令和６年度富山県高等学校教育研究会情報部会

「プログラミング指導の実践」

学　　校　　名　金沢大学附属高校
指導者 職・氏名　教諭・斎藤　瑞紀

1　生徒の状況

本校生徒は附属中学校以外の中学校から約半分の生徒が入学しており，中学校までの技術家庭で体験したプログラミングの内容は様々である。また，本校では個人でパソコンを用意しており，持っているデバイスは個人で異なる。プログラミング言語はPythonを使用し，Visual Studio Codeで実行しており，他教科で使用するアプリケーションとともに，事前にセットアップの時間をとり実行できる環境を整えている。

１学期は,問題解決方法やデータの分析,メディアと情報デザインの分野を行った。Word,Excel,PowerPointを使用し，パソコンの基本操作に不慣れだった生徒が少しずつ慣れるよう授業設計している。２学期は法規や権利，モデル化シミュレーションの後，プログラミング分野を行っている。

2　プログラミング学習における教材選定の理由

プログラミング学習では，学習を進めるうちに，「なぜ」「どうして」よりも，とにかく実行して正しい答えが出てくることがゴールになりがちだと感じている。また，深く学ばせるだけの時間がないことも悩ましい。

昨年度は，最終的に様々なアプリケーションソフトウェアがもつ“検索”や“並べ替え”などの機能を実現することで，生活の中で情報技術が問題解決に活かされていることを実感させたいと考えた。また，アルゴリズムの表現方法や正確に表現する重要性を学習しながら，アルゴリズムによる効率の違いを実感させたいと考えた。

そこで最終的な課題を「クイックソートのアルゴリズムの理解」に設定した。これは関数の有用性や，ほかの選択ソート，バブルソート，挿入ソートなど古典的なアルゴリズムと比較することでアルゴリズムによる効率の違いを考えさせたいと考えたからである。単元全体の流れは以下の通りである。

単元（題材）の指導と評価の計画（総時数　11　時間）

ねらい・主な学習活動	
１．基本制御構造を理解し，簡単なアルゴリズムを設計し，プログラムを実行できる	3 時間
２．基本制御構造を用いた簡単アルゴリズムを考え，コーディングできる（レポート作成）	1 時間
３．線形探索のアルゴリズムを理解し，リストを用いてプログラムをコーディングできる。	1 時間
４．選択ソート，バブルソート，挿入ソートのアルゴリズムを理解し，プログラムをコーディングできる。	3 時間
５．コードを読み，クイックソートのアルゴリズムを理解し，表現する。	2 時間
６．アルゴリズムの違いを理解する。	1 時間

3　指導方針

基本制御構造や，入出力，演算子，変数，配列など基本的な知識技能を学習する過程で，簡単なアルゴリズムを考える機会を設けながら授業を展開する。（他の教科との連携を考え，数学の期待値が難しすぎず考えやすそうだと感じた。）

その後古典的なソートのアルゴリズムを学習し，フローチャートを書いたり，フローチャートをもとにコーディングしたりする。特にバブルソートなどはアルゴリズムがわかりやすいが，それをどのように入力すればよいか悩む生徒も多いので，ここでフローチャートの有用性を実感させたい。また，生徒にとって「少し悩んだら出来る」くらいの難易度の課題を与えたいので，フローチャートも穴埋めから始め，少しずつ自力で作れるように授業展開を工夫したい。

