様々なソートアルゴリズム

クリスティアン　ハルジュノ *

2025 年 6 月 17 日

1　初めに

データに最適なソートアルゴリズムを選択するのは容易ではありません. Big-O 記法を用いて理論的にアルゴリズムを比較することは可能ですが, 現実のソートアルゴリズムはそれ以上の要素を含んでいます.

理論的に効率的なアルゴリズムの中には, 実装前に徹底的に調査する必要がある弱点を持つものもあります. 一部のアルゴリズムは, データがアルゴリズムを最適に実行できる特定の状態にある場合にのみ効率的です.

現代のコンピュータは, 大容量メモリ, 高速 CPU, 効率的なストレージなど, 十分なハードウェアスペックを備えていますが, 一部のアルゴリズムは高速実行のためにリソースを犠牲にしています. データサイズが小さい場合は, これは問題にならないかもしれません. しかし, ビッグデータ企業のようにデータを大規模化すると, リソースは適切に対処する必要がある非常に重要な問題になります. そのため, アルゴリズムを理論的に比較するだけでなく, 対象データセットに対してこれらのアルゴリズムをテストし, 結果を分析することが重要です.

本レポートでは, 実装が最もシンプルなものからより複雑なものまで, 最も普及している 6 つのソートアルゴリズムを検証し, 7 つの固定データセットと比較します. 各データセットには, 1 から 50000 までの整数が 10 万行含まれています. 各データセットは, 完全にランダム, 反転, ソート済みなど, さまざまな段階にあります. これらの比較により, 選択した 6 つのソートアルゴリズムの長所, 短所, および動作を検証できます.

*釧路工業高等専門学校情報工学科 5 年情報 21 番

2　背景

このセクションでは, この実験で使用した 6 つのソートアルゴリズムを紹介し, それぞれがどのように機能するかを説明します.

2.1　ソートアルゴリズムを紹介

バブルソート バブルソートは最もよく知られているソートアルゴリズムの一つです. 実装が非常に簡単であり, 少量のデータに対してはメモリ使用量が少なく, 処理も比較的高速です. この方法にはデータを最初から最後まで調べながら, 現在の数値と次の数値を比較することで機能します. 現在の数値が次の数値よりも大きい場合は, 位置を入れ替えます. このプロセスは, データが完全にソートされるまで何度も繰り返されます.

シェーカーソート シェーカーソートまたはカクテルソートはバブルソートと似てるですが, シェーカーソートでは左から右に移動だけでなく逆向きにもう通過する.

挿入ソート 挿入ソートは, 要素を適切な位置に直接挿入することで並べ替えを行うアルゴリズムです.　現在処理している要素を, それより前の要素と右から左に 1 つずつ比較し, 比較対象が現在の要素より大きければ, 右にずらします.　適切な位置が見つかったところで, 現在の要素を挿入します.

バケットソート バケットソートは他方法より速度が最高と言えます. しかし, ソートの高速にはメモり使用量の増加が伴う. バケットソートでは, データを複数のバケットに分割します. 各バケットは, バブルソートや挿入ソートなどの手法を用いてソートされます. すべての要素を指定されたバケットに挿入した後, すべてのバケットを連結してソート済みの配列を作成します. この手法は, 大規模なデータに対して単純なソートアルゴリズムを実行する必要がないため, 高速です. 前述のように, バブルソートなどの一般的なソート手法は, 大規模なデータベースではパフォーマンスが低下します. しかし, 少量のデータであれば問題ありません. データを独自のバケットに分割することで, 1 回の操作で処理しなければならないデータ量が大幅に削減され, アルゴリズムの負荷が軽減されます.

クイックソート クイックソートは分割統治ソートと言われています. このソート方法ではある配列から枢軸を選び, 全要素をその枢軸に比較する. 枢軸より

小さいとその要素を枢軸の左がわにづらす. 枢軸より大きいと, その要素を枢軸の右側にづらす. 枢軸を分岐点とし, 配列を左配列と右配列に分割する. この処理を各文割された配列に対応し, 繰り返して行う. 最小配列に着くと, 各配列をまた分岐点枢軸の左と右側に繋ぎます.

カウントソート バケットソートと似たように, このソート方法でもバケットのような配列を作成する. まず, データ配列内の最大値を探索し, その値をもとにカウント配列（頻度を格納するための配列）を作成する. 次に, 元のデータ配列を 1 つずつ走査し, それぞれの値に対応するインデックスのカウント配列の値をインクリメントすることで, 各値の出現回数を記録する. その後, カウント配列を累積和に変換し, それをもとに元のデータを安定的に新しい配列へと並べ替えていく. このソートは整数値に限定されるが, 非高速でソートできます.

2.2 理論的な比較

表 1, セクション 2.1 で前述した各ソート手法の基本的な特性または動作を示しています. 各アルゴリズムの特性は, 最良の計算複雑度, 平均的な計算量, 最悪の計算量に分類されます. 最良の計算量は, 特定の入力状態においてデータをソートするために必要な最小の演算量を表します. 平均的な計算量は, 考えられるすべての入力を考慮し, データをソートするために必要な予想される演算量を表します. 最悪の計算量は, 特定の入力状態においてデータ処理に必要な最大の時間を表します.

