第６章　アルゴリズムとプログラミング
２節　プログラミングの実践
その５

学習項目

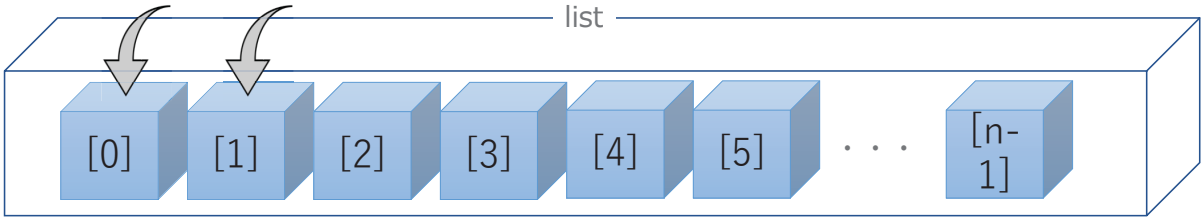
- 配列を利用した簡単なコードを書くことができる。
- 線形探索のアルゴリズムを理解している。

配列とは

配列とは，複数のデータをまとめて管理するために用いる，番号を付けた変数のこと。
Pythonには**リスト**というデータ構造が用意されている。

下の図では「list」（←自分でつけた名前）という名前でまとめられた箱が n 個ある。
箱の１つ１つを要素といい，箱の数を要素数という。

それぞれの「箱」に数値や文字列を出し入れし，
複数のデータをまとめて管理する。



配列とは

要素は list[0] から list[n-1] というように [] を用いて記述する。
list[]という配列を用いて，データをまとめて管理できる。

入力時　自分でつける

配列名=[データ０, データ１, データ２, …]

複数のデータをコンマで区切って記述

出力時　**配列名[番号]**

例えばデータ０を返したいときは　配列名[0]

※　中のデータの番号は０からスタートすることに注意

```
subject=["国語","数学","英語","理科","社会","情報"]
score=[80,100,92,96,74,99]
total=0
for i in range(6):
    print(subject[i],"の点数は",score[i])
    total=total+score[i]
print("合計点は",total)
```

```
国語 の点数は 80
数学 の点数は 100
英語 の点数は 92
理科 の点数は 96
社会 の点数は 74
情報 の点数は 99
合計点は 541
```



```
subject=["国語","数学","英語","理科","社会","情報"]
score=[80,100,92,96,74,99]
total=0
for i in range(6):
    print(subject[i],"の点数は",score[i])
    total=total+score[i]
print("合計点は",total)
```

ここはsubjectのリストの個数を入れるが,
自動でリストの個数が計算された方が楽な場合もある

len(配列名) … 配列名の要素数を返す命令

例えば上のコードに対して

```
n=len(subject)
```

とすると，変数nに配列subjectの要素数6が入る

配列練習問題

次の配列に記された値 n を全て調べ，
偶数であれば「n は偶数です」，奇数であれば「n は奇数です」
と出力するプログラムを作成しなさい。

```
a=[10,-5,0,29,6,2,77,35,7,-365]
```

フローチャートを作ってみよう

サーチ（線形探索）

サーチとは探索とも呼ばれる。

線形探索は，データの中に目的の値があるかを１つずつ調べて判断する方法。

例えば…

あなたは１パック10個入りの卵を買いました。

卵の中からちょうど60gのものを１個探したいです。

このとき，みなさんは卵を１つずつ秤に載せて60gのものを探すでしょう。
もちろん，全部調べても目的の重さの物はないかもしれません。

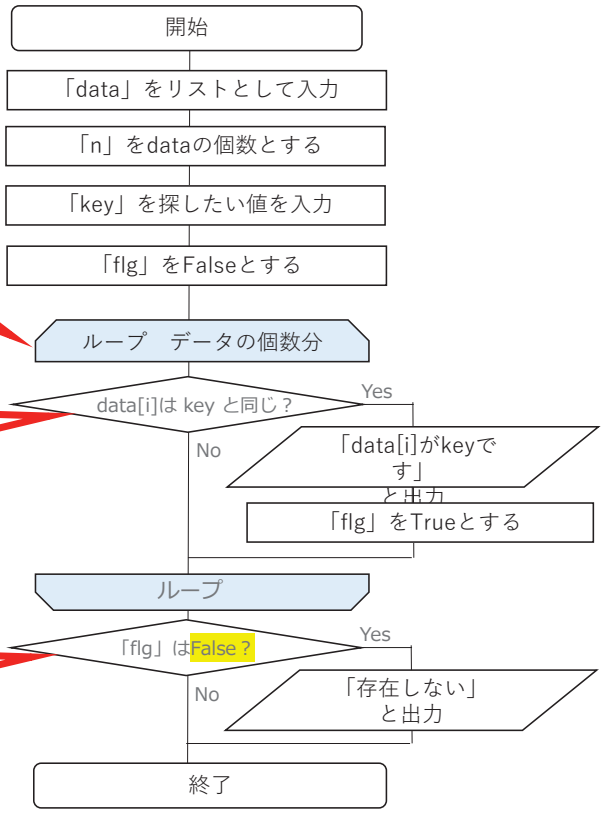
これが線形探索です。

サーチ（線形探索）

線形探索では反復構造を使ってdataの中身を１つずつチェックしていく

反復構造の中で，分岐構造を作り，keyと一致したもののみ出力するようにする

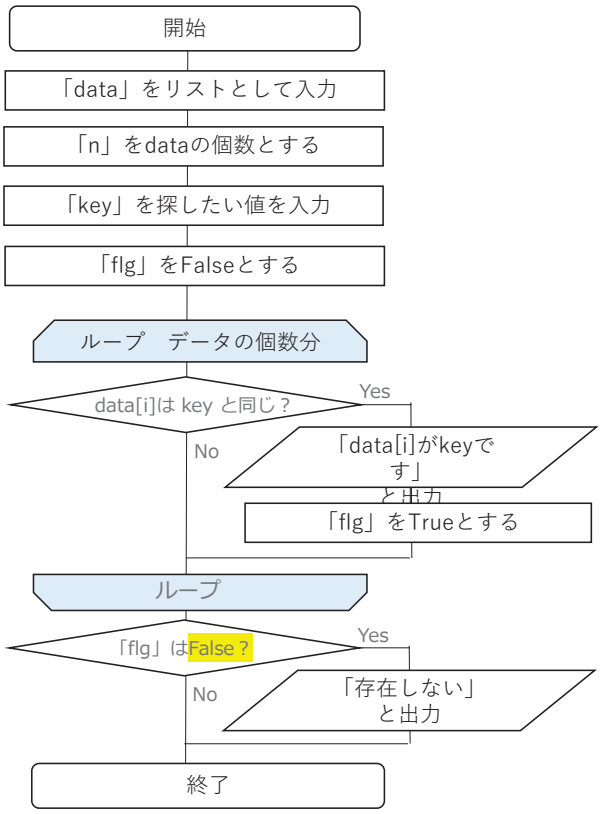
「flg」を目印として，keyと一致するものがなかったときは「存在しない」と出力するようにする