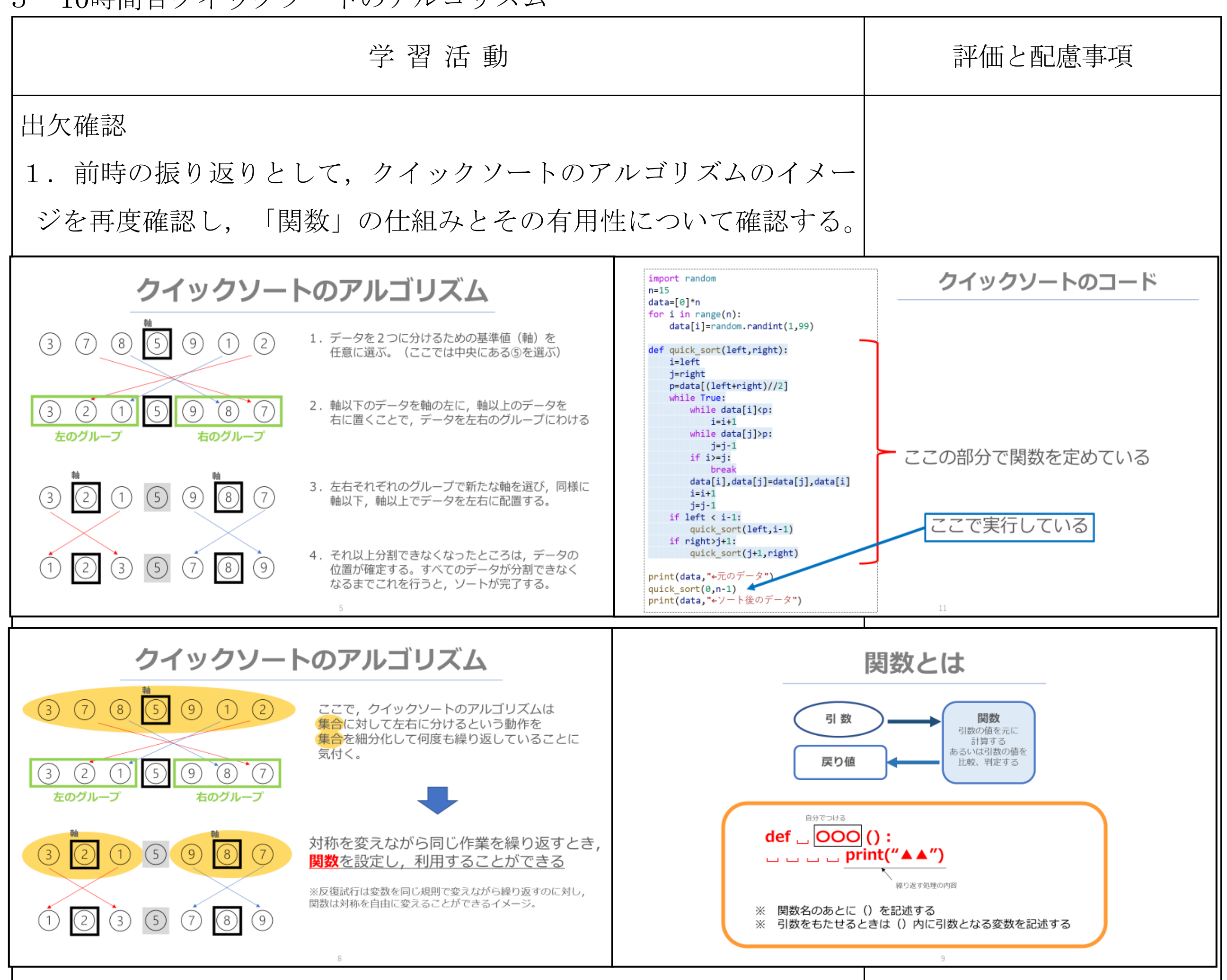
学習を進めるうえでコーディングを行うことが目的にならないよう留意し，アルゴリズムそのものについて考えさせられるよう声掛けを工夫したい。

4　単元（題材）の評価規準

時数	主な学習活動	知技	思判表	態度	評価の方法等
1 2 3	演算子の使い方，入出力，表示の仕方，データの種類と変数の使い方など， 基本的な文法を理解し，簡単なアルゴリズムを考える中で活用することができるようにする。	○			行動観察 成果物
4	① 「ある数値を入力し，その数値より小さい素数をすべて出力する」 ② 「十二面体サイコロを２つ投げ，出目が異なる場合は２数の積の1000倍，出目が同じときは1万円のお年玉がもらえるときの期待値を出力する」 の２つのコードを提出させる。①に関しては前単元（アルゴリズムの方法）で学習し，アルゴリズムはフローチャートでまとめてある状態からコーディングを行う。②は数Aの内容を踏まえ，学習したばかりの期待値の復習もかねてアルゴリズムから考える。		○	○	レポート
5	配列について理解し，基本制御構造と合わせて簡単なプログラムを作成し実行する。 また，線形探索のアルゴリズムを学び，プログラムを実行する。	○	○		行動観察 成果物
6 7 8	選択ソート，バブルソート，挿入ソートの概念について説明し，カードを使用しアルゴリズムをペアで説明し合ったり，フローチャートで表現したりする活動を通して理解を深める。 また，アルゴリズムをコーディングする際に，正確に書くことの重要性を実感させるとともに，ループの回数や範囲を考えさせたり error修正を繰り返し行うことを通して，コードやアルゴリズム自体を振り返り評価する。		○	○	行動観察 成果物