表 1: 各ソートアルゴリズムの計算量と特徴の比較

アルゴリズム	最良計算量	平均計算量	最悪計算量	空間計算量	インプレース
バブルソート	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	はい
シェーカーソート	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	はい
挿入ソート	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	はい
バケットソート	$O(n+k)$	$O(n+k)$	$O(n^2)^\dagger$	$O(n+k)$	いいえ
クイックソート	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)^*$	はい
カウントソート	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$	いいえ

計算量を指定するために, Big O 表記法を使用します. Big O 表記法は, 処理する必要があるデータの量に基づいてソート アルゴリズムのパフォーマンスを評価する最も基本的な方法の 1 つです. 表 1 に示されているように, バブルソート, シェイカーソート, 挿入ソートの平均計算量は $O(n^2)$ です. Big O 表記法では, n はデータ自体の長さに対応します. データのソートに必要な操作数に対して n をプロットすると, 計算量が $O(n^2)$ の図 1 に示すように, ソートするデータの数がわずかに

変化するだけでも操作数が大幅に増加し, 大量のデータのソート効率が低下する可能性があります. しかし, 計算量が $O(n \log n)$ のクイックソートを 1 と比較すると, データ量を大幅に増やした場合でも, 特定のデータをソートするために必要な操作の量は, 複雑度が $O(n^2)$ のアルゴリズムに比べてはるかに少なくなります.

上記のアルゴリズムの最良計算量について、バブルソート、シェイカーソート、挿入ソートはいずれも、与えられた入力データが既にソート済みであることを前提としています。この場合、アルゴリズムはデータを 1 回だけ処理すればよいため、計算量は $O(n^2)$ から $O(n)$ に減少します。バケットソート最良計算量では、すべての要素が各バケットに均等に分散されていると想定されます。クイックソートの最良計算量では、ピボットの選択によって値が左右に均等に分散されると想定されます。最後に、カウントソートの最良計算量では、入力データの整数の範囲が狭いと想定されます。

最後に, 外部配列を用いてデータの内容をソートするバケットソートとカウントソートの場合, 平均的な計算量は $O(n + k)$ で測定されます. ここで, k は外部配列の処理に必要な演算量を表します. k を調整することで, 特定のデータのソートに必要な演算量を大幅に削減可能性がある.

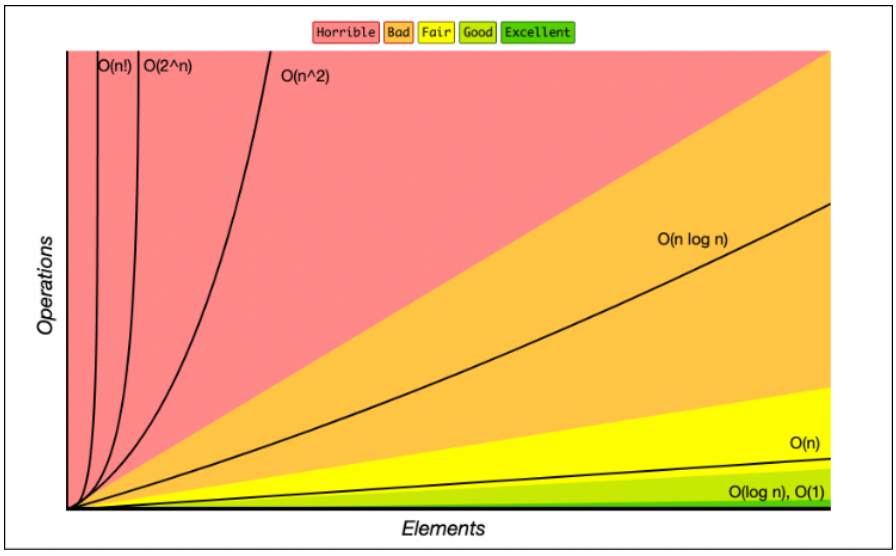

図 1: 様々な計算量を比較

表 1 に基づくと, 空間計算量とインプレースは, ソートが元のデータ配列自体の外で行われることを指します. バブルソート, シェイカーソート, 挿入ソートの空間計算量が $O(1)$ ので, 与えられたデータをソートするために必要な空間の量は最初から変わりません. つまり, これらのアルゴリズムは非常に空間効率が高く, 元のデータとは別に新しいメモリを割り当てる必要がありません. バケットソートと

カウントソートのアルゴリズムの空間計算量は$O(n+k)$と$O(k)$であり, この場合, kはデータをソートするために作成される新しい空間の量を指します. これらのアルゴリズムは新しい空間を割り当てる必要があり, アルゴリズムの設定方法によっては大量のメモリを消費する可能性があります. クイックソートは特に注目すべき点です. クイックソートの空間計算量は$O(\log n)$ですが, すべての操作はデータ配列内で行われ, 新しいメモリを割り当てる必要はありません. そのため, クイックソートはインプレースソートアルゴリズムと呼ばれます. クイックソートはソートが進むにつれて, 処理が必要なデータの長さが縮小し続けるためです.

3　実験セットアップ

3.1　ハードウェアと環境

3.2　実験に使用するデータセット

3.3　測定について

3.4　各アルゴリズムの説明

4　実験結果

5　分析と議論

6　まとめ

7　発表について