サーチ（線形探索）

```
data=[57,48,49,48,61,59,56,64,60,60]
key=60
```

として右のフローチャートを参考に線形探索のプログラムを作成しなさい。



第６章　アルゴリズムとプログラミング
２節　プログラミングの実践
その６・７

学習項目

- 選択ソート，バブルソートのアルゴリズムを理解する

ソートとは

ソートとは，データを一定の規則に従って並び替えること。

データを小さいものから大きいものへと順に並び替えることを昇順にソートするといい，

大きいものから小さいものへと順に並び替えることを降順にソートするという。

ソートとは

～ソートを学ぶ意味～

ソートはデータの前処理として行われ，例えば目的の値を探しやすくするために重要な作業です。

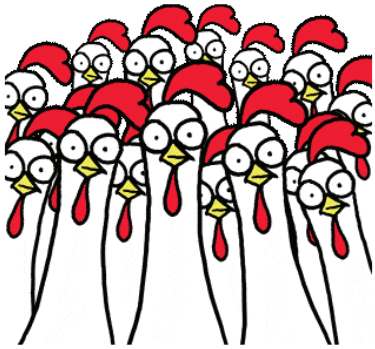
国語辞典や英語辞典は五十音順やアルファベット順にかかれています。もしこれが不規則に単語が載っていたとしたら，目的の語を探すのにどれだけ無駄な時間を費やすか想像にたやすいでしょう。

コンピュータも人間も，整然と並ぶデータであれば目的のものを見つけやすいのです。とくにデータ数が多くなればなるほど，ソートなしでは時間も負荷もかかるようになってしまいます。

したがってソートは私たちの生活の中でも使用頻度の高いアルゴリズムとなっているのです。

ソートの種類

ソートのアルゴリズムは古くから考えられており，何種類もある。
ここでは代表的なソートのアルゴリズムについて学習する。



① 選択ソート
② バブルソート
③ 挿入ソート
④ クイックソート

以降ばらばらの数値を昇順ソートするアルゴリズムを考えてみましょう。

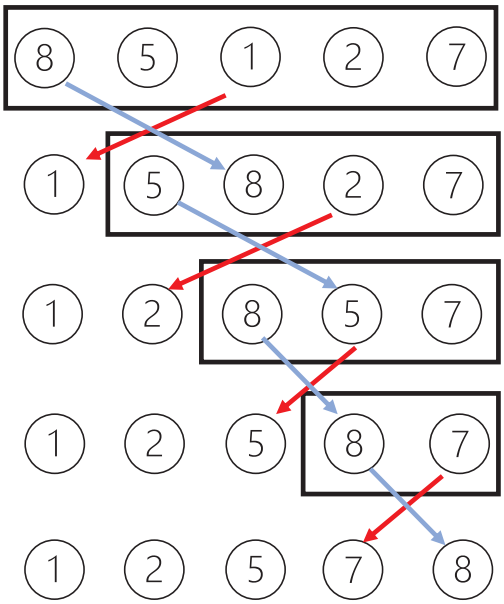
① 選択ソート とは

選択ソートとは，データの並びから最も小さな値を探し，
それを先頭のデータと入れ替える。

次に入れ替えた先頭のデータを除いたデータの並びから最
も小さな値を探し，その並びの先頭と入れ替える。

これを繰り返す手法。

選択ソートのアルゴリズム



1．データの中から最も小さな値を探し（ここでは①），先頭の⑧と入れ替える。

2．先頭の①を除いたデータの中から最も小さな値を探し（ここでは②），先頭の⑤と入れ替える。

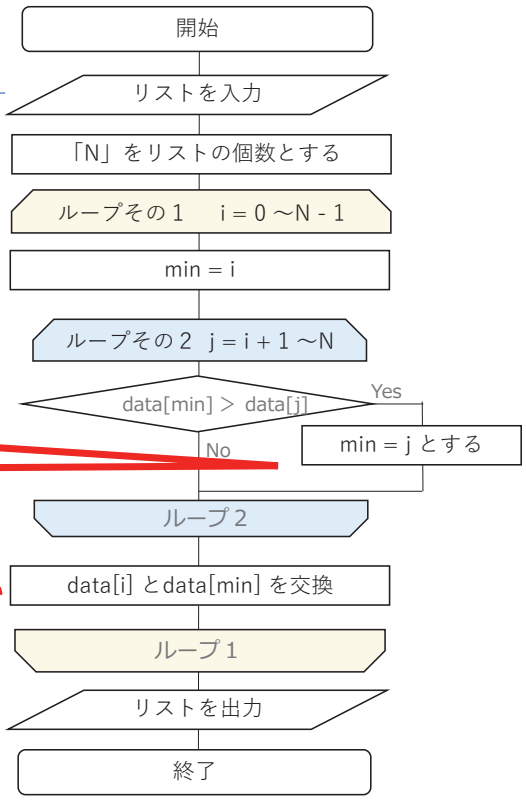
3．先頭の①と②を除いたデータの中から最も小さな値を探し（ここでは⑤），先頭の⑧と入れ替える。