9 10	クイックソートのアルゴリズムをコードから読み取り理解する。このとき，関数の仕組みについて理解し，その有用性を実感できるようにする。 また，アルゴリズムを理解する際，手元のカードで ｉ や ｊ を動かしながら考え，ワークシートにフローチャート等で表現する。	○	○	○	行動観察 成果物 ワークシート
11	配列の個数を増やし，学習したソートのアルゴリズムを比較することで，その違いについて考える。				

5　10時間目クイックソートのアルゴリズム

<table>
<tr><th>学 習 活 動</th><th>評価と配慮事項</th></tr>
<tr><td>出欠確認
1．前時の振り返りとして，クイックソートのアルゴリズムのイメージを再度確認し，「関数」の仕組みとその有用性について確認する。</td><td></td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td>2．２人ペアまたは３人ペアになり，コードを参考にしながらカードを用いてクイックソート実行時のコンピュータの動きを再現する。
◇「手持ちの数字カードを７枚取り出し，「i」「j」を用いてコンピュータの動きを再現してみましょう」
◇「できたところは，配列そのものや配列数を変更して，再度再現してみましょう」</td><td>・様子をみて「p」のカードを配布する。
・必要なグループには，机間指導にて配列個数 n に対して「n-1」がどこを指しているか考えさせる。</td></tr>
</table>

◇「クイックソートのコードを配付するので，適切な位置で配列をprintし確かめてみてもよいでしょう」

・必要なグループには，配列が書いてあるデッキを配付し，i と data[i]の違いを意識させる。

・必要なグループには，出来ているグループからアドバイスに行かせ，なるべく生徒間のコミュニケーションを中心に理解が深まるようにする。

```
【配布コード】

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")
```

