1 ソートアルゴリズムの評価

学習目標とキーワード

学 習 目 標　ソートプログラムには複数のアルゴリズムがあり、それれに特性があることを理解しましょう。

キーワード　・計算速度　・計算量　・安定性　・メモリ使用量

ソートの評価

ソートアルゴリズムの重要な評価ポイントは速度ですが、その他にも使用するコンピュータの性能やシステムの状況により重要な評価となるアルゴリズムの特性があります。

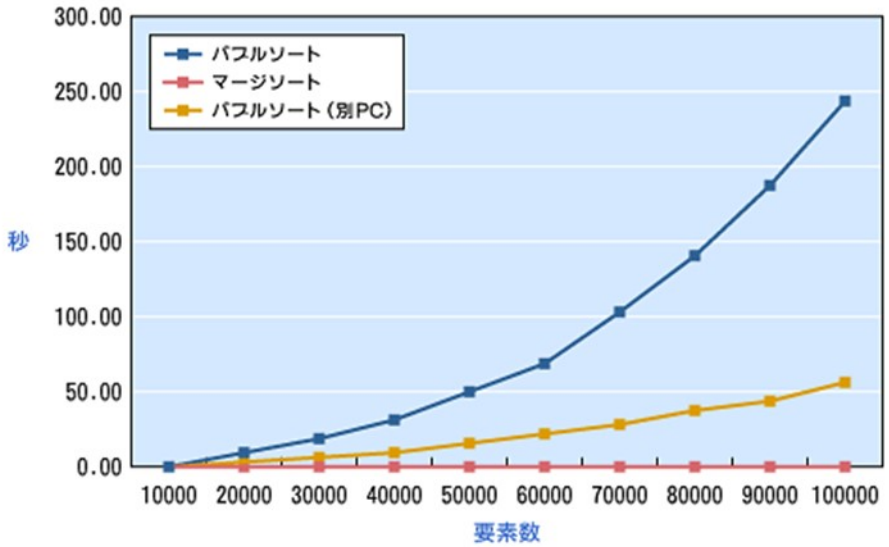
1　計算量

計算速度を左右するのが計算量です。データ読出　→　比較　→　データ書出　の計算量とデータ数の関係が直線的だと、データ数と処理時間は比例するので処理時間は極端に遅くなりません。しかし、計算量とデータ数が指数的だとデータ数が多くなるほど処理時間が遅くなります。

代表的なソートアルゴリズムと計算量

ソート	平均	最悪	備考
選択ソート	n^2	n^2	
バブルソート	n log n	n log n	ソート済のデータで時間が掛かる
クイックソート	n log n	n^2	
ヒープソート	n log n	n log n	
マージソート	n log n	n log n	

