 昇順に並び替えられている。
2, 7, 8, 10, 12, 13, 16

```

【アルゴリズム解説】

データ中の最小値を求め、次にそれを除いた部分の中から最小値を求める。この操作を繰り返していく。

02



数列を線形探索し、最小値を探します。最小値1が見つかりました（線形探索は、3-1節で解説しています）。



03



最小値の1を列の左端の6と交換し、ソート済みにします。なお、最小値がすでに左端であった場合には、何の操作も行いません。



05



2を左から2番目の6と交換し、ソート済みにします。





ソートが完了しました。

◇ クイックソート

【実装例】

1. 適当な値を基準値（Pivot）とする（※できれば中央値が望ましい）
2. Pivotより小さい数を前方、大きい数を後方に分割する。
3. 二分割された各々のデータを、それぞれソートする。
4. ソートを繰り返し実行する。

```
function quickSort(Array $array): Array
{
    // 配列の要素数一つしかない場合、クイックソートする必要がないので、返却する。
    if (count($array) <= 1) {
        return $array;
    }

    // 一番最初の値をPivotとする。
    $pivot = array_shift($array);

    // グループを定義
    $left = $right = [];

    foreach ($array as $value) {
        if ($value < $pivot) {
            // Pivotより小さい数は左グループに格納
            $left[] = $value;
        } else {
            // Pivotより大きい数は右グループに格納
            $right[] = $value;
        }
    }
}
```

```
// 処理の周回ごとに、結果の配列を結合。
return array_merge
(
    // 左のグループを再帰的にクイックソート。
    quickSort($left),

    // Pivotを結果に組み込む。
    array($pivot),

    // 右のグループを再帰的にクイックソート。
    quickSort($right)
);
}
```

```
// 実際に試してみる。
$array = array(6, 4, 3, 7, 8, 5, 2, 9, 1);
$result = quickSort($array);
var_dump($result);