4．先頭の①②⑤を除いたデータの中から最も小さな値を探し（ここでは⑦），先頭の⑧と入れ替える。

5．ソート完了

選択ソートのアルゴリズム

選択ソートのアルゴリズムをフローチャートに表すと右のようになる。



「min」は最小値の位置を保持する変数

ここでリストの i 番目より j 番目の方が小さければ，最小値の位置である「min」が j に変わる

これでデータの左端に小さい数が来る

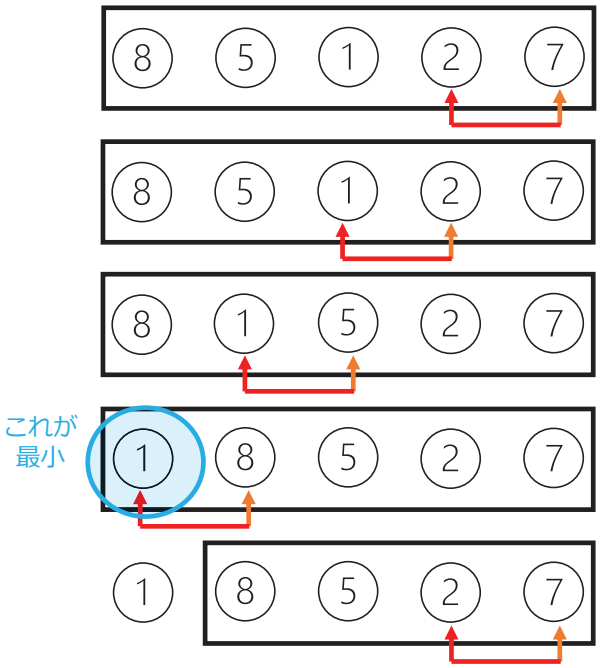
②　バブルソート　とは

バブルソートとは，データの左右を比較し，小さい値を左に，
大きい値を右に移すことで，データを昇順に並べる方法。

１番右の２ペアから比較し，順に比較ペアを左に移していく
ことで，左端が１番小さな値になる。

これを繰り返す手法。

バブルソートのアルゴリズム



１．データの右端の左右に並んだ値を比較し，左＞右
　　なら入れ替える。そうでないならそのまま。

２．１つ左のペアの左右を同じように比較する。

３．さらに１つ左のペアを比較する。
　　（今回は①と⑤を入れ替える。）

４．左端までこれを行うと，最も小さな値が先頭に来る。

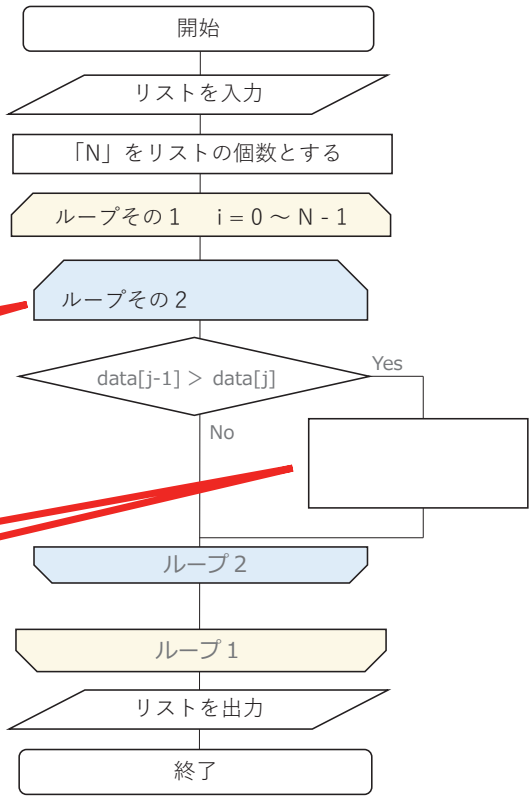
５．左端を除いたデータで同じことを繰り返す

バブルソートのアルゴリズム

バブルソートのアルゴリズムをフローチャートに表した。
以下の問いに答えなさい。

右端から見ていきたい。ループ２のrange
はどう設定すればよいでしょう？

ペアの左側の方が大きいときはどういう処
理を行えばよいのでしょう？



コードを書いてみよう

選択ソートとバブルソートのコードを書いてみましょう。

ここで，どちらにも使っている“入れ替え”の命令は以下の通りです。

変数a,bに対してaとbの箱の中身を入れ替えたいときは

a , b = b , a

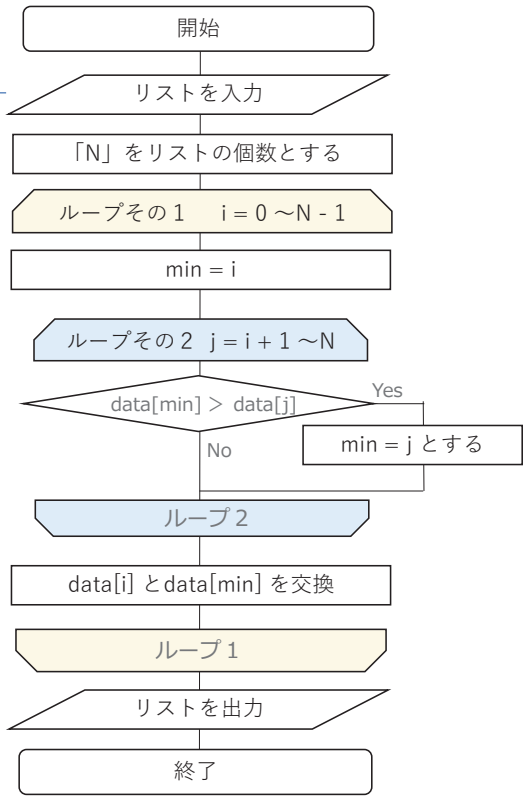
リストのm番目とn番目の要素を入れ替えたいときは

data[m] , data[n] = data[n] , data[m]

選択ソートのコード

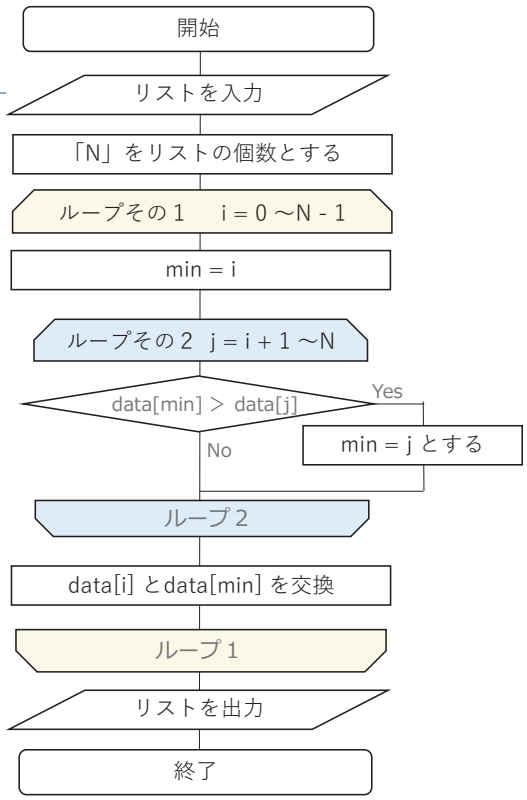
選択ソートのコードを書いてみましょう。
ただし，次の続きから作成しなさい。

```
data=[9,4,7,2,3,8,6,1,5,0]
print(data,"←元のデータ")
N=len(data)
```



