Vol.2022-CE-164 No.33
2022/3/13

アルゴリズム入門教育に適したソートアルゴリズムの検討

兼宗 進[1]　島袋 舞子[1]　岸本 浩輝[1]　漆原 宏丞[1]　本多 佑希[1]　岸本 有生[2]

概要：新課程の情報 I では、プログラミングと共にアルゴリズムが必修化された。そこで、わかりやすい教材や題材が求められている。現在、多くの教科書などで扱われているアルゴリズムとしては、探索や整列が挙げられる。整列については、バブルソート、挿入ソート、選択ソートなどが分かりやすいプログラムとして紹介されることが多い。これらは考え方としては理解がしやすい側面があるが、プログラムでコードを記述すると、二重ループが使われるため、実際にプログラムを生徒が書いて試すことは難しいという問題点があった。そこで本研究では、考え方の理解が容易であり、プログラムのコードとしても記述しやすいシンプルな入門用のアルゴリズムを検討したい。これらのアルゴリズムについて、考え方の理解とソースコードの理解しやすさを評価するとともに、大学の授業で行った理解度の調査について、他のアルゴリズムとの比較を報告する。

1. はじめに

高等学校の新課程「情報 I」では、プログラミングと合わせてアルゴリズムが必修化される。高校生に教えるにあたっては、初学者でも扱いやすいアルゴリズムを選ぶことは重要である。現在、多くの教科書などでは、探索や整列に関するアルゴリズムが多く扱われている。

その際には、整列についてはバブルソート、挿入ソート、選択ソートなどが分かりやすい例として紹介されることが多い。「隣の値と比較して大きい方が右側になるように入れ替える」であったり、「一番小さい値を探して一番左側に移動させる」であったりなど、基本となる考え方が単純であるため理解しやすい。しかし、プログラムでは二重ループが使われるため生徒が実際に書いて実行することは難しい側面もあった。

そこで、考え方の理解が容易であり、プログラムのコードとしても記述しやすいシンプルな入門用のアルゴリズムを検討することにした。本稿では、これらのアルゴリズムについて、考え方とソースコードの理解しやすさを評価するとともに、大学の授業で行った理解度の調査について、他のアルゴリズムとの比較を報告する。

2. 授業で扱われているアルゴリズム

現在、教科書などでは、表 1 で示すように、様々なアルゴリズムが扱われている。中でも探索や整列のアルゴリズムでは配列などのデータ構造と合わせて学習できることに加えて実用的なアルゴリズムであるため有用である。しかし、探索は多くの教科書が扱っているのに対し、整列は扱っていない教科書も見受けられる。

整列のアルゴリズムの例としては、バブルソート、選択ソートなどが扱われることが多い。これらのアルゴリズムは、バブルソートであれば「隣の値と比較して大きい方が右側になるように入れ替える」、選択ソートであれば「一番小さい値を探して一番左側に移動させる」など、基本的な考え方が単純であり、わかりやすい。こういった理由から生徒でも理解しやすいアルゴリズムではあるが、プログラムのコードとして記述する場合には、「繰り返すことを繰り返す」といった二重ループの考え方が必要になるため、生徒が自身で記述することは難しいという側面もある。そのため、これらのソートよりもわかりやすいアルゴリズムが求められている。

また、整列アルゴリズムには上記のソート以外にも多くのアルゴリズムが存在する。クイックソートは高速に動作するソートであるが、プログラムが難しいという問題もあるが、プログラムを用いず、机の上に並べたカードを昇順に並べ替えるようなコンピュータサイエンスアンプラグド [3, 4] を基にした授業方式を使った場合、高校生でもクイックソートを理解することができることを間辺らが明らかにしている [5]。これらの実践結果を受け、文部科学省が示した情報 I の教員研修用教材でも選択ソートに合わせてクイックソートが取り上げられている [6] また、マージソートなど一般的には授業で採用される機会が少ないもの

[1] 大阪電気通信大学
Osaka Electoro-Communication University
[2] 大阪電気通信大学高等学校
Osaka Electoro-Communication University High School

表 1 各教科書 [1,2] で取り上げられているアルゴリズム

出版社	教科書番号	教科書名	扱われていたアルゴリズム
日本文教出版	情科 310	新・情報の科学	線形探索、二分探索、選択ソート
第一学習社	情科 311	高等学校 情報の科学	バブルソート、コムソート、素数
実教出版	情科 308	情報の科学 新訂版	線形探索、二分探索、数独
数研出版	情科 309	改訂版 高等学校 情報の科学	線形探索、二分探索、バブルソート
東京書籍	情科 306	情報の科学	選択ソート、素数
第一学習社	情 I 713	高等学校 情報 I	素数
実教出版	情 I 703	高校情報 I Python	バブルソート、線形探索、二分探索
実教出版	情 I 706	図説 情報 I	選択ソート、バブルソート、線形探索、二分探索
数研出版	情 I 708	高等学校 情報 I	線形探索、二分探索、平方根、フィボナッチ数列
東京書籍	情 I 701	新編 情報 I	なし
東京書籍	情 I 702	情報 I Step Forward!	線形探索、二分探索、選択ソート
日本文教出版	情 I 710	情報 I	線形探索
日本文教出版	情 I 711	情報 I 図解と実習 図解編	素数