３．クイックソートについて，ワークシートにまとめる。

◇関数を含むフローチャートは初めて考えるので，例を用いて書き方を説明する。

◇昨年度のレポートを見せ，ループごとに配列を書くことで説明する書き方も説明する。

・概ね時間内にまとめきれないことが予想されるので，適当な時間で切り上げ，後日提出方法を確認する。

・ワークシートをもとに評価を行う。

４．本時のまとめと次回の展開

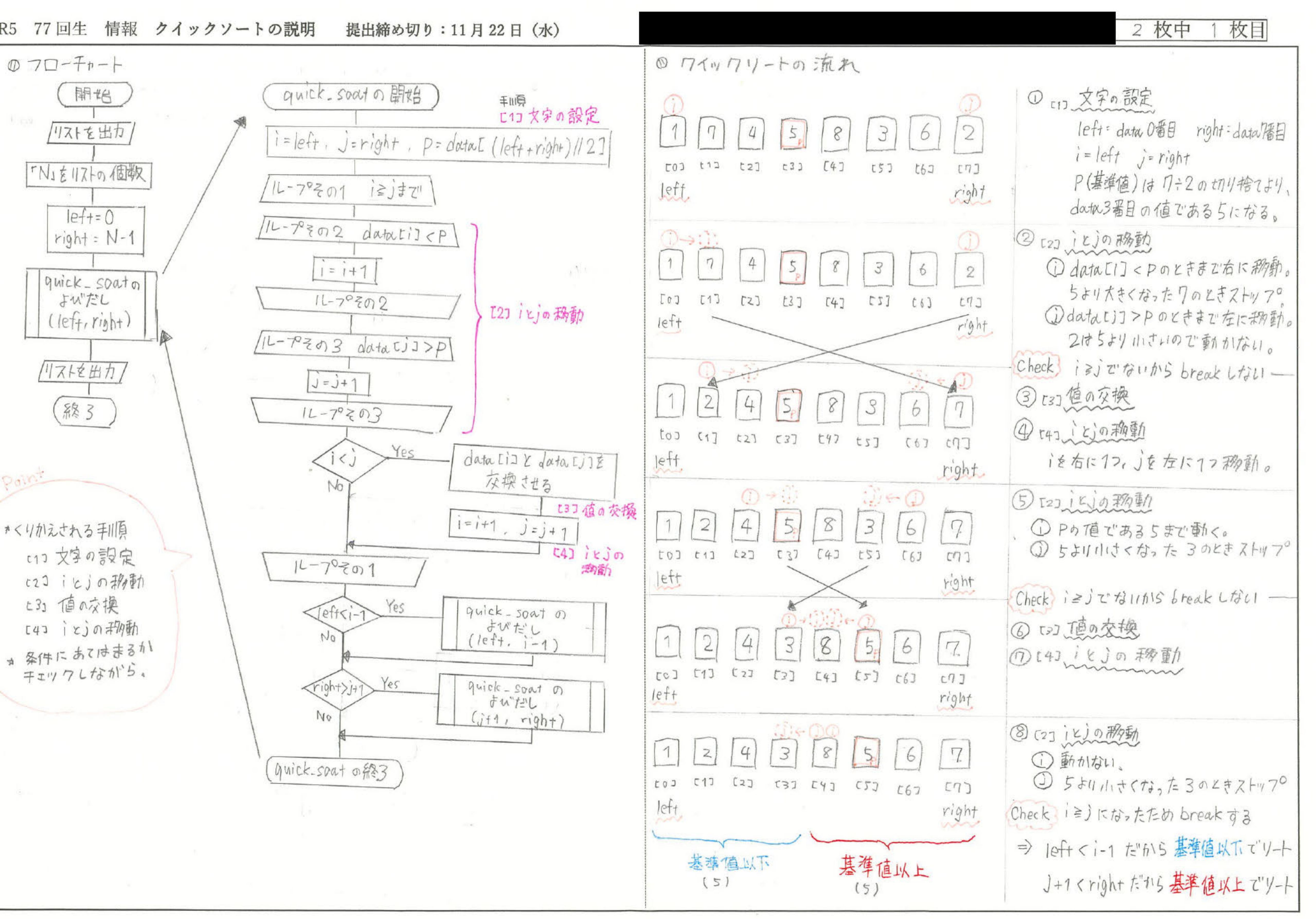
・クイックソートのように同じ作業を別の範囲で繰り返すとき，関数が有用であることを確認する。

・クイックソートは今までのソートと比べてどのくらい早いのか実験し，次回はなぜこのような差ができるのか考えることを確認する。

・選択ソートとクイックソートで同じ配列を並べ替え，どのくらい時間に差があるのか実際に実行してみる。

６　昨年度反省点

- ✓ 特に２階層目のソートを行う段階で，Leftとrightの位置がわからなくなってしまった生徒が見受けられた
  →「数字カード」「i，j，pカード」のほかに「left,rightカード」もあると良かったのではない。
- ✓ 並び替えていて，うまくいかなかったときに，元の状態に戻せなくなり，どこが間違ったかわからなくなってしまっていた
  →練習問題のような形で，特定の数字の並びを全員で共有し，途中経過も含めて答えが共有しやすいように工夫するとよいのではないか。
  →一階層目のソートが終わるタイミングはどんなときなのか整理するなど，細かくステップを踏ませるような活動を入れるとよかったのではないか。
- ✓ 時間内でなかなか理解しきれない生徒が家で試行錯誤できるようにしたい。
  →せっかく授業中にコードを配ったので，プリント関数を途中にはさむことなどを少し解説できていれば，生徒が家に帰ってから試行錯誤できたのではないか。特にiとjが今どこにあるのかは，出力させることで解決できる部分も大きかったのではないか。
  （ここまでの選択ソート，バブルソート，挿入ソートの過程で，実行途中でプリント関数を挟む作業を行ってきていた。後半にコードを使用して考える方法を伝えればよかった。）