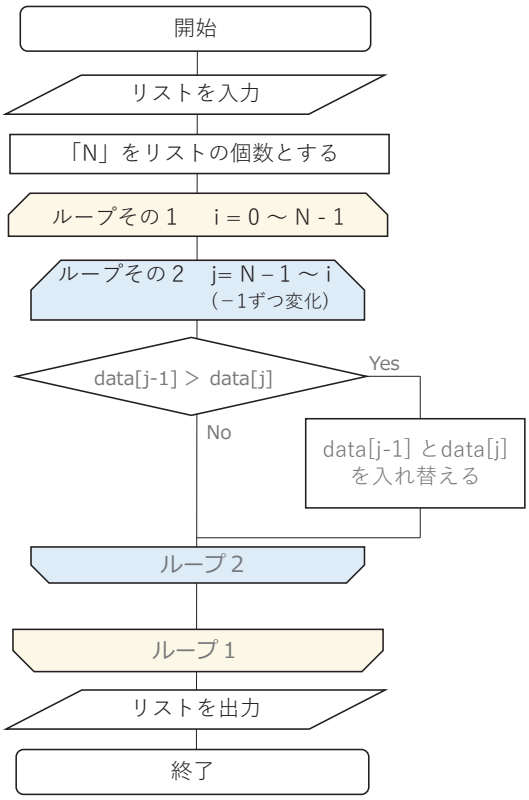
選択ソートのコード

メモ



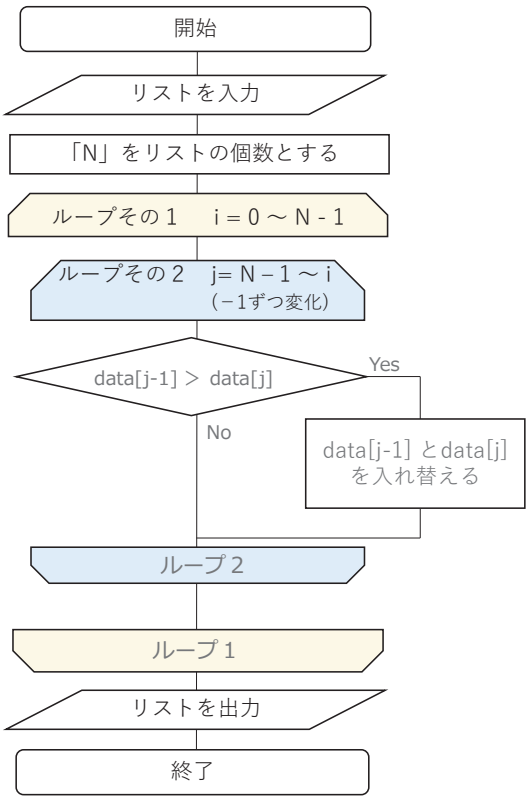
バブルソートのコード

バブルソートのコードを書いてみましょう。



バブルソートのコード

メモ



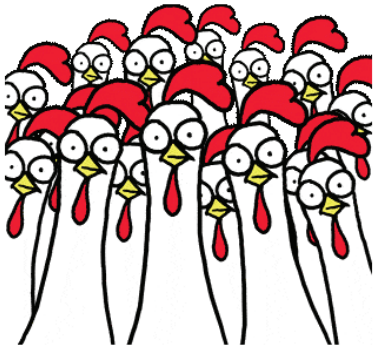
第６章　アルゴリズムとプログラミング
２節　プログラミングの実践
その８・９

学習項目

- 挿入ソートのアルゴリズムを理解する
- 挿入ソートのアルゴリズムからフローチャートを作成することができる

ソートの種類

ソートのアルゴリズムは古くから考えられており，何種類もある。
ここでは代表的なソートのアルゴリズムについて学習する。



① 選択ソート
② バブルソート
③ 挿入ソート
④ クイックソート

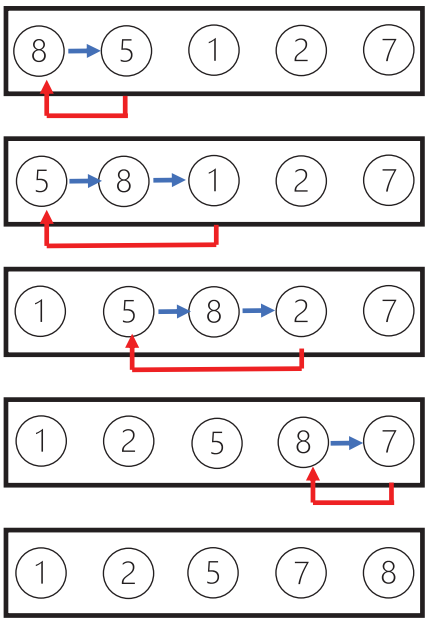
以降ばらばらの数値を昇順ソートするアルゴリズムを考えてみましょう。

③ 挿入ソート とは

挿入ソートとは“基準の値”を定め，それより左のすべての
値を順に調べ，“基準の値”をあるべき位置に挿入する。

“基準の値”を順に右にずらしていくことでソートする手法。

挿入ソートのアルゴリズム



1．左端と１つ右の値を比較し，右の方が小さければ，左の値を右にずらして，その前に挿入する。

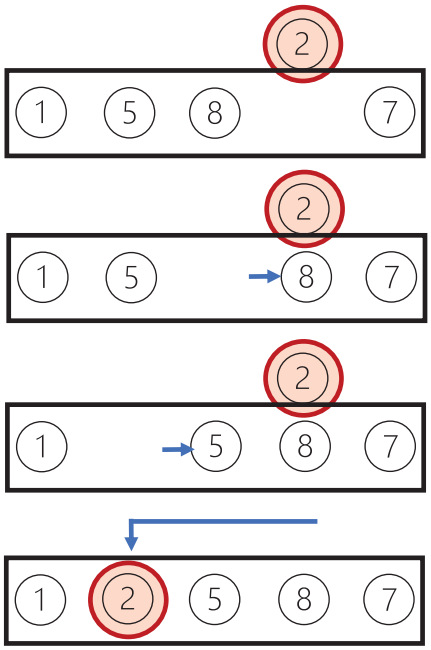
2．左から２つ目の値（ここでは①）をそれより左の値と比べ，挿入する（ここでは⑤⑧を右にずらしてその前に①を入れる）