を授業で使用した例も見受けられる [7]。

3. 授業で使うことができるアルゴリズム

前章で述べたように、入門用のアルゴリズムは検討の余地があると思われるため、授業で使うことができるアルゴリズムを検討することにした。授業で使うためには、以下のようなことが重要だと考えられる。

- 考え方がわかりやすいこと
- プログラムのコードがわかりやすいこと
- データ構造が紙などに書き出しやすいこと

今回は、上記の条件を満たすアルゴリズムとして、ビンソート、ノームソート、スリープソートに着目した。これらはデータ構造として 1 次元配列を使う上、それぞれループが入れ子になっていないことから、プログラムも比較的簡潔である。また、これらに加えて、基数ソートにも着目した。他のソートと比べると少し複雑ではあるが、考え方に不思議さがあるため意欲的に学ぶことができるのではないかと考えた。

4. 整列アルゴリズムの比較

表 2 に、教育用に使われることが多い整列アルゴリズムと、今回提案した整列アルゴリズムとの比較を示す。バブルソート、選択ソートなどは、高等学校や大学の授業で利用されることが多く、一般的には難易度が低いアルゴリズムと考えられている。しかし、バブルソート、選択ソートはそれぞれ二重のループであるため、生徒が記述することは難しいのではないかと考えている。また、クイックソートは教え方を工夫することにより高校生でも学びやすいという報告があるが、プログラムが複雑になってしまう。一方で、我々が着目しているノームソート、ビンソート、バケットソート、スリープソートはループが入れ子になっていないため、プログラムの可読性が高く、高校生でも考え方を理解できることに加え、プログラムを生徒自身が記述することも可能になる可能性がある。

5. 実験授業 1

5.1 授業概要

広く授業で扱われているアルゴリズムと今回提案したアルゴリズムについて、「プログラムとしての記述のしやすさ」を評価するために実験授業を行った。対象は工学部の 3 年生 11 名、4 年生 2 名と工学研究科の大学院修士 2 年生 1 名であり、ゼミの一環として実施した。この実験では、対象として広く授業で扱われているアルゴリズムから選択ソート、バブルソート、今回提案したアルゴリズムからノームソート、ビンソートと合わせてバケットソートを扱った。なお、対象者は C 言語や Python の基礎を学んでいるが、アルゴリズムの授業は受けていない。

この授業では、学生はそれぞれのアルゴリズムを説明するプリントを見て考え方を理解した後、事後テストで理解度を確認した。

5.2 授業内容と事後テスト

学生に渡したプリントには、アルゴリズムの実行により配列の値がどう変化するかを時系列的に示す説明図と、手順を日本語で簡単に記述した手順書、アルゴリズムを C 言語で記述した際のプログラムを記載している。なお、4 つのアルゴリズムが対象ではあるが、時間の都合上、1 人あたり 3 つのアルゴリズムを学習してもらった。この際には、

表 2 整列アルゴリズムの比較

種類	一般的な難易度	本研究で予想した難易度	コードの行数	ループの深さ	備考
バブルソート	低	中〜高	6	2	
選択ソート	低〜中	中〜高	10	2	
クイックソート	中〜高	中	21	1	再帰呼び出し
基数ソート	中〜高	中〜高	13	2	
ノームソート	−	低〜中	11	1	
バケットソート	中〜高	低〜中	7	1	
ビンソート	−	低	7	1	
スリープソート	−	低	8	1	並列処理

選択者数には偏りが出ないように注意している。

今回は、プログラムとしての記述のしやすさを事後テストで測ることにした。このテストでは付録に載せる入れ替えが行われる流れを示した図のみを見せ、その図をもとに日本語で手順を記述させた。今回のテストはプログラムではなく日本語で手順を記述させたため、正確にはプログラムとしての記述のしやすさとは異なっている。しかし日本語として手順を記述する能力とプログラムを正しく記述する能力には相関があると考えられるため、そのアルゴリズムがプログラムとして記述しやすいかどうかを示す、1 つの目安になると考えている。また、日本語として記述させることによって、完全に理解しておらずともある程度は理解しているなど、ある程度段階的な評価ができる。なお、評価の際には教員が主観で「不正解を 0 点、正解を 100 点、部分点を 50 点ないしは 80 点」と段階的に点数をつけた。