ソートプロトコル別データ数と処理時間の関係

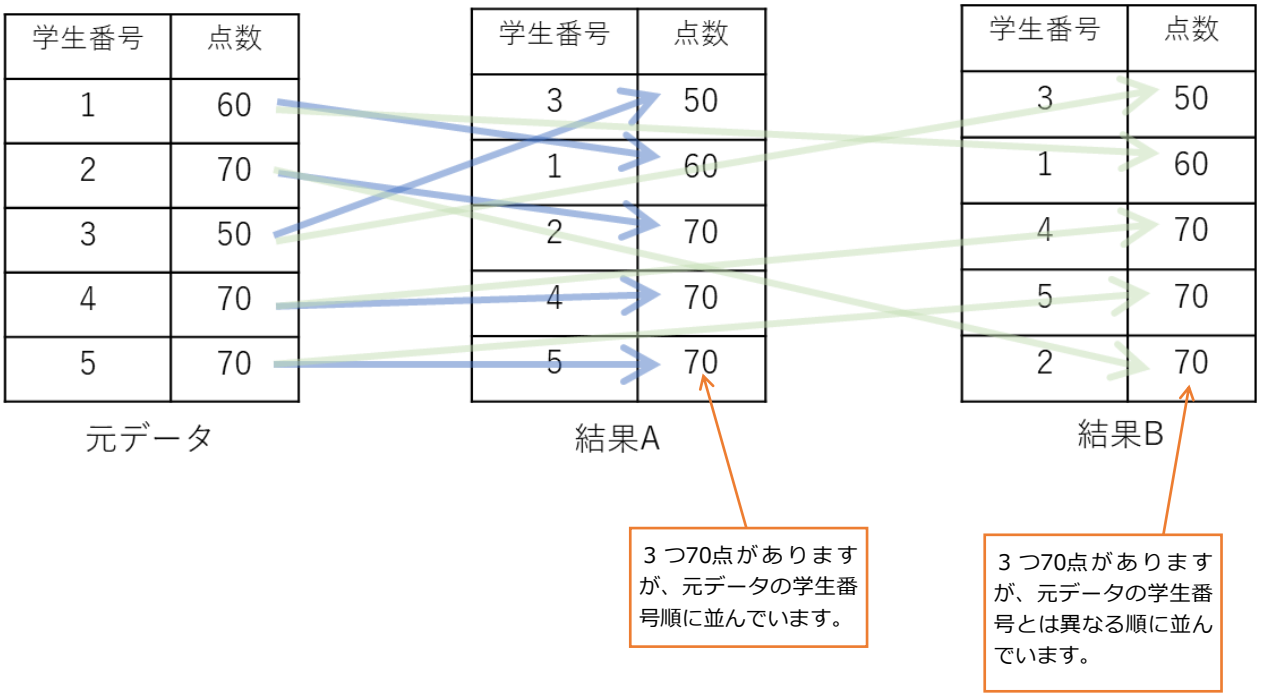


いまから始めるアルゴリズム（2）

https://www.atmarkit.co.jp/ait/articles/0710/18/news130.html

2　安定性

同じ値が入ったデータをソートした時、元のデータの並び順が変化しない状況を安定化と言います。例えば、点数でソートした場合結果Aでは同じ点数の学生番号は番号順で、結果Bは学生番号が変化してしまいます。この場合、結果Aの方が安定性が良いと言います。ソートの優先順位が、１点数、２学籍番号の場合、結果Aは何もすることがありませんが、結果Bでは点数でソートした後に、改めて学籍番号をソートする必要があります。



3　メモリ使用量

入力データの格納域以外にメモリが必要なるソートがあります。ストレージ量が限られている場合は、重要な評価になります。

入力データの格納域以外に必要なメモリと安定性

ソート	メモリ使用量	安定性
選択ソート	1	（実装による）
バブルソート	1	安定
クイックソート	n	×
ヒープソート	1	×
マージソート	n	安定

4　その他

ソートの速度を測定する場合、一様にランダムなデータを使用しますが、実際のデータには偏りがあります。例えば、数百万件のソート済の顧客データに数千件の新規顧客のデータを加えてソートする。ソート済の数万件のデータを数件結合してソートする。など、ソートするデータの状況により最適なアルゴリズムを選びます。

2 選択ソート（select sort）

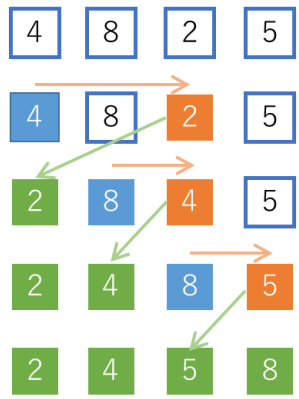
学習目標とキーワード

学 習 目 標　選択ソートのアルゴリズムを学び、プログラムを理解しましょう。

キーワード　・選択ソート　・アルゴリズム　・グラフアニメーション

1 選択ソートのアルゴリズム

選択ソートは配列の最小値(最大値)を持つ要素を探して、それを配列の先頭要素と交換することで整列を行うアルゴリズムです。交換数が少ない場合はバブルソートより速くなります。



ソート前のリスト

4 を初期値として 4 より小さい値を探して入れ替える。最後のデータまで探すと最小値 2 が見つかる

2 を先頭に決定する。次の 8 を初期値として同様に最小値を探す。最小値 4 が見つかる

4 を2番目に決定する。次の 8 を初期値として同様に最小値を探す。最小値 5 が見つかる

5 を 3 番目に決定する。最後の 8 が自動的に決定する。

2 プログラムを実行してみましょう

プログラム

```
#選択ソート
#プログラム中の①～⑪はアニメーション用のプログラムです
#アニメーション表示なしで高速に実行するためには①～⑪（最小で⑦～⑪）をコメント文にしてください。
import numpy.random as rd               # numpy.randomモジュールをインポートして名前をrdとする
from matplotlib import animation,rc     # ①アニメーションのモジュールをインポートする（Google Colab用）
# ②%matplotlib nbaggはアニメーション機能の開始（Jupyter 用。Google Colabでは①行必併用。コメント分不可）
%matplotlib nbagg
rc('animation',html='jshtml')           # ③アニメーションをjavascriptで実行する設定
import matplotlib.pyplot as plt         # ④グラフを使うライブラリ　名前をpltにします
fig = plt.figure()                      # ⑤グラフの描画領域の設定
ims=[]                                  # ⑥グラフのイメージ（画像）のリスト定義
#選択ソートの関数
def selectionsort(data):                    # 並び替え前のランダムなデータリストを受け取る
   for i in range(0,len(data),1):           # iは０からリスト数－１まで１ずつ増える
     for j in range(i+1,len(data),1):       # jはi＋１からリスト数－１ま１ずつ増える
        if data[i]>data[j]:                 # data[i]とdata[j]を比べdata[j]が小さければ入れ替える
          temp = data[i]                    # 入れ替え処理
          data[i] = data[j]                 #
          data[j] = temp                    #
     im = plt.bar(b,data,color="orange")    # ⑦棒グラフを描画してイメージをimに入れる
     ims.append(im)                         # ⑧イメージimをイメージリストimsに追加していく
#メインプログラム
data_list = []                              #データリストの定義
data_list = rd.randint(1,100,100)           #１から１００の整数を１００個作りdata_listに入れる
b=range(len(data_list))                     # ⑨グラフのX軸用に、データリストの要素数分のリストを作る

#ソート前
print(" ソート前（先頭10個のデータ）",data_list[0:10])#ソート前データ、データ数が多いので先頭１０個だけ表示
selectionsort(data_list)                    # ソート実行
print(" ソート後（先頭10個のデータ）",data_list[0:10])#ソート後データ、データ数が多いので先頭１０個だけ表示
anim = animation.ArtistAnimation(fig,ims,interval=100)#⑩グラフのイメージを表示エリアに100ms間隔で表示する設定
anim  #⑪グラフのアニメーションを表示
```