① フローチャート
開始
リストを出力
「N」をリストの個数
left=0
right=N-1
quick_soatのよびだし (left, right)
リストを出力
終了
quick_soatの開始
手順
[1] 文字の設定
i=left, j=right, P=data[(left+right)//2]
ループその1 i≧jまで
ループその2 data[i]<P
i=i+1
ループその2
[2] iとjの移動
ループその3 data[j]>P
j=j+1
ループその3
i<j
Yes
No
data[i]とdata[j]を交換させる
[3] 値の交換
i=i+1, j=j+1
[4] iとjの移動
ループその1
left<i-1
quick_soat のよびだし (left, i-1)
right>j+1
quick_soat のよびだし (j+1, right)
quick_soatの終了
Point
*くりかえされる手順
[1] 文字の設定
[2] iとjの移動
[3] 値の交換
[4] iとjの移動
* 条件にあてはまるかチェックしながら。
② クイックソートの流れ
left
right
基準値以下 (5)
基準値以上 (5)
① [1] 文字の設定
left: data0番目 right: data7番目
i=left j=right
P(基準値)は 7÷2の切り捨てより、data3番目の値である5になる。
② [2] iとjの移動
ⓘ data[i]<Pのときまで右に移動。5より大きくなった7のときストップ。
ⓙ data[j]>Pのときまで左に移動。2は5より小さいので動かない。
Check i≧jでないからbreakしない
③ [3] 値の交換
④ [4] iとjの移動
iを右に1つ、jを左に1つ移動。
⑤ [2] iとjの移動
ⓘ Pの値である5まで動く。
ⓙ 5より小さくなった3のときストップ
Check i≧jでないからbreakしない
⑥ [3] 値の交換
⑦ [4] iとjの移動
⑧ [2] iとjの移動
ⓘ 動かない。
ⓙ 5より小さくなった3のときストップ
Check i≧jになったためbreakする
⇒ left<i-1 だから 基準値以下でソート
j+1<right だから 基準値以上でソート

基準値以下でソート

1 (i)	2p	4	3 (j)
[0] left	[1]	[2]	[3] right

⑨ [1] 文字の設定
left：data 0番目
right：data 3番目（i−1より）
i = left　j = right
P（基準値）は 3÷2の切り捨てより、data 1番目の値である2になる。

1 (i→)	2p (i j)	4	3 (←j)
[0] left	[1]	[2]	[3] right

基準値以下 / 基準値以上

⑩ [2] iとjの移動
Ⓘ Pの値である2まで動く。
Ⓙ Pの値である2まで動く。
Check i≧jになったから break
⇒ left < i−1 でないから 基準値以下のソート完了〃
j+1 < right だから 基準値以上でソート

基準値以上でソート

4p (i)	3 (j)
[2] left	[3] right

⑪ [1] 文字の設定
left：data 2番目
right：data 3番目
i = left　j = right
P（基準値）は 5÷2の切り捨てより、data 2番目の値である4になる。

⑫ [2] iとjの移動
ⓘ 動かない。
ⓙ 動かない。
Check i≧jでないから breakしない
⑬ [3] 値の交換
⑭ [4] iとjの移動
⑮ [2] ＝ ⓘⓙ動かない。
Check i≧jなので break〃

3 (i)	4p (j)
[2] left	[3] right

基準値以上でソート

8 (i)	5p	6	7 (j)
[4] left	[5]	[6]	[7] right

⑬ [1] 文字の設定
left：data 4番目（j+1より）
right：data 7番目
i = left　j = right
P（基準値）は 11÷2の切り捨てより、data 5番目の値である5になる。

⑭ [2] iとjの移動
Ⓘ 動かない。
Ⓙ Pの値である5まで動く。
Check i≧jでないから breakしない。
⑮ [3] 値の交換
[4] iとjの移動

5p (i)	8	6	7
[4] left	[5]	[6]	[7] right

⑯ [2] iとjの移動
ⓘⓙ 動かない。
Check i≧jになったから break
⇒ left < i−1 でないから 基準値以下のソート完了〃
j+1 < right だから 基準値以上でソート

基準値以下 / 基準値以上

基準値以上でソート

8 (i)	6p	7 (j)
[5] left	[6]	[7] right

⑰ [1] 文字の設定
left：data 5番目
right：data 7番目
i = left　j = right
P（基準値）は 12÷2より data 6番目の値である6になる。