5.3 結果

実験授業 1 の結果を表 3 に示す。表の中の B3 は大学 3 年生、B4 は大学 4 年生、M2 は修士課程の 2 年生を表す。この結果から、以下の 3 点がわかる。

- バケットソートは記述が難しい
- ノームソートは記述が容易である
- バブルソート、選択ソートはバケットソートより記述は容易であるが、ノームソートと比較すると難しい

仮説の通り、ノームソートは記述が容易であること、バブルソート、選択ソートは記述が難しいことがわかったが、バケットソートは、仮説と異なり記述が難しいことがわかった。

図 1 に、自由記述コメントの一部を示す。14 名中 3 名がノームソートが容易であったことを示すコメントをしていた。選択ソートやバブルソートは、難しいという記述も簡単だという記述もあった。

5.4 考察

今回の授業の結果から、ノームソートは記述が容易である可能性が示唆された。特に考え方が近いバブルソートとは大きく結果が異なった。また、学生からの感想を聞くアンケートでも、ノームソートの簡単さに触れる意見も一部見られた。

一方で、選択ソートやバブルソートが簡単だったという意見はノームソートに比べると低く、考え方は簡単である一方でプログラムは多少難しく感じているようである。このように差が生まれた原因としては、ノームソートはループが 1 重であるため、バブルソートよりも手順が考えやす

表 3 実験授業 1 の結果

被験者	ノーム	バケット	バブル	選択
(1)B3	80	50	80	-
(2)B3	100	50	-	100
(3)B3	80	-	80	100
(4)B3	-	0	0	0
(5)B3	100	100	0	-
(6)B3	100	0	-	0
(7)B3	80	-	80	80
(8)B3	-	50	80	80
(9)B3	50	0	80	-
(10)B3	100	-	0	0
(11)B3	50	0	-	0
(12)B4	0	50	-	80
(13)B4	80	-	0	0
(14)M2	100	50	80	-
平均	77	35	48	44

・ビンを理解するとバケットが理解しやすかった
・ノームはアルゴリズムの理解が難しかったが、
プログラムはわかりやすかった
・選択はアルゴリズムもプログラムもわかりやすい
・バブルは中間くらいのわかりやすさ
・左右の比較など言葉の説明が簡単なアルゴリズムほ
プログラムが難しかった
・バブルと選択は、アルゴリズムはわかりやすいが
プログラムは複雑
・ノームは、アルゴリズムは複雑だがプログラムはわかりやすい
・ノームのプログラムがいちばん読みやすかった

図 1 実験授業 1 の自由記述（一部抜粋）

いことが考えられる。

6. 実験授業 2

6.1 授業概要

広く授業で扱われているアルゴリズムと今回提案したアルゴリズムについて、「考え方のわかりやすさ」を評価するために実験授業を実施した。対象は文系の大学生と短大生 114 名である。共通教育科目の中で実施したため学生ごとの学部は統一されていないが、理系分野を専攻している学生もおらず、他の授業の中でプログラムやアルゴリズムを扱う科目もない。同授業のうち、入門用プログラミング言語「ドリトル」や Python などを合計 7 回扱っているためプログラムに関してはある程度見慣れている。なお、今回実施した授業はオンデマンド動画を配信する形式の授業である。

学生には動画視聴後に授業課題に取り組んでもらうとともに、授業の感想を聞くアンケートに答えてもらった。この授業課題の正答率と、感想を基に考察を行うことにした。

6.2 授業内容

動画の中では、コンピュータサイエンスアンプラグド [3] や間辺ら [5] の実践に倣って、数字が書かれたカードを並べ替えるといった説明を行うことにした。図 2 に動画の一部を示す。この場面では、基数ソートを説明している。また、学生には実際に自身でカードを作って動画内での説明を手元で再現しながら理解するように指示した。

扱ったアルゴリズムは以下の通りである。

- 選択ソート
- バブルソート
- クイックソート
- ビンソート
- 基数ソート
- スリープソート

動画内ではアルゴリズムを学習する動機づけを話したのち、上記の各アルゴリズムについて「戦略を簡単に説明し、実際に紙のカードを並べ替えを行う様子を見せる」をそれぞれ行った。なお、スリープソートについてはその性質上 1 人でカードを使って並べ替えることは難しいため、プログラムとその実行結果を見せて紹介する程度に留めた。

また、今回は「考え方のわかりやすさ」を評価するという目的上、各アルゴリズムのプログラムを示すと結果に影響すると考えたため、スリープソート意外のプログラムは提示していない。