3．左から３つ目の値（ここでは②）をそれより左の値と比べ，挿入する（ここでは⑤⑧を右にずらしてその前に②を入れる）

4．左から４つ目の値（ここでは⑦）をそれより左の値と比べ，挿入する（ここでは⑧を右にずらしてその前に⑦を入れる）

5．ソート完了

「比べ，挿入する」の詳細



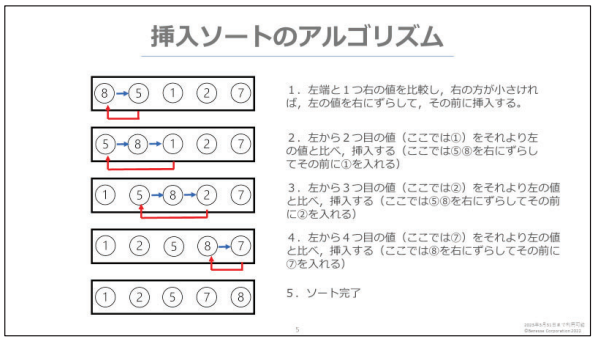
1．②を一旦基準として保持する

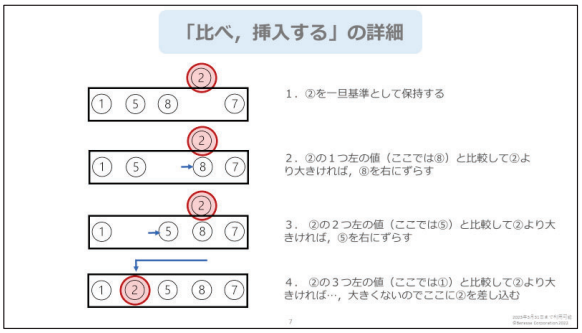
2．②の１つ左の値（ここでは⑧）と比較して②より大きければ，⑧を右にずらす

3．②の２つ左の値（ここでは⑤）と比較して②より大きければ，⑤を右にずらす

「基準」よりも左側は
すでに小さい順に並んでいるはず

4．②の３つ左の値（ここでは①）と比較して②より大きければ…，大きくないのでここに②を差し込む

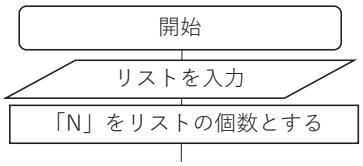




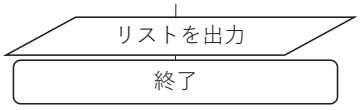
ここからわかるように，挿入ソートは分岐構造と反復構造が入り組んでいる。

今回のソートは自分でアルゴリズムを考えてほしい。

まずはどのようにすれば，左の作業を再現できるのか，フローチャートを作ってみましょう。



メモ欄



挿入ソートのコードを書いてみましょう

```
data=[9,4,7,2,3,8,6,1,5,0]
print(data,"←元のデータ")
N=len(data)









print(data,"←ソート後のデータ")
```

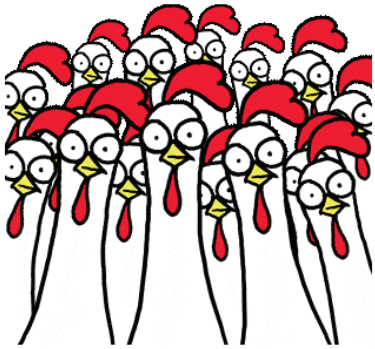
第６章　アルゴリズムとプログラミング
２節　プログラミングの実践
その１０

学習項目

- クイックソートのアルゴリズムを理解する

ソートの種類

ソートのアルゴリズムは古くから考えられており，何種類もある。
ここでは代表的なソートのアルゴリズムについて学習する。



① 選択ソート
② バブルソート
③ 挿入ソート
④ クイックソート

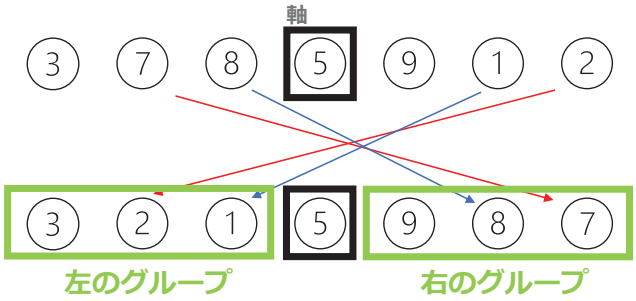
以降ばらばらの数値を昇順ソートするアルゴリズムを考えてみましょう。

④ クイックソート とは

クイックソートとは“基準の値”を選び，その値以下かその値以上かで，データを左右に分けることを繰り返し，並び替えを行う方法。

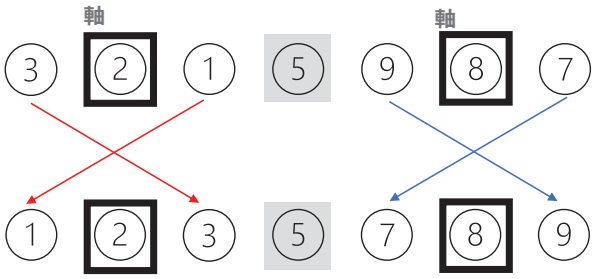
クイックソートは選択ソートやバブルソートより高速にデータを並び替えることができる，実用性の高いアルゴリズムです。

クイックソートのアルゴリズム



1．データを２つに分けるための基準値（軸）を
任意に選ぶ。（ここでは中央にある⑤を選ぶ）

2．軸以下のデータを軸の左に，軸以上のデータを
右に置くことで，データを左右のグループにわける

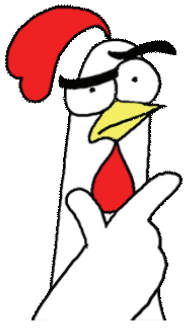


3．左右それぞれのグループで新たな軸を選び，同様に
軸以下，軸以上でデータを左右に配置する。

4．それ以上分割できなくなったところは，データの
位置が確定する。すべてのデータが分割できなく
なるまでこれを行うと，ソートが完了する。

クイックソートのアルゴリズム

2．軸以下のデータを軸の左に，軸以上のデータを
右に置くことで，データを左右のグループにわける



ん？　**左 右 に わ け る**

ってどういうアルゴリズムでやってるの？？？

```python
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)

def quick_sort(left,right):
    i=left
    j=right
    p=data[(left+right)//2]
    while True:
        while data[i]<p:
            i=i+1
        while data[j]>p:
            j=j-1
        if i>=j:
            break
        data[i],data[j]=data[j],data[i]
        i=i+1
        j=j-1
    if left < i-1:
        quick_sort(left,i-1)
    if right>j+1:
        quick_sort(j+1,right)

print(data,"←元のデータ")
quick_sort(0,n-1)
print(data,"←ソート後のデータ")
```