出力

```
ソート前（先頭10個のデータ） [ 6 37 78 30 97 1 4 6 91 20]
ソート後（先頭10個のデータ） [1 2 3 4 4 5 5 6 6 6]
```

グラフアニメーション

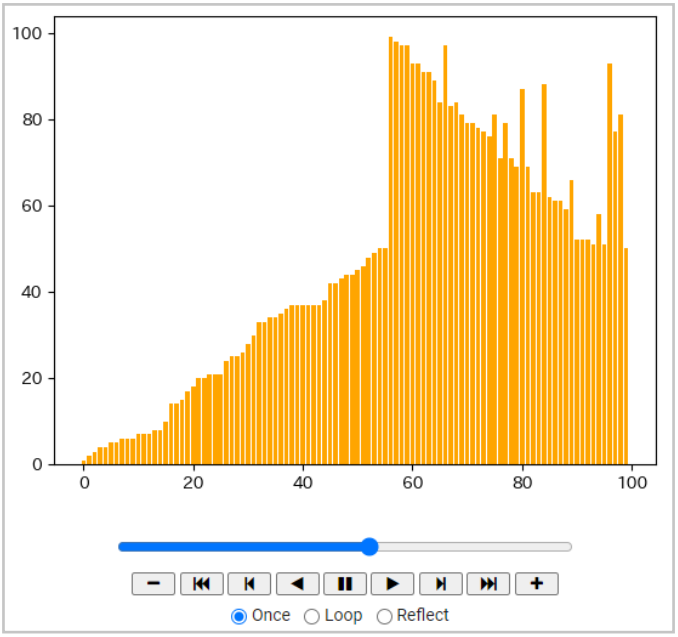
　データリストがソートされて行く様子をアニメーションでみることができます。

最小値から順番にソートしていくことが分かります。

グラフアニメーションの制作手法

　グラフアニメーションは、プログラムの途中でグラフを作成して画像ファイルのリストを作ります。

このファイルをプレイヤーでパラパラ漫画の手法で再生しています。



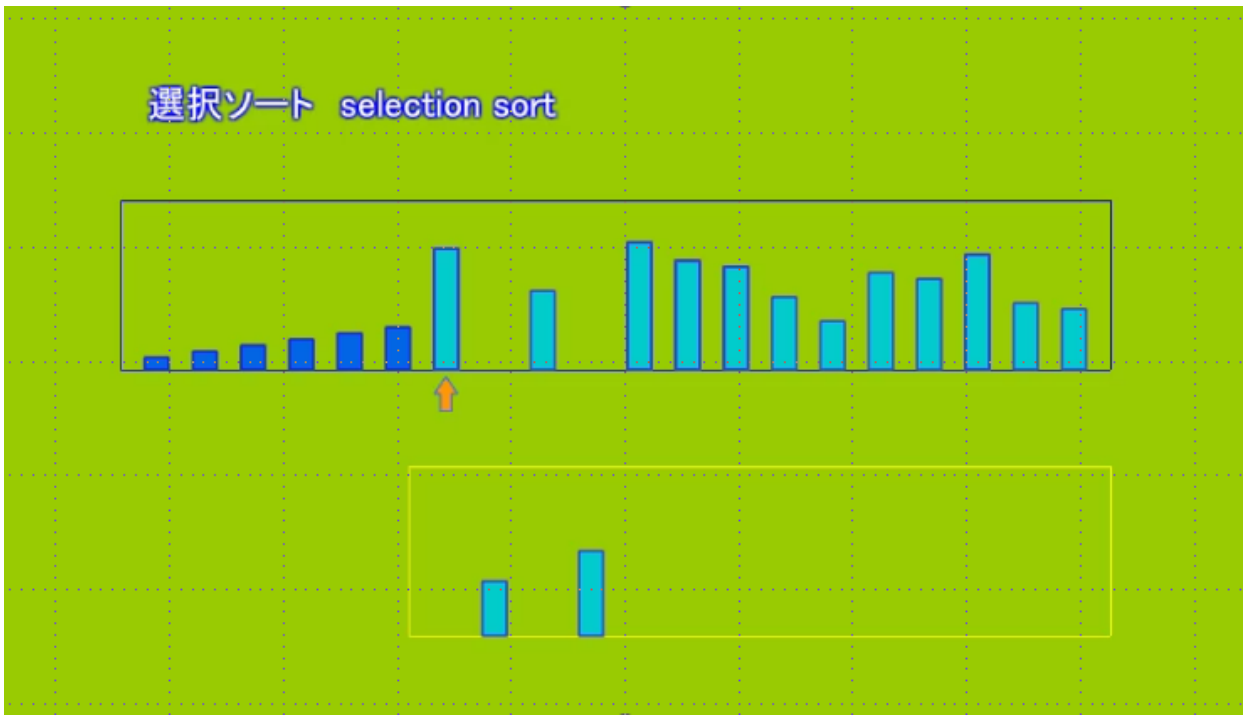
課題 2-1

(1) データを大きい順にソートするようにプログラムを変えてみましょう。

3 選択ソートアルゴリズムのアニメーション

選択ソートアルゴリズムのアニメーションを見て、データの操作を理解しましょう。

動画はURLをクリックするがブラウザにコピーして実行してください。



動画URL　https://youtu.be/JAWOSwwh8KU

3 交換ソート（bubble sort）

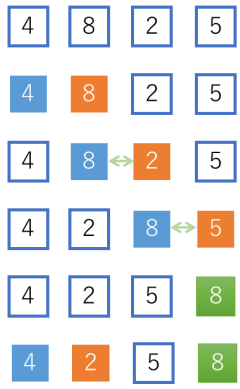
学習目標とキーワード

学 習 目 標　交換ソートのアルゴリズムを学び、プログラムを理解しましょう。

キーワード　・交換ソート　・アルゴリズム　・グラフアニメーション

1 交換ソート

交換ソートは隣り合うふたつの要素の値を比較して、条件に応じで交換するアルゴリズムです。データが泡（バブル bubble）のように動くのでバブルソートと呼ばれます。