6.3 授業課題とアンケート

学生の各ソートの理解度を測るため、動画視聴後に授業課題を課した。課題で提示した画像を図 3 から図 7 に示す。これらの図中の空欄に当てはまる数値を答えさせた。

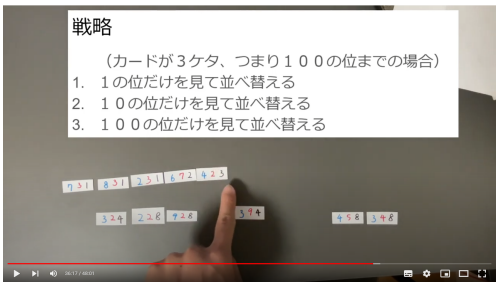


図 2 オンデマンド動画の一場面

(1)	8	2	9	4	3	7	6
(2)	2	8	9	4	3	7	6
(3)							
(4)							
(5)	2	3	4	6	8	7	9
			（以下略）				

図 3 選択ソートの課題で示した画像

(1)	8	2	9	4	3	7	6
(2)	2	8	9	4	3	7	6
(3)							
(4)							
(5)	2	8	4	3	9	7	6
(6)	2	8	4	3	7	9	6
			（以下略）				

図 4 バブルソートの課題で示した画像

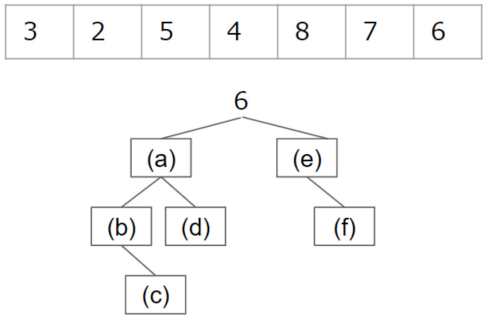


図 5 クイックソートの課題で示した画像

また、各アルゴリズムに対する感想を聞くアンケートを実施した。設問は「好きだと感じたアルゴリズム」「簡単だと思ったアルゴリズム」「難しいと思ったアルゴリズム」「中でも理解できなかったアルゴリズム」の 4 つで、それぞれ複数回答可能である。

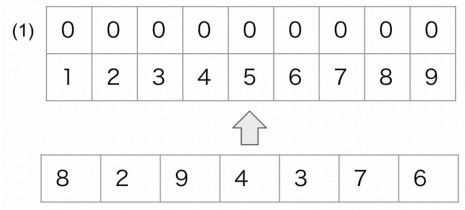


図 6 ビンソートの課題で示した画像

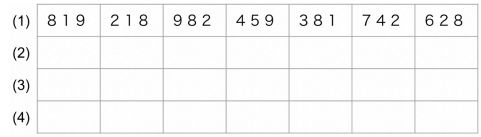


図 7 基数ソートの課題で示した画像

表 4 各アルゴリズムに対する正答率

アルゴリズムの種類	正答率 [%]
選択ソート	48.24
バブルソート	38.60
クイックソート	61.40
ビンソート	64.91
基数ソート	69.30

表 5 「好きだと感じたアルゴリズム」に対する回答率 (降順)

アルゴリズムの種類	回答率 [%]
基数ソート	41.12
クイックソート	28.03
選択ソート	12.14
ビンソート	9.34
バブルソート	3.73
スリープソート	1.86

表 6 「簡単だと感じたアルゴリズム」に対する回答率 (降順)

アルゴリズムの種類	回答率 [%]
選択ソート	30.63
ビンソート	27.02
基数ソート	26.12
クイックソート	25.22
バブルソート	14.41
スリープソート	0.90
全部	0.90

表 7 「難しいと感じたアルゴリズム」に対する回答率 (降順)

アルゴリズムの種類	回答率 [%]
バブルソート	29.35
クイックソート	27.52
基数ソート	14.67
ビンソート	12.84
なし	11.00
選択ソート	10.09
スリープソート	5.50
全部	2.75

表 8 「理解できなかったアルゴリズム」に対する回答率 (降順)

アルゴリズムの種類	回答率 [%]
なし	73.63
バブルソート	9.09
ビンソート	9.09
選択ソート	3.63
全部	3.63
クイックソート	2.72
基数ソート	2.72
スリープソート	0.90

6.4 結果

授業課題の結果を表 4 に示す。基数ソートは一番正答率が高く、次にビンソート、クイックソート、後に選択ソート、一番正答率が低いのはバブルソートだった。

また、授業課題とともに実施したアンケートの回答を表 5 から表 8 に示す。これらの表にスリープソートは登場しているが、実際にカードを並び替える作業をしていないため他のソートとは同列に扱うことができない。そのため、これらの表の中ではスリープソートの順位などは大きな意味を持たない。

学習してみて好きだと感じたアルゴリズムを答える設問（表 5）を見ると、教育の場面でよく選択ソート、バブルソートはあまり好きだと感じた学生は少ないことがわかった。また、一見複雑で難しく思える基数ソートやクイックソートが好きだと感じた学生が多いことがわかる。