⑱ [2] iとjの移動
ⓘ 動かない。
ⓙ Pの値である6まで動く。
Check i≧jでないから breakしない。
⑲ [3] 値の交換
[4] iとjの移動

6p (i)	8	7
[5] left	[6]	[7] right

⑳ [2] iとjの移動
ⓘⓙ 動かない。
Check i≧jになったから break
⇒ left < i−1 でないから 基準値以下のソート完了〃
j+1 < right だから 基準値以上でソート

基準値以下 / 基準値以上

基準値以上でソート

8p (i)	7 (j)
[6] left	[7] right

㉑ [1] 文字の設定
left：data 6番目
right：data 7番目
i = left　j = right
P（基準値）は 13÷2の切り捨てより data 6番目の値である8になる。

㉒ [2] iとjの移動
ⓘⓙ 動かない。
Check i≧jでないから break しない。
㉓ [3] 値の交換
㉔ [4] iとjの移動
㉕ [2] ＝ ⓘⓙ 動かない
Check i≧jなので break〃

7 (i)	8p (j)
[6] left	[7] right

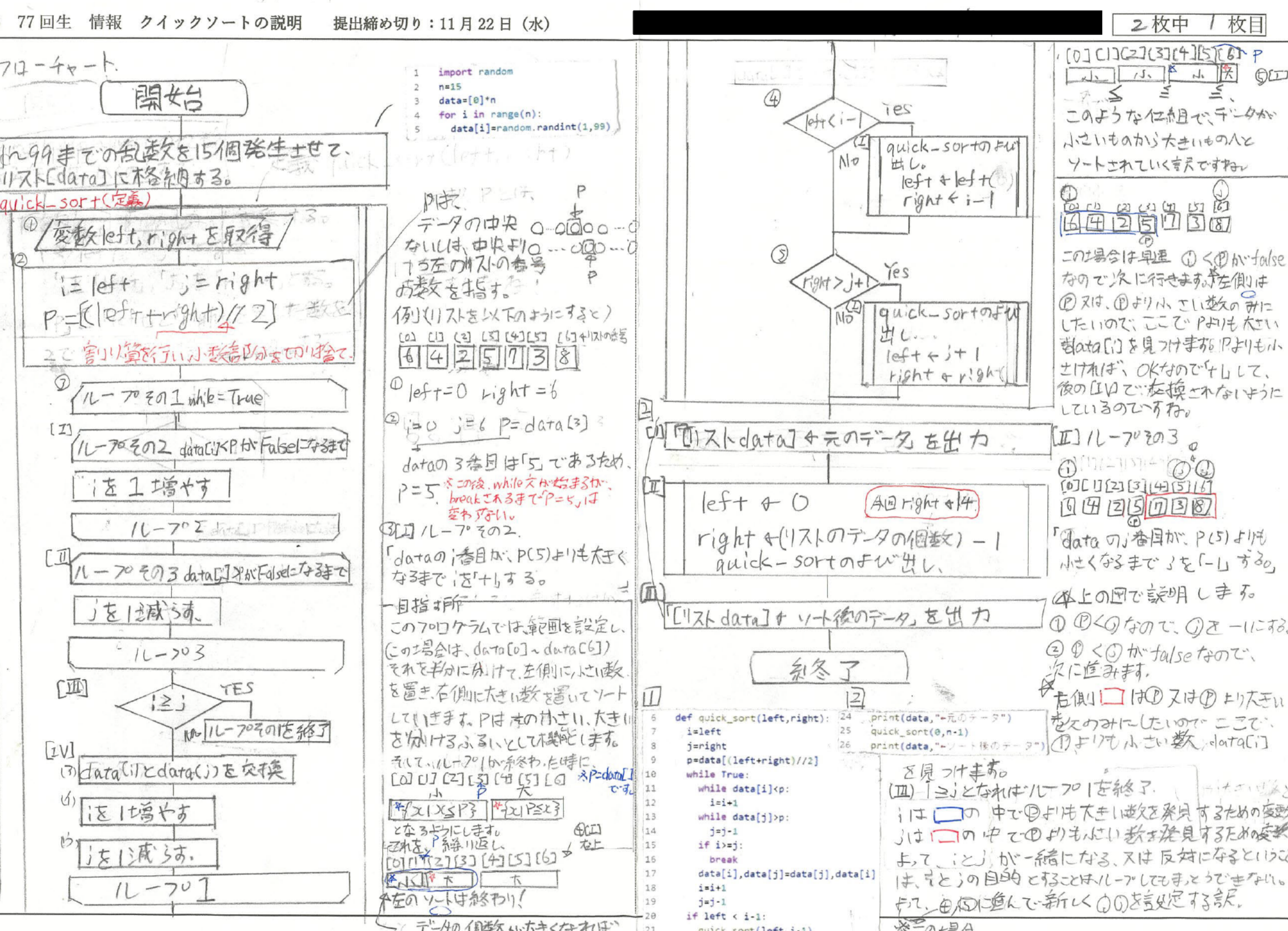
クイックソート完了

1	2	3	4	5	6	7	8

Check ⑨,⑩ / ⑪⑫ / Check ⑬〜⑯ / ⑰〜⑳ / Check ㉑〜㉓ / ①〜⑧

↑ ソート順番

Point ◎ 基準値は以上、以下どちらの場合にふくまれてもよい。



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

```python
6  def quick_sort(left,right):
7      i=left
8      j=right
9      p=data[(left+right)//2]
10     while True:
11         while data[i]<p:
12             i=i+1
13         while data[j]>p:
14             j=j-1
15         if i>=j:
16             break
17         data[i],data[j]=data[j],data[i]
18         i=i+1
19         j=j-1
20     if left < i-1:
21         quick_sort(left,i-1)
22     if right>j+1:
23         quick_sort(j+1,right)
```

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

[IV]数の交換

(ア)

[0][1][2][3][4][5][6]
6 4 2 5 7 3 8
入れ替え

これで、入れ替えの第一弾は終了。
もとの入れ替えた位置には、この場合だと それぞれ、
1番目に小さい数 2番目に大きい数
3 …… 6 8 と、この左右の3個のデータの
並び替えは完了しているため、①を1個右にずらし、②を1個左に動かします。