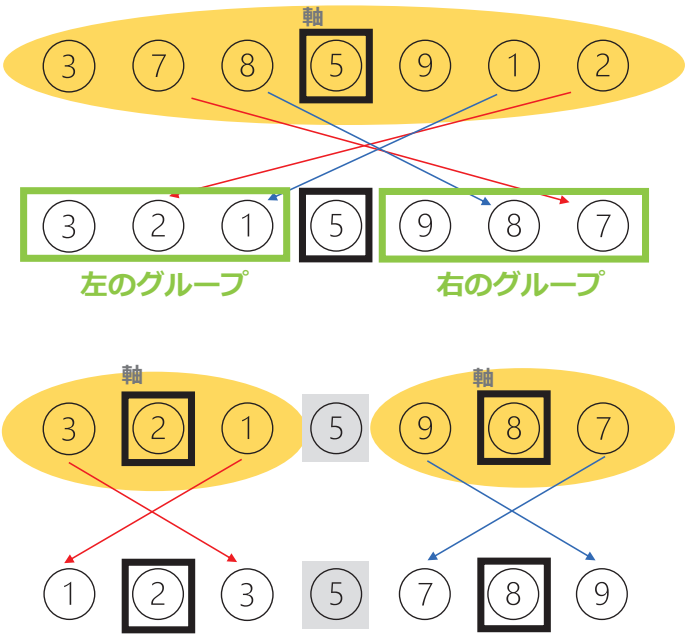
クイックソートのコード

左のコードをよんで
クイックソート実行時のコンピュータ内の
処理をわかりやすく説明しなさい。

フローチャートを使っても良いし
図や文章でも構いません。

…といってもナニコレ？
だと思うので、説明します！

クイックソートのアルゴリズム

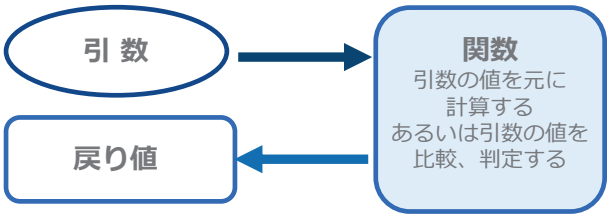


ここで，クイックソートのアルゴリズムは
集合に対して左右に分けるという動作を
集合を細分化して何度も繰り返していることに
気付く。

対称を変えながら同じ作業を繰り返すとき，
関数を設定し，利用することができる

※反復試行は変数を同じ規則で変えながら繰り返すのに対し，
関数は対称を自由に変えることができるイメージ。

関数とは



自分でつける

```
def ␣ OOO () :
␣ ␣ ␣ ␣ print(“▲▲”)
```

繰り返す処理の内容

※　関数名のあとに（）を記述する
※　引数をもたせるときは（）内に引数となる変数を記述する

関数の例

```
def volume_sphere(r):
    v=4*3.14*r*r*r/3
    return v

print("半径10cmの球の体積",volume_sphere(10),"cm3")
print("半径20cmの球の体積",volume_sphere(20),"cm3")
```

v の値を返すという意味

```
PS C:\Users\msaito\Documents\python> & C:/User
半径10cmの球の体積は 4186.666666666667 cm3
半径20cmの球の体積は 33493.333333333336 cm3
```

関数の r に10を代入した値が返ってくるので、ここには 4*3.14*10*10*10 の値が入っている

クイックソートのコード

```
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)

def quick_sort(left,right):
    i=left
    j=right
    p=data[(left+right)//2]
    while True:
        while data[i]<p:
            i=i+1
        while data[j]>p:
            j=j-1
        if i>=j:
            break
        data[i],data[j]=data[j],data[i]
        i=i+1
        j=j-1
    if left < i-1:
        quick_sort(left,i-1)
    if right>j+1:
        quick_sort(j+1,right)

print(data,"←元のデータ")
quick_sort(0,n-1)
print(data,"←ソート後のデータ")
```

ここの部分で関数を定めている

ここで実行している

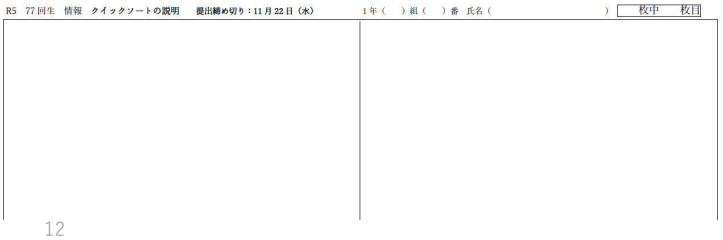

```
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)

def quick_sort(left,right):
    i=left
    j=right
    p=data[(left+right)//2]
    while True:
        while data[i]<p:
            i=i+1
        while data[j]>p:
            j=j-1
        if i>=j:
            break
        data[i],data[j]=data[j],data[i]
        i=i+1
        j=j-1
    if left < i-1:
        quick_sort(left,i-1)
    if right>j+1:
        quick_sort(j+1,right)

print(data,"←元のデータ")
quick_sort(0,n-1)
print(data,"←ソート後のデータ")
```