ソート前のリスト

隣合った値を比べ、右が大きければそのままに、小さければ交換します。ここでは４＜８なのでそのままにします

一つ右にずらして２つを比べます。ここでは８＞２なので交換します。

一つ右にずらして２つを比べます。ここでは８＞４なので交換します。

８が最大値に決定しました。

左に戻り、同じ方法で２番目に大きい値を決めます。

以後すべてが決まるまで繰り返します。

2 プログラムを実行してみましょう

プログラム

```
# 交換ソート
#プログラム中の①～⑪はアニメーション用のプログラムです
#アニメーション表示なしで高速に実行するためには①～⑪（最小で⑦～⑪）をコメント文にしてください。
import numpy.random as rd             # numpy.randomモジュールをインポートして名前をrdとする
from matplotlib import animation,rc   # ①アニメーションのモジュールをインポートする（Google Colab用）
# ②%matplotlib nbaggはアニメーション機能の開始（Jupyter 用。Google Colabでは①行必併用。コメント分不可）
%matplotlib nbagg
rc('animation',html='jshtml')         # ③アニメーションをjavascriptで実行する設定
import matplotlib.pyplot as plt       # ④グラフを使うライブラリ　名前をpltにします
fig = plt.figure()                    # ⑤グラフの描画領域の設定
ims=[]                                # ⑥グラフのイメージ（画像）のリスト定義

# 交換ソートの関数
def bubblesort(data):
   for i in range(len(data)):           # iは０からリスト数－１まで１ずつ増える
     for j in range(len(data)-i-1):     # jは０からリスト数－１から１づつ減る数まで増える
        if data[j] > data[j+1]:         # 隣（グラフでは右）のデータと比較して大きければ入れ替える
          temp = data[j]                # 入れ替え処理
          data[j] = data[j+1]           # jの繰り返しで一番大きいデータがうしろ（グラフでは右）に置かれる
          data[j+1] = temp              #
     im = plt.bar(b,data,color="lightgreen") # ⑦棒グラフを描画してイメージをimに入れる
     ims.append(im)                     # ⑧イメージimをイメージリストimsに追加していく
data_list = []                          # リストの定義
data_list = rd.randint(1,100,100)    #１から１００の整数を１００個作りdata_listに入れる
b=range(len(data_list))                 # ⑨グラフのX軸用に、データリストの要素数分のリストを作る
print(" ソート前（先頭10個のデータ）",data_list[0:10])
bubblesort(data_list)                   # ソート実行
print(" ソート後（先頭10個のデータ）",data_list[0:10])
anim = animation.ArtistAnimation(fig,ims,interval=100) # ⑩グラフのイメージを表示エリアに100ms間隔で表示する設定
anim # ⑪グラフのアニメーションを表示
```

出力

```
ソート前（先頭10個のデータ） [68 27 2 77 24 3 86 15 53 48]
ソート後（先頭10個のデータ） [1 2 3 4 5 5 6 6 8 9]
```