(イ)(ウ)

[0][1][2][3][4][5][6]
3 4 2 5 7 6 8

[Final]

以上[I]〜[IV]を、[III]が「True」となるまで繰り返します。

この場合は [/] [ ] と既に一順で分けれたので
5より小さい集合 5より大きい集合 これで終了

もう一周して、[III]が「True」となりbreakになります。[0][1][2][3][4][5][6] 3 4 2 5 7 6 8

④,⑤

詳細は1枚目の「目指す所」に書いてありますが、
④で☆の範囲内で、quick_sortを再度行う。Pの位置は☆と☆を隔てるものなので、rightはi=1とする。
5より小さい数の集合 P 5より大きい数の集合

※注意するべきは新しく始まったquick_sortのleft, right, j, iは④を実行していたdef内のものとは別であるということ。
なので、left', right', j', i'などとおいて、関数の中の中を表す時は、元のleftなどと混合しないよう注意する。

⑤も④と同様。
そして、すべてのquick_sortで、④,⑤の条件式がfalseとなれば、[2]IIが終わり、[2]IIIで結果を出力して、プログラムは終了する。

まめっと豆知識！

このプログラムではquick_sortの中でquick_sortを行い……と

□□□□□□□□□□□□ ← こんな風に問題を分割して、ソートしていますが、しかし、データの数が多ければquick_sortの中でquick_sortを行い、またその中でquick_sortを行いと、

何回も自己の中でプログラムを繰り返すので、ここにある問題が発生します。それは、Pythonの再帰の深さには制限があるということです。
（再帰…ある関数が自分自身を呼び出すプログラミングの手法）

そのため、あまりにも大きな数のデータを用いると、最大再帰深度（Maximum Recursion Depth）に達しエラーを吐きます。下図参照。通常、これは1000程度に設定されますが、

```
1  import sys
2  sys.setrecursionlimit(10)
3  data=list(range(1,1000001))
4  n=1000000
5  def quick_sort(left,right):
6    i=left
7    j=right
8    p=data[(left+right)//2]
9    while True:
10     while data[i]<p:
11       i=i+1
12     while data[j]>p:
13       j=j-1
14     if i>=j:
15       break
16     data[i],data[j]=data[j],data[i]
17     i=i+1
18     j=j-1
19   if left < i-1:
20     quick_sort(left,i-1)
21   if right>j+1:
22     quick_sort(j+1,right)
23 print(data,"←元のデータ")
24 quick_sort(0,n-1)
25 print(data,"←ソート後のデータ")
```