クイックソートのコード

左のコードをよんで
クイックソート実行時のコンピュータ内の
動きをわかりやすくまとめなさい。

フローチャートや**図**や**文章**をうまく組み合わせて
表現できると良いですね。
ワークシートの使い方も枚数も自由です。

※複数枚になる場合は何枚使用したかわかるように記入
しておいてください。

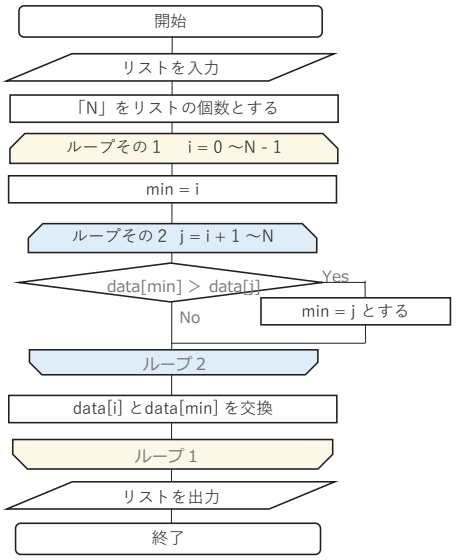


復習：フローチャート

■表１　おもなフローチャート記号（JIS X 0121より）

名称	記号	意味
端子		開始と終了
データ		データ入出力
処理		演算などの処理
判断		条件による分岐
ループ始端		繰り返しの始まり
ループ終端		繰り返しの終わり
定義済み処理		別な場所で定義された処理
線		処理の流れ

例：選択ソートのフローチャート



復習：フローチャート

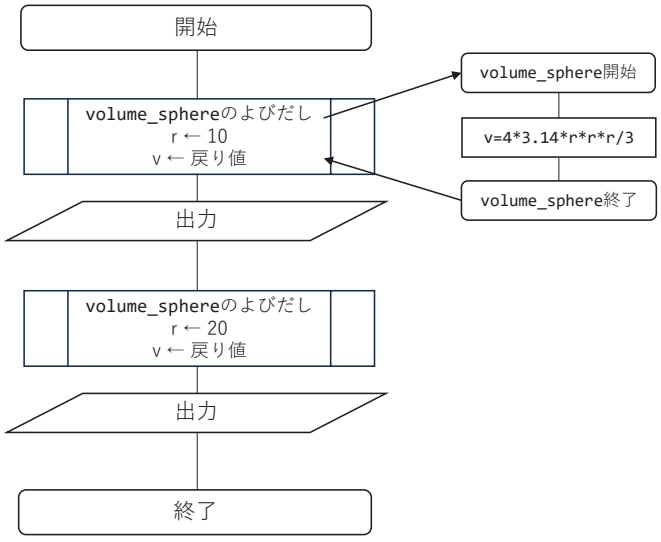
「関数」のフローチャートは

の記号を用いる。

さらに矢印で繋ぎ，横に関数の中の処理を書いておくと見やすいかもしれませんね！

```
def volume_sphere(r):
    v=4*3.14*r*r*r/3
    return v

print("半径10cmの球の体積",volume_sphere(10),"cm3")
print("半径20cmの球の体積",volume_sphere(20),"cm3")
```

例：左下のプログラムのフローチャート



第６章　アルゴリズムとプログラミング
２節　プログラミングの実践
その１１

学習項目

- アルゴリズムを比較し，その違いについて考察できる

ソートの種類

今まで学習してきたソートの種類を比較してみよう

① 選択ソート

② バブルソート

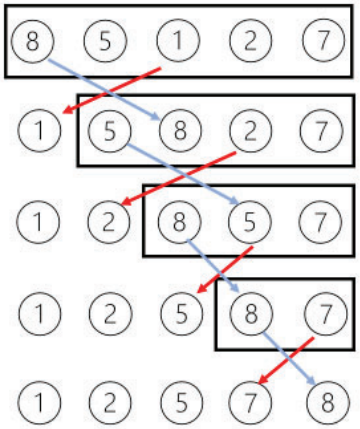
③ 挿入ソート

④ クイックソート

3

【復習】①選択ソート

選択ソートのアルゴリズム



1．データの中から最も小さな値を探し（ここでは①），先頭の⑧と入れ替える。

2．先頭の①を除いたデータの中から最も小さな値を探し（ここでは②），先頭の⑤と入れ替える。

3．先頭の①と②を除いたデータの中から最も小さな値を探し（ここでは⑤），先頭の⑧と入れ替える。

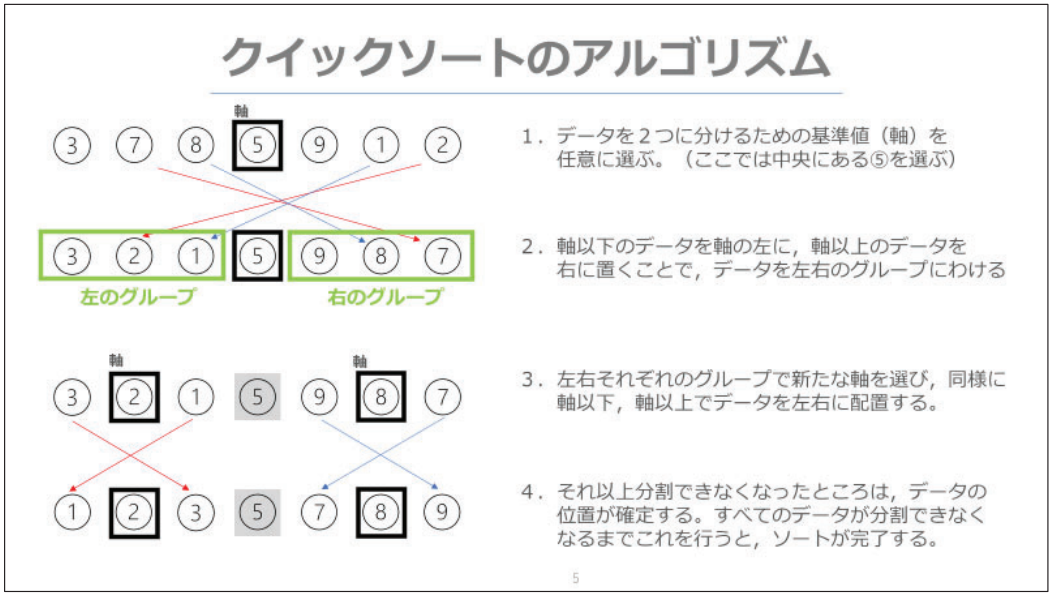
4．先頭の①②⑤を除いたデータの中から最も小さな値を探し（ここでは⑦），先頭の⑧と入れ替える。