グラフアニメーション

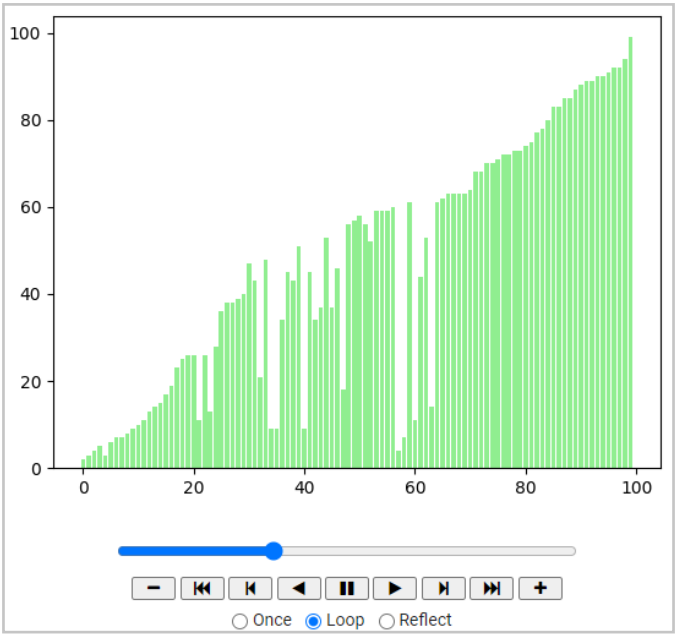
　データリストがソートされて行く様子をアニメーションでみることができます。

全体的にソートが行われ、最大値から決定していくことが分かります。

グラフアニメーションの制作手法

　グラフアニメーションは、プログラムの途中でグラフを作成して画像ファイルのリストを作ります。

このファイルをプレイヤーでパラパラ漫画の手法で再生しています。



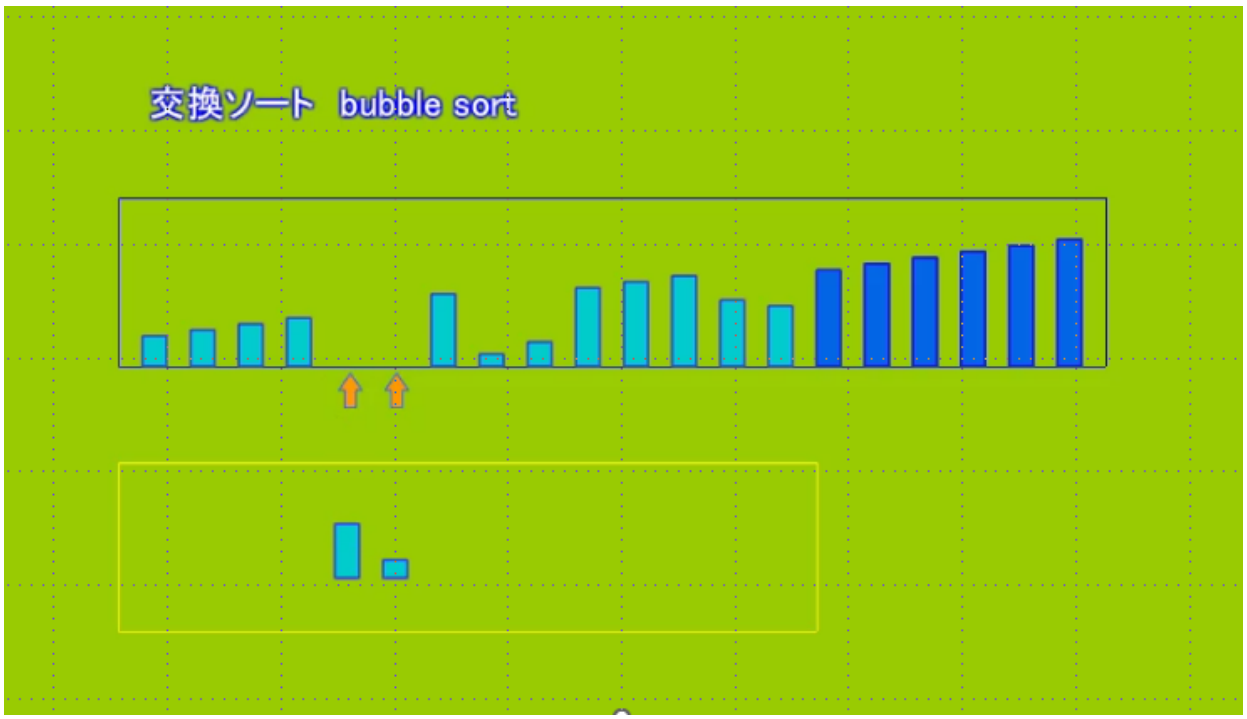
課題 3-1

(1) データを大きい順にソートするようにプログラムを変えてみましょう。

3 交換ソートアルゴリズムのアニメーション

交換ソートアルゴリズムのアニメーションを見て、データの操作を理解しましょう。

動画はURLをクリックするがブラウザにコピーして実行してください。



動画URL　https://youtu.be/rsfnWVQShcE

4 クイックソート（quick sort）

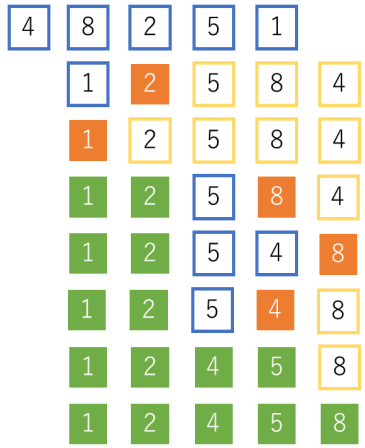
学習目標とキーワード

学 習 目 標　クイックソート のアルゴリズムを学び、プログラムを理解しましょう。

キーワード　・クイックソート ソート　・ピボット　・アルゴリズム　・グラフアニメーション

1 クイックソートソート

クイックソートはピボットと呼ぶデータを決め、ピボットの両側のデータを大きいグループと小さいグループに移動します。更にそのグループでピボットを決めてピボットより大きいグループと小さいグループに移動していきます。この操作を比較するデータが無くなるまで繰り返します。高速ですが、ソート済のデータをソートすると遅くなる場合があります。