// 昇順に並び替えられている。
1, 2, 3, 4, 5, 6, 7, 8
```

【アルゴリズム解説】

適当な値を基準値（Pivot）とし、それより小さな値のグループと大きな値のグループに分割する。同様に、両グループの中でPivotを選び、二つのグループに分割する。グループ内の値が一つになるまで、この処理を繰り返していく。

02



基準となる数（ピボット）を、数列の中からランダムに1つ選びます。今回は4が選ばれました。



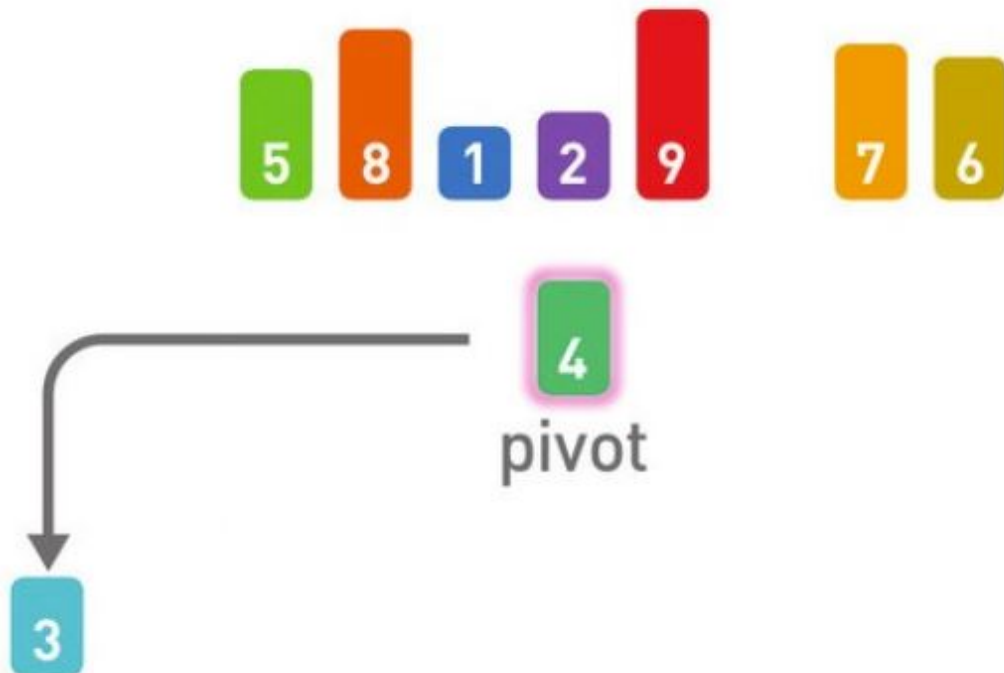
03



ピボット以外の各数字を、ピボットと比較していきます。ピボットより小さい数字は左に、大きい数字は右に移動します。



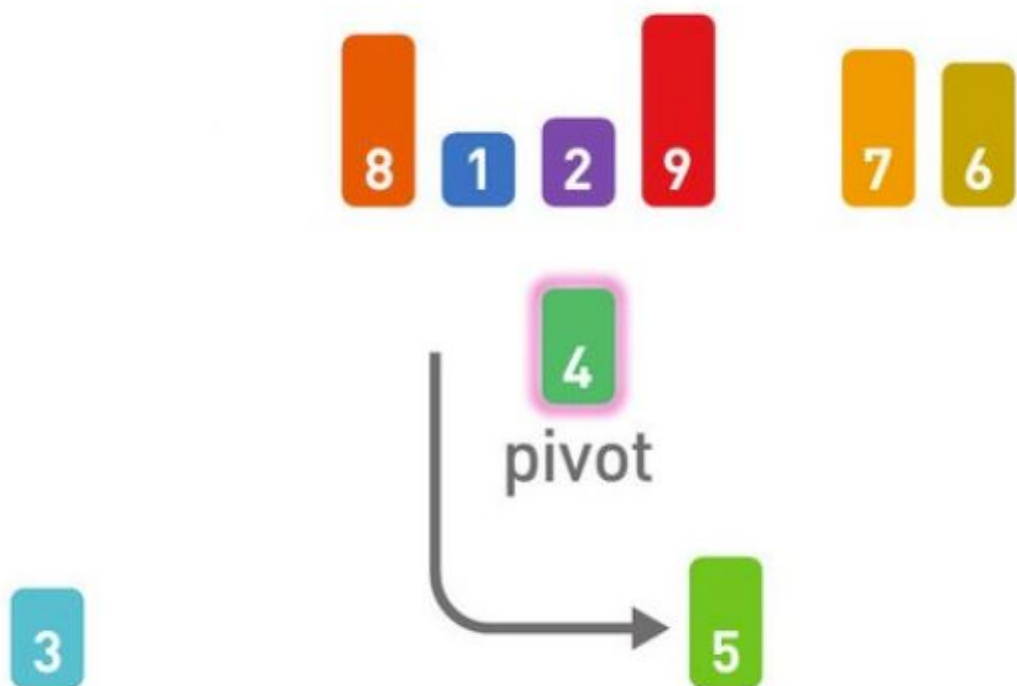
05



$3 < 4$ なので、3は左に移動します。



07



5 > 4なので、5は右に移動します。



08



他の数字も同様に比較し、移動させていくと、このようになります。



10



したがって、左側と右側をそれぞれ独立にソートすれば、全体のソートが完成します。



13

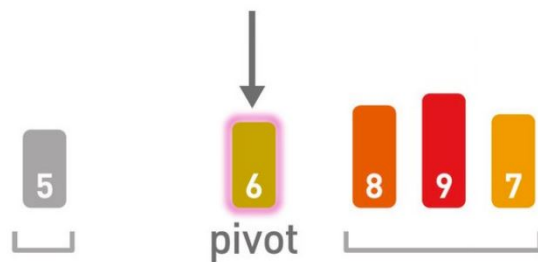


A single yellow bar with the number '6' inside, highlighted with a pink glow. Below it is the word 'pivot'.

ピボットの6とそれぞれの数字を比較し、小さければ左へ、大きければ右へ移動します。



15



先ほどと同様、左右を独立にソートすれば、この部分のソートが完成します。しかし、左側は5のみなので、すでにソート済みです。これ以上やる必要はありません。右側はこれまでと同様に、ピボットを選んでいきます。



16



8がピボットとして選ばれました。



17



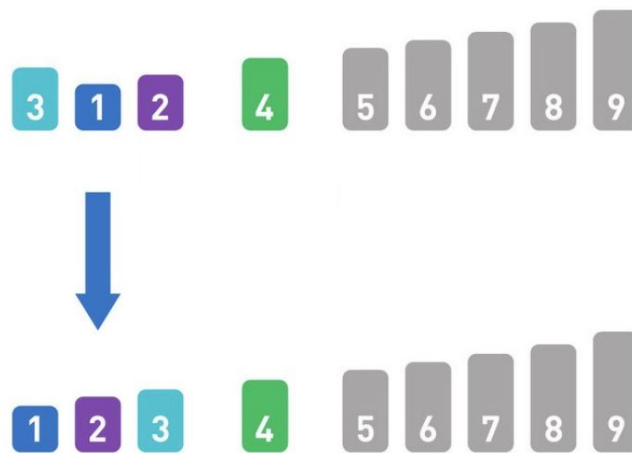
9と7が8と比較されて、左右に振り分けられました。8の左右はどちらも1つの数字しかないので、これで終わりです。これで、7 8 9がソート済みになりました。





その結果、最初のピボットの4の右側は、ソートが終了します。





左側も同様にソートしていくと、全体のソートが完了します。

◇ 基本交換法（バブルソート）

隣り合ったデータの比較と入替えを繰り返すことによって、小さな値のデータを次第に端のほうに移していく方法。

01



数列の右端に天秤を置き、天秤の左右の数字を比較します。比較した結果、右の数字の方が小さければ入れ替えます。



03



比較が完了すると天秤を1つ左に移動し、同様に数字を比較します。今回は $4 < 6$ なので、数字は入れ替えません。



05



並べ替えを繰り返し、天秤が左端に到達しました。一連の操作で、数列の中で最も小さい数字が左端に移動したことになります。



07



天秤を右端に戻します。先ほどと同様の操作を、天秤が左から2番目に到達するまで繰り返します。



09



天秤を右端に戻します。同様の操作を、すべての数字がソート済みになるまで繰り返します。



12



ソートが完了しました。

◇ 基本挿入法（挿入ソート）

既に整列済みのデータ列の正しい位置に、データを追加する操作を繰り返していく方法。

◇ ヒープソート

◇ シェルソート

01-02. 探索のアルゴリズムと実装

◇ 線形探索法

今回は「6」を探す。

02



まず、配列の左端の数字を調べます。6と比較し、一致すれば探索を終了します。一致しなければ、1つ右の数字を調べます。



04



6が見つかるまで比較を繰り返します。



05



6が見つかったので探索を終了します。

◇ 二分探索法

前提として、ソートによって、すでにデータが整列させられているとする。今回は「6」を探す。

02



まず、配列の真ん中にある数を調べます。この場合は5になります。



03



5と、探索する数である6を比較します。



04



必要のなくなった数字は候補から外します。



05



残った配列の真ん中にある数を調べます。この場合は7になります。



07



必要のなくなった数字は候補から外します。



08



残った配列の真ん中にある数を調べます。この場合は6になります。

◇ ハッシュ法