表 6 を見ると、基数ソートを好きだと感じた学生は、必ずしも簡単だと思ったわけではないこともわかる。加えて、選択ソートは簡単だと感じながら好きではないと感じる学生が多いようだ。これらの表を見ると、学生は簡単なアルゴリズムを好きと感じているわけではないことがわかる。

表 7 を見ると、教育の場で採用されることが多いバブルソートは、難しいと感じる学生が多かったようだ。この原因としてはいくつかの理由が考えられるが、根幹がシンプルである故に、実際にカードを並び替える場合には手数が多くなってしまうことなどが関係していると考えられる。

表 8 を見ると、理解できないほどの難易度のソートは殆どなかったことがわかる。ただ、今回はプログラムを書かせるような内容ではないため、「理解できたような気がする」という程度の学生もいたと思われる。

6.5 考察

バブルソートや選択ソートと合わせて様々なアルゴリズムを授業で扱い、事後テストやアンケートを実施してみた結果、これまで多くの場面で使用してきたバブルや選択などのアルゴリズムは必ずしもわかりやすく適しているわけではないのではないのではないかと感じた。

バブルソートや選択ソートは正答率がかなり低く、ビンソートやクイックソートの方が正答率が高かった。また、基数ソートが一番正答率が大きいという結果になった。バ

ブルや選択が低い原因は、これらのソートが難しいだけではなく、カードを並べ替える際に手順が多く、そのためどこかで 1 つミスがあればすべての結果が誤りとなってしまうなど様々な要因が関係していると考えられる。

興味関心を聞くアンケートではバブルや選択などのソートは学生の興味をあまり引かず、特に基数ソートは多くの学生が興味を引く結果となった。ビンソートは興味はあまり引かなかったようだが、簡単でわかりやすかったという意見は多かった。スリープソートは今回の授業ではカードの並べ替えなどを体験させられなかったため有効であるかは確認できなかったが、わかりやすく興味を引くものだと感じている。

バブルや選択などのアルゴリズムが教育で用いられることが多い原因として、根幹となる基本の考え方が単純であることや、多重ループ、比較や入れ替えなど同時に様々な技能を教えられることなどがあると考えている。しかし、順不同なカードを並べ替えたいという問題を解決する手段としてはこれらの技能は必須ではないことに加えて、比較や入れ替えなどを一切用いないソートも扱うことで、様々な方面からのアプローチで問題を解決する姿勢を身に着けさせることもできる可能性もあると考えている。

7. まとめ

アルゴリズム教育の多くの場面で扱われるバブル・選択などのアルゴリズム以外にも、高校生でも理解できてわかりやすいアルゴリズムがあるのではないかと着目し、ノームソート、ビンソート、バケットソート、基数ソート、スリープソートなど授業での実践例が少ないアルゴリズムを用いて授業実践を行った。実験授業 1 ではプログラムとしての記述のしやすさを、実験授業 2 では考え方のわかりやすさを測った。その結果、必ずしもバブルや選択などのアルゴリズムがわかりやすく適したものではない可能性が見えた。

選択やバブルは根幹の考え方が単純ではあるが、手順が多く、実際に手を動かして並び替えてみることは難しい。実験授業 2 ではカードの並び替えを課題として扱った結果、生徒の理解度には大きく差が生まれた。この差の原因はこういった、手順の多さも原因だと考えられる。

今後は、アルゴリズム間での理解度の差は何が原因なのかをより明らかにしていくとともに、より適したアルゴリズムがないかを模索するなど、進めていきたい。

参考文献

[1] 文部科学省: 高等学校用教科書目録 (令和 3 年度使用), 入手先〈https://www.mext.go.jp/content/20200430_mxt_kouhou02_.mext_00001_03.pdf〉 (参照 2022-02-16).
[2] 文部科学省: 高等学校用教科書目録 (令和 4 年度使用), 入手先〈https://www.mext.go.jp/content/20210604-mxt_kyokasyo02-000014470_4.pdf〉 (参照 2022-02-16).
[3] Bell, T., Witten, I.H. and Fellows, M.: *Computer Science Unplugged - An enrichment and extension programme for primary-aged children* (2005).
[4] 兼宗進 (監訳): コンピュータを使わない情報教育アンプラグドコンピュータサイエンス, イーテキスト研究所 (2007).
[5] 間辺広樹, 兼宗進, 並木美太郎: CS アンプラグドのアルゴリズム学習における教具による理解度の影響, 情報処理学会論文誌, Vol.54, No.1, pp.14-23 (2013).
[6] 文部科学省: 高等学校情報科「情報 I」教員研修用教材, 入手先〈https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1416756.htm〉(参照 2022-02-16).
[7] 間辺広樹, 神道健朗, 並木美太郎, 兼宗進: コンピュータ・アルゴリズムの「発見・記述・伝達」を導く授業の実践と評価, 情報処理学会論文誌「教育とコンピュータ」, Vol.2 No.1 pp.10-24 (2016).