ソート前のリスト

中心の２をピボットとして、左側に小さい値、右側に大きい値を移動します

左グループの中心１をピボットして交換を試みます

交換できないので、１、２は決定します。右グループの８をピボットにして交換します

５、４が左へ移動しました

左グループの４をピボットにして交換します

これ以上交換できないので４、５が決定します　８が最大値に決定しました。

ピボットの決め方は、中央値や平均値などいくつかの方法があります

2 プログラムを実行してみましょう

プログラム

```
# クイックソート
#プログラム中の①～⑪はアニメーション用のプログラムです
#アニメーション表示なしで高速に実行するためには①～⑪（最小で⑦～⑪）をコメント文にしてください。
import numpy.random as rd             # numpy.randomモジュールをインポートして名前をrdとする
from matplotlib import animation,rc   # ①アニメーションのモジュールをインポートする（Google Colab用）
# ②%matplotlib nbaggはアニメーション機能の開始（Jupyter 用。Google Colabでは①行必併用。コメント分不可）
%matplotlib nbagg
rc('animation',html='jshtml')         # ③アニメーションをjavascriptで実行する設定
import matplotlib.pyplot as plt       # ④グラフを使うライブラリ　名前をpltにします
fig = plt.figure()                    # ⑤グラフの描画領域の設定
ims=[]                                # ⑥グラフのイメージ（画像）のリスト定義

#クイックソートの関数
def quick_sort(data,left, right):     #引数はデータリスト、左開始点、右開始点
    i = left                          # 左の開始点のインデックス（データ位置）
    j = right                         # 右の開始点のインデックス（データ位置）
    pivot =data[ (left + right) // 2]# 左右の中心位置のデータ（ピボット）を求める(//は商の演算)
    # ソート対象のインデックスを探索
    while True:
        while data[i] < pivot:        #ピボットよりデータが小さかったら（正しい順）
            i = i + 1                 #探索するデータを右にずらす
        while data[j] > pivot:        #ピボットよりデータが大きかったら（正しい順）
            j = j - 1                 #探索するデータを左にずらす
        # 無限ループ終了条件
        if i >= j:                    #探索する位置が右と左が同じか、左が右を超えたら終わり
            break                     #関数の終わり
        # ピボットの両側にあるデータを交換
        tmp = data[i]                 #tmpを介した交換
        data[i] = data[j]             #
        data[j] = tmp                 #
```

つづく

```
        # 範囲を一つ狭める
        i = i + 1                       #交換が終わったので左の開始点を増やす
        j = j - 1                       #交換が終わったので右の開始点を減らす

        im = plt.bar(b,data,color="violet") #⑦棒グラフを描画してイメージをimに入れる
        ims.append(im)                  #⑧イメージimをイメージリストimsに追加していく

    # 再帰処理　交換の範囲を現在のピボットの左側または右側に設定して再度ソート関数に渡す
    if left < i - 1:                    #左開始点より探索位置が大きかったら（まだソートできていない）
        quick_sort(data,left, i - 1) #左開始点からピボット位置手前までをソート
    if right > j + 1:                   #右開始点より探索位置が小さかったら（まだソートできていない）
        quick_sort(data,j + 1, right)#右開始点からピボット位置手前までをソート
data = []                               # リストの定義
data = rd.randint(1,100,100)            #１から１００の整数を１００個作りdata_listに入れる
b=range(len(data))                      #⑨グラフのX軸用に、データリストの要素数分のリストを作る
print(" ソート前（先頭10個のデータ）",data[0:10])
quick_sort(data,0,len(data)-1)          # ソート実行
print(" ソート後（先頭10個のデータ）",data[0:10])
anim = animation.ArtistAnimation(fig,ims,interval=100)#⑩グラフのイメージを表示エリアに100ms間隔で表示する設定
anim    #⑪グラフのアニメーションを表示
```

出力

```
ソート前（先頭10個のデータ） [74 90 97 91 85 27 13 11 42 13]
ソート後（先頭10個のデータ） [ 1 1 7 10 11 11 12 13 13 16]
```