5．ソート完了

7

4

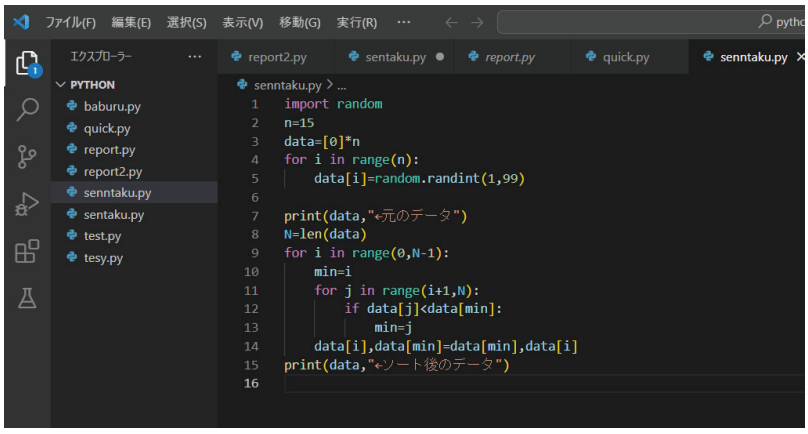
【復習】④クイックソート



比較してみましょう

```python
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)
```

ここの値を変更することで配列の個数を変更できる



配列の個数を増やして

選択ソート
クイックソート

をそれぞれ実行してみましょう

・・・どんな違いがありましたか？

実行時間を比較してみましょう

```
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)

print(data,"←元のデータ")
N=len(data)
for i in range(0,N-1):
    min=i
    for j in range(i+1,N):
        if data[j]<data[min]:
            min=j
    data[i],data[min]=data[min],data[i]
print(data,"←ソート後のデータ")
```

配列数	選択ソート	クイックソート

```
import random
n=15
data=[0]*n
for i in range(n):
    data[i]=random.randint(1,99)

def quick_sort(left,right):
    i=left
    j=right
    p=data[(left+right)//2]
    while True:
        while data[i]<p:
            i=i+1
        while data[j]>p:
            j=j-1
        if i>=j:
            break
        data[i],data[j]=data[j],data[i]
        i=i+1
        j=j-1
    if left < i-1:
        quick_sort(left,i-1)
    if right>j+1:
        quick_sort(j+1,right)

print(data,"←元のデータ")
quick_sort(0,n-1)
print(data,"←ソート後のデータ")
```


ソートの計算量

一般にアルゴリズムの計算量というのは少し難しいので，ここではざっくりとしたイメージを共有する。

プログラムを評価する方法として時間計算量というものがあり，これは答えをだすまでにどの程度の計算時間を必要とするかを考えるもの。正確に求めるには命令による時間と回数を調べ，すべて足し合わせればよい。が，普通はそこまで細かく考えない。

プログラムの文が実行される回数を調べて計算量の目安とすることが一般的である。これをアルゴリズムの解析（analysis of algorithms）という。

ソートでいうのであれば，**数の比較や入れ変えをざっくり何回行っているか**を考えることになる。

選択ソートの計算量

1000個のデータを並び替えるとすると・・・

１番左を決めるのに1000回比較，２番目を決めるのに999回比較
↓
1000＋999＋998＋・・・＋３＋２＋１＋（入れ替えとか）
↓
1001×500＋（入れ替えとか）
↓
ざっくり　1000×500

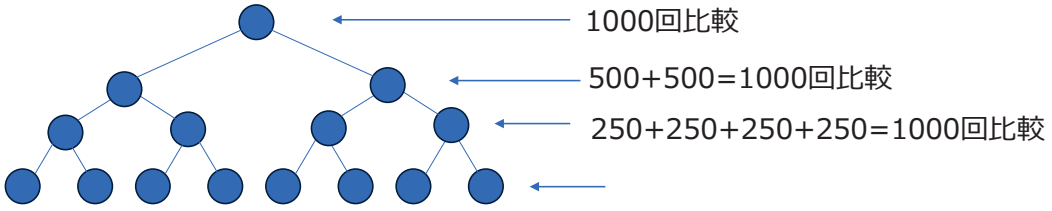
n 個のデータを並び替えるとすると

$$n \times \frac{n}{2} = \frac{n^2}{2}$$

※これは n^2 に依存するので計算量を $\boldsymbol{O(n^2)}$ （オーダーnの２乗）という

クイックソートの計算量

1000個のデータを２つに分ける作業を考えると，2^{10}が1024であることから10階層分（クイックソートの中のクイックソートの中の…を10回），２つに分ける作業を行わなくてはいけないことがわかる



ざっくり　1000×10

n 個のデータを並び替えるとすると

$$n \times \log_2 n$$

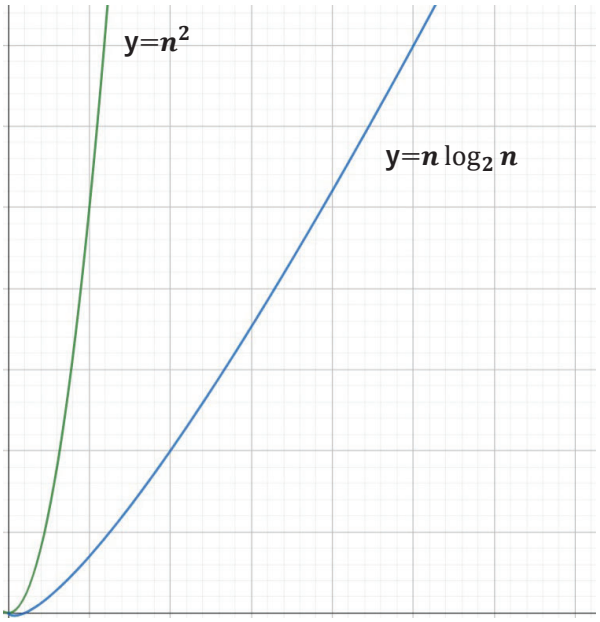
$2^{●} = n$ をみたす●を$\log_2 n$ と表す

※計算量を $\boldsymbol{O(n \log_2 n)}$ という

クイックソートの計算量

$O(n^2)$ と $O(n \log_2 n)$ を比較すると，
右図のように数が大きくなればなるほど差が大きくなる
細かい計算は省いているが，概ね大きく変動する部分は
これだけ差ができる。

だから
クイックソートは
早いのだ



アルゴリズムの比較

ソートのアルゴリズムは，**わかりやすさ，時間計算量，領域計算量**など様々な視点で比較ができるだろう。

- ✓ **複数のアルゴリズムを様々な視点から比較すること**
- ✓ **使用者やその状況を踏まえさらに改良すること**
- ✓ **時間，領域の効率化を考えさらに効率すること**

を意識し，プログラミングだけでなく，様々な日常にあふれるアルゴリズムに応用していければよいですね