Vol.2022-CE-164 No.33
2022/3/13

ノームソート

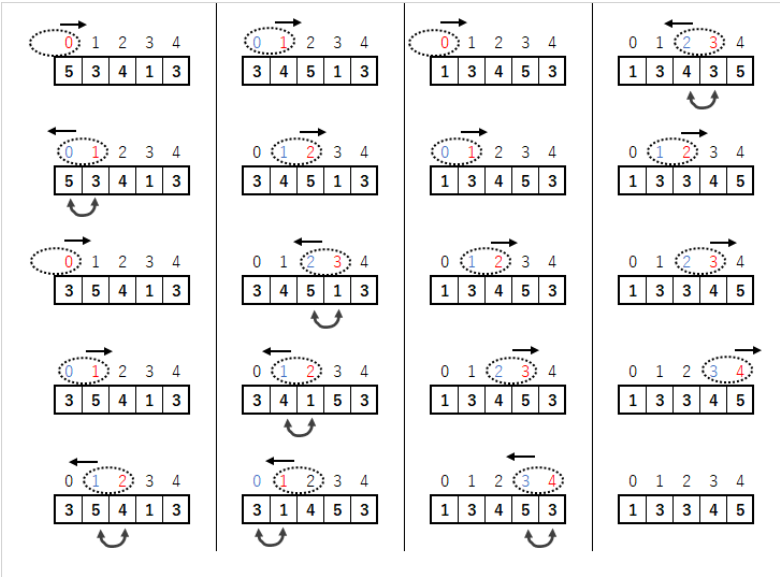
基本的な考え方

ノームソートは、先頭から順に隣り合う要素（ペア）を比較し、交換していくことで整列を行う。交換を行わなかった場合は次のペアに移り、交換を行った場合は1つ前のペアに戻る。

例えば、[5,3,4,1,3] を先頭から昇順に並び替える場合は、次の図のように並び替えが行われる。比較を行うペアは丸で囲ってある。

手順１：左端にいる場合は、ペアが存在しないのでひとつ右に移る
手順２：ペアの値を比較し、左の値のほうが大きい場合は、ペアの値を交換し、左に戻る
手順３：ペアの値を比較し、左の値のほうが大きくない場合は、右に移る
手順４：右端まで比較を終えるまで、手順1～３を繰り返す

<[5, 3, 4, 1, 3] を先頭から昇順に並び替える例>



プログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを入力、実行してデータの動きを確認しましょう。

```
data = [5, 3, 4, 1, 3]
pos = 0

while pos != len(a):
    if pos == 0:
        pos += 1

    if data[pos - 1] > data[pos]:
        tmp = data[pos - 1]
        data[pos - 1] = data[pos]
        data[pos] = tmp
        pos -= 1
    else:
        pos += 1

print(data)
```

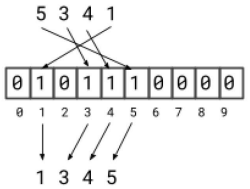
ビンソートとバケットソート

ビンソートの基本的な考え方

ビンソートは、あらかじめ配列を用意しておき、並び替えを行う要素の値（データ）と同じ添字の要素に1を代入する。その後、1が代入された要素の添字を順に取り出すことで整列を行う。ビンソートでは、複数の同じ値を扱うことはできない。

手順１：配列を用意し、すべての要素に0を入れておく
手順２：データを見て、データと同じ添字の要素に1を代入する
手順３：手順2をデータの数だけ繰り返す
手順４：配列の要素を順に調べ、1が代入されていれば添字の値を書き出す

＜[5, 3, 4, 1] を先頭から昇順に並び替える例＞



ビンソートのプログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

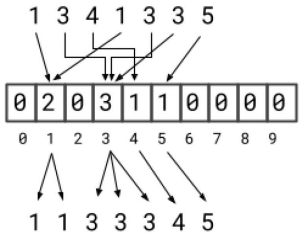
```
data = [5, 3, 4, 1]
bucket = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for i in data:
    bucket[i] = 1

for i in range(len(bucket)):
    if bucket[i] == 1:
        print(i, end=" ")
```

バケットソートの基本的な考え方

バケットソートは、あらかじめ配列を用意しておき、並び替えを行う要素（データ）の値と同じ添字の要素に1を加算することをくり返す。その後、加算された数だけ添字の要素を順に取り出すことで整列を行う。バケットソートでは複数の同じ値を扱うことができる。

手順１：配列を用意し、すべての要素に0を入れておく
手順２：データを見て、データと同じ添字の要素に1を加算する
手順３：手順2をデータの数だけ繰り返す
手順４：配列の要素を順に調べ、値の数だけが添字の値を書き出す