グラフアニメーション

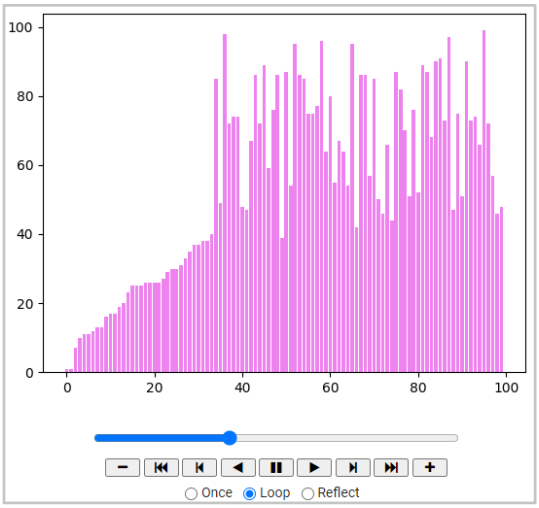
　データリストがソートされて行く様子をアニメーションでみることができます。

ピボットで左右に分割しながらソートしていきます。左側から先にソートしていくことが分かります。

グラフアニメーションの制作手法

　グラフアニメーションは、プログラムの途中でグラフを作成して画像ファイルのリストを作ります。

このファイルをプレイヤーでパラパラ漫画の手法で再生しています。



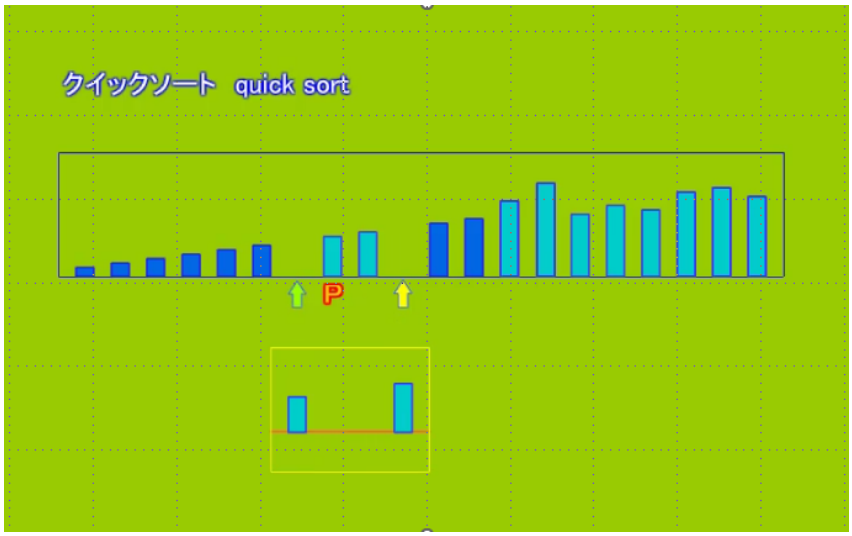
課題 4-1

(1) データを大きい順にソートするようにプログラムを変えてみましょう。

3　クイックソートアルゴリズムのアニメーション

　クイックソートアルゴリズムのアニメーションを見て、データの操作を理解しましょう。

　動画はURLをクリックするがブラウザにコピーして実行してください。



動画URL　https://youtu.be/KeyYcPy8BEY

5 ヒープソート（Heap sort）

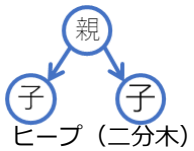
学習目標とキーワード

学 習 目 標　ヒープソートのアルゴリズムを学び、プログラムを理解しましょう。

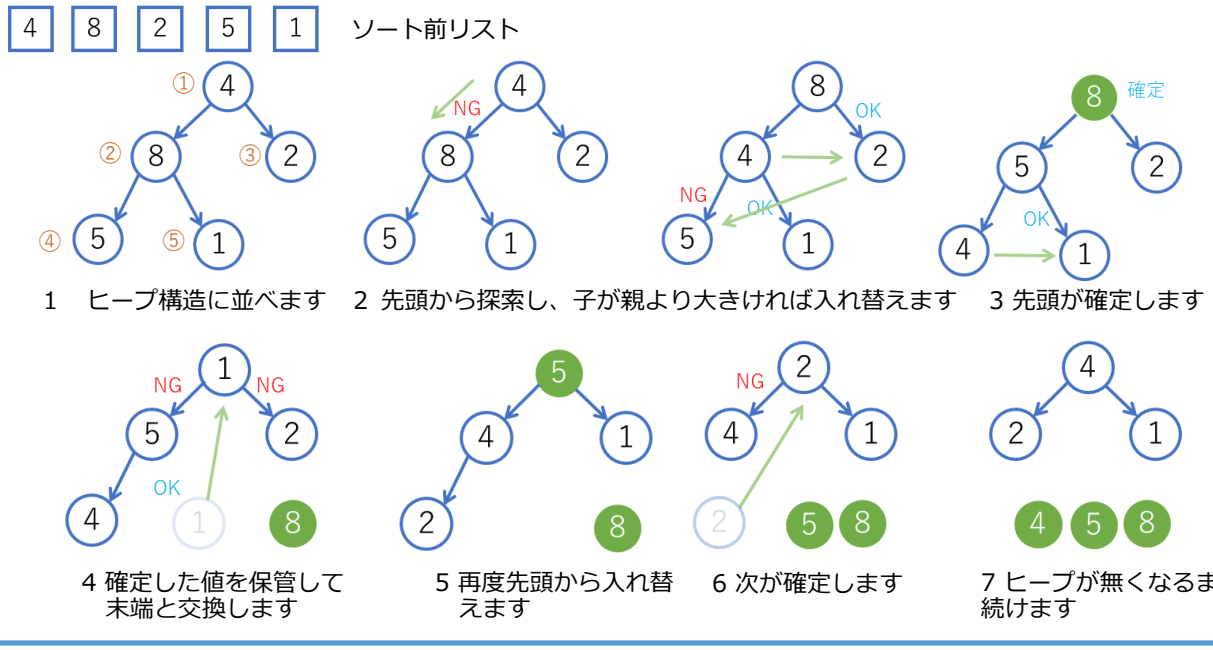
キーワード　・ヒープソート ソート　・ヒープ（二分木）　・アルゴリズム　・グラフアニメーション

1 ヒープソート

ヒープは、「親要素は子要素より常に等しいか大きい」（逆もある）と言う木構造のことで、このヒープを完成することでソートします。ソートのための特別なメモリ領域を必要としないので、大量のデータのソートに向いています。