＜[1, 3, 4, 1, 3, 3, 5] を先頭から昇順に並び替える例＞



バケットソートのプログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
data = [1, 3, 4, 1, 3, 3, 5]
bucket = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for i in data:
    bucket[i] += 1

for i in range(len(bucket)):
    for j in range(bucket[i]):
        print(i, end=" ")
```

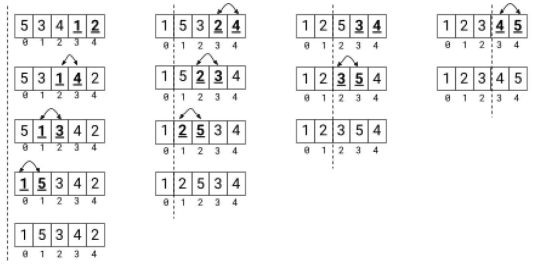
バブルソート

基本的な考え方

バブルソートでは、隣り合う要素（データ）の大小を比較し、前の要素よりも後の要素が小さい（または大きい）ときに交換することをくり返すことで整列を行う。

手順１：右端の2個の値を比較する
手順２：左側の値が大きい場合は、2個の値を交換する
手順３：それぞれ左に1つ分、移動する
手順４：手順1～3を左端（点線）に行くまで繰り返す
手順５：左端（点線）まで来たら、最も小さい値が左端に来たので、右端に戻り、手順1～５を繰り返す

＜[5, 3, 4, 1, 2] を先頭から昇順に並び替える例＞



プログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
data = [5, 3, 4, 1, 2]

for i in range(len(data) - 1):
    for j in range(len(data) - 1, i, -1):
        if data[j - 1] > data[j]:
            tmp = data[j - 1]
            data[j - 1] = data[j]
            data[j] = tmp

print(data)
```

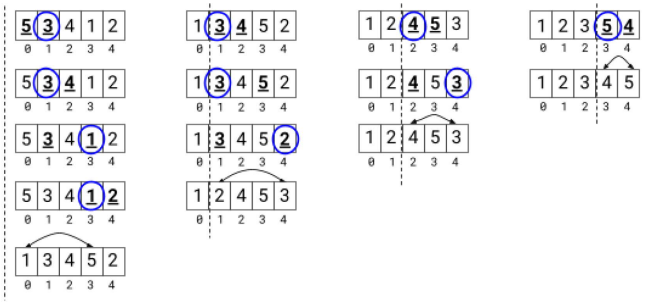
選択ソート

基本的な考え方

選択ソートは、並び替えを行う要素（データ）の中から最も小さい値を探し、配列の先頭の要素と入れ替えることをくり返すことで整列を行う。

手順１：点線の右側にある2個の値を比較して、小さい方に○を付けて覚えておく
手順２：小さかった値と、右の新しい値を比較して、小さい方に○を付けて覚えておく
手順３：手順２を右端に行くまで繰り返す
手順４：最も小さい値○を、点線の右の値と交換する
手順５：最も小さい値が左端にきたので、その右隣（点線より右）から手順1～5を繰り返す

＜[5, 3, 4, 1, 2] を先頭から昇順に並び替える例＞



プログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
data = [5, 3, 4, 1, 2]

for i in range(len(data)):
    pos = i
    maxi = data[i]
    for j in range(i, len(data)):
        if maxi > data[j]:
            maxi = data[j]
            pos = j
    tmp = data[i]
    data[i] = data[pos]
    data[pos] = tmp

print(data)
```

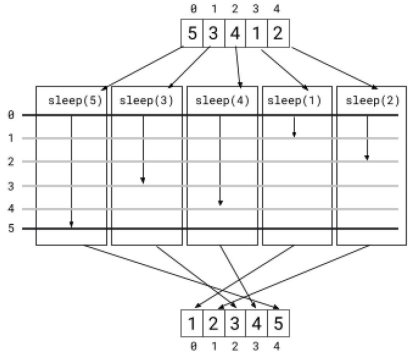
スリープソート

基本的な考え方

スリープソートは、並び替えを行う要素（データ）の時間だけ動作するスレッド（タイマー）を用意し、それらを一斉に開始してスレッドが終了した順に値を出力することで整列を行う。

手順１：左側の値を読み、その値の時間だけ動作するスレッド（タイマー）を作成し、右に移る。
手順２：手順1を並び替えを行う要素の数だけ繰り返す。
手順３：作成したスレッド（タイマー）を一斉に開始し、終了した順に値を出力する。

＜[5, 3, 4, 1, 2] を先頭から昇順に並び替える例＞



プログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
import threading
import time

def sleep(t):
    time.sleep(t)
    print(t)

data = [5, 3, 4, 1, 2]

for x in data:
    threading.Thread(target=sleep, args=(x,)).start()
```