ヒープ（二分木）



2 プログラムを実行してみましょう

プログラム

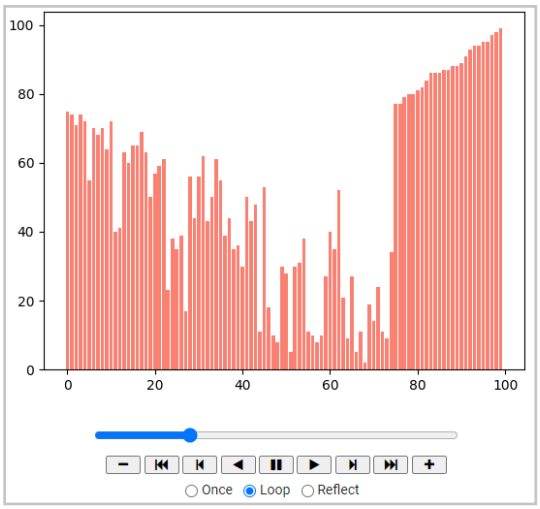
```
# ヒープソート
#プログラム中の①～⑪はアニメーション用のプログラムです
#アニメーション表示なしで高速に実行するためには①～⑪（最小で⑦～⑪）をコメント文にしてください。
import numpy.random as rd             # numpy.randomモジュールをインポートして名前をrdとする
from matplotlib import animation,rc   # ①アニメーションのモジュールをインポートする（Google Colab用）
# ②%matplotlib nbaggはアニメーション機能の開始（Jupyter 用。Google Colabでは①行必併用。コメント分不可）
%matplotlib nbagg
rc('animation',html='jshtml')         # ③アニメーションをjavascriptで実行する設定
import matplotlib.pyplot as plt       # ④グラフを使うライブラリ　名前をpltにします
fig = plt.figure()                    # ⑤グラフの描画領域の設定
ims=[]                                # ⑥グラフのイメージ（画像）のリスト定義

# ヒープソートの関数
def heap_sort(data):#引数はデータリスト
    i = 0                             #開始位置
    n = len(data)                     #終了位置（データの数）
    #ヒープの構成
    while(i < n):#開始位置が終了位置より小さいとき
        upheap(data, i)#[i]を親としたピープを構成する関数へ
        i += 1                        #開始位置を一つ増やします
    while(i > 1):# 最終位置から先頭の前まで繰り返し
        i -= 1   #
        data[0] , data[i] = data[i] , data[0]#[0]の最大値と[i]を交換
        # ヒープの再構成
        downheap(data, i-1)#ダウン関数へ
        im = plt.bar(b,data,color="salmon") #⑦棒グラフを描画してイメージをimに入れる
        ims.append(im) #⑧イメージimをイメージリストimsに追加していく
```

つづく

```
# ルート[0]をヒープ(0～n)の最適な位置へ移動
def downheap(data, n):#
    if n==0: return#終了位置が０なら戻る
    parent = 0#親を０番とする
    while True:#
        child = 2 * parent + 1 # data[n]の子要素（親の右となり）
        if child > n:# 要素外へ出るとbrake（子供の位置が終了点より大きい）
            break
        if (child < n) and data[child] < data[child + 1]:# 隣の子がいてかつ 左<右 なら右の子を見る
            child += 1
        if data[parent] < data[child]:# 親が子より小さい場合
            data[child] , data[parent] = data[parent] , data[child]#親子を入れ替え
            parent = child; # 交換後のインデックスを保持
        else:
            break
#ヒープを構成する関数
def upheap(data, n):
    while n:
        parent = int((n - 1) / 2)#位置nの親位置を計算する
        if data[parent] < data[n]:#もし親が子より小さかったら
            data[n] , data[parent] = data[parent] , data[n]#入れ替え
            n = parent
        else:
            break
data_list = []                          # リストの定義
data_list = rd.randint(1,100,100)#１から１００の整数を１００個作りdata_listに入れる
b=range(len(data_list))  #⑨グラフのX軸用に、データリストの要素数分のリストを作る(初期表示)
im = plt.bar(b,data_list,color="salmon") #⑦棒グラフを描画してイメージをimに入れる（初期表示）
ims.append(im) #⑧イメージimをイメージリストimsに追加していく
print(" ソート前（先頭10個のデータ）",data_list[0:10])
heap_sort(data_list)            # ソート実行
print(" ソート後（先頭10個のデータ）",data_list[0:10])
anim = animation.ArtistAnimation(fig,ims,interval=100)#⑩グラフのイメージを表示エリアに100ms間隔で表示する設定
anim#⑪グラフのアニメーションを表示
```

出力

```
ソート前（先頭10個のデータ） [74 90 97 91 85 27 13 11 42 13]
ソート後（先頭10個のデータ） [ 1 1 7 10 11 11 12 13 13 16]
```



グラフアニメーション

　データリストがソートされて行く様子をアニメーションでみることができます。

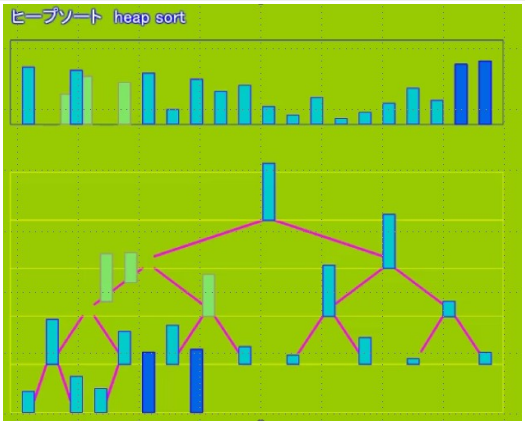
ヒープ構造ができるとすぐに最大値を一番最後に移します。最大値から決まっていく様子が分かります。

課題 5-1

ヒープソートでは、1次元のリスト（配列）の番号をヒープ構造の親子関係に変換してソートしています。

(1)　親の番号から子の番号はどのように計算していますか

(2)　子の番号から親の番号はどのように計算していますか



動画URL　https://youtu.be/7FcbbUqlh2g

3 ヒープソートアルゴリズムのアニメーション

　ヒープソートアルゴリズムのアニメーションを見て、データの操作を理解しましょう。

　動画はURLをクリックするがブラウザにコピーして実行してください。