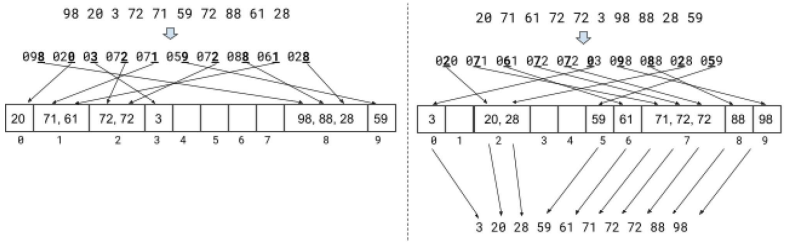
基数ソート

基数ソートの基本的な考え方

基数ソートは、あらかじめ配列を用意しておき、並び替えを行う要素の値（データ）の位を取り出す。取り出した値と同じ添字の要素に値を代入することをくり返すことで整列を行う。

手順１：配列を用意する。
手順２：並び替えを行う要素の値の先頭に「0」をつける。
手順３：データのn桁目を見て、それと同じ添字の要素にデータを代入する（nの初期値は1）。
手順４：手順3をデータの数だけくり返す。次の桁に移動する（n=n+1）。
手順５：手順2から手順4をくり返す。
手順６：配列の要素を順に調べ、値を書き出す

＜[98, 20, 3, 72, 71, 59, 72, 88, 61, 28] を先頭から昇順に並び替える例＞



基数ソートのプログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
data = [98, 20, 3, 72, 71, 59, 72, 88, 61, 28]
bucket = [[], [], [], [], [], [], [], [], [], []]

for t in range(2):
    for i in range(len(data)):
        digits = "0" + str(data[i])
        digit = int(digits[len(digits) - (t + 1)])
        bucket[digit].append(data[i])

    data = []
    for x in bucket:
        for y in x:
            data.append(y)
    bucket = [[], [], [], [], [], [], [], [], [],
[]]
print(data)
```

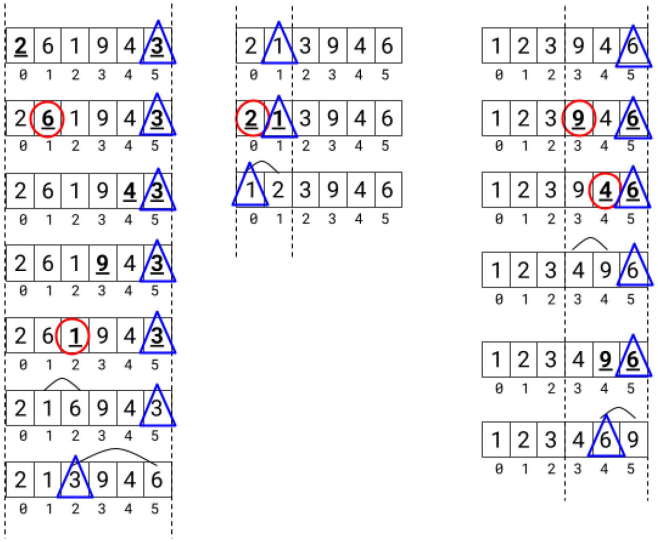
クイックソート

基本的な考え方

クイックソートは、並び替えを行う要素（データ）を基準値と比較して、大きい値のまとまりと小さい値のまとまりに分けることをくり返すことで整列を行う。

手順１：右端の値を基準値（ピボット）として△を付けて覚えておく。
手順２：基準値よりも大きい値が現れるまで、左端から順に基準値と比較する。基準値よりも大きい値には○を付けて覚えておく。
手順３：基準値よりも小さい値が現れるまで、基準値の左にある値から順に左に1つずつ基準値と比較する。基準値よりも小さい値には○を付けて覚えておく。
手順４：手順2と手順3で○をつけた値を交換する。
手順５：○をつけた大きい値と基準値を交換する。
手順６：基準値の左側にある値と右側にある値に分割し、整列を終えるまで手順1～5をくり返す。

＜[2, 6, 1, 9, 4, 3] を先頭から昇順に並び替える例＞



プログラム

上記の考え方をもとにプログラムにすると、次のようになる。Bit Arrow上でプログラムを実行してデータの動きを確認しましょう。

```
import random


def qsort(data):
    if len(data) <= 1:
        return data

    piv = data[-1]

    small = []
    big = []

    for i in range(len(data)):
        if data[i] <= data[piv]:
            small.append(data[i])
        else:
            big.append(data[i])

    small = qsort(small)
    big = qsort(big)

    result = []
    for s in small:
        result.append(s)
    for b in big:
        result.append(b)

    return result


data = [2, 6, 1, 9, 4, 3]
print(qsort(data))
```