←今回は、PCへの負荷を考え、最大深度を10とするコードを上2行に追加し、エラーを誘発する形にはなりますが、実際にエラーを出せるか検証します。

```
99934, 999935, 999936, 999937, 999938, 999939, 99
47, 999948, 999949, 999950, 999951, 999952, 99995
 999961, 999962, 999963, 999964, 999965, 999966,
9974, 999975, 999976, 999977, 999978, 999979, 999
7, 999988, 999989, 999990, 999991, 999992, 999993
←元のデータ
Traceback (most recent call last):
  File "c:\Users\kozak\OneDrive\ドキュメント\Pyth
    quick_sort(0,n-1)
  File "c:\Users\kozak\OneDrive\ドキュメント\Pyth
    quick_sort(left,i-1)
  File "c:\Users\kozak\OneDrive\ドキュメント\Pyth
    quick_sort(left,i-1)
  File "c:\Users\kozak\OneDrive\ドキュメント\Pyth
    quick_sort(left,i-1)
  [Previous line repeated 6 more times]
RecursionError: maximum recursion depth exceeded
```

←実行結果

はい。↑エラー出ました。
つまり、普通の環境でも、これより大きな（いっぱい）個数のリストを入力すれば、エラーは出るはずです理論上。という小話でした。

```
import random
n = 15
data = [0]*n
for i in range(n):
    data[i] = random.randint(1,99)
```
} 並び替える15個の数をランダムに決める

```
def quick_sort(left,right):
    i = left
    j = right
    p = data[(left+right)//2]
```
} ①

```
    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
```
} ②

```
    if left < i-1:
        quick_sort(left,i-1)
    if right > j+1:
        quick_sort(j+1,right)
```
} ③

```
print(data,"← 元のデータ")
quick_sort(0,n-1)
print(data,"← ソート後のデータ")
```
} ④

① quick_sort(left,right):

すべての数が乱雑に並べられている中で、一番左の値(データそのものではなくその数を1ブロックとしてみた場合)から0～14まで番号をつける。その中で0をleft、14をrightとする。

i = left
j = right

上で決めたleftの値をi、rightの値をjとする。

p = data[(left+right)//2]

leftの番号とrightの番号をたして2で割って、あまりの数がある場合は切り捨てて、出た数をpとして、その数の番号のブロックをpと定める。

- その数のブロックが動く場合はpもともに動く。
- leftとrightの番号をたして2で割ると、データの個数に合った真ん中の位置が出る

i														j
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
○	○	○	○	○	○	○	ⓟ	○	○	○	○	○	○	○

② while True:

無限ループを作っている

```
while data[i]<p:
    i = i+1
```
iの値がpより小さい場合はもともとあったiのブロックから1つ後ろの数(1から2など)へ進む。次の数もpより小さければこれを繰り返す

```
while data[j]>p:
    j = j-1
```
jの値がpより大きい場合はもともとあったjのブロックから1つ前の数字(15から14など)に進む。次の数字もpより小さければこれを繰り返す

```
if i>=j:
    break
```
iがjより大きくなったときか同じになったときにループをbreakする(iの値でなく、ブロックの番号)

```
data[i],data[j] = data[j],data[i]
```
i>p、j<pになったとき iの値とjの値を入れ換える

```
i = i+1
j = j-1
```
iを1つ後ろの数(1から2など)へ移動させる
jを1つ前の数(15から14など)へ移動させる
⇐ データiとjを入れ換える動作をした後

③ if left < j-1 :
    quick_sort (left, i-1)　pよりも左側の部分配列を再帰的にソートする
  if right > j+1 :
    quick_sort (j+1, right)　pよりも左側の部分配列を再帰的にソートする

④ print (date, "← 元のデータ")
プログラムにおいて数を並び替える前の数字の並び方をプリントする
quick_sort(0, n-1)
print (data, "← ソート後のデータ)
0からn-1までのデータを小さい数から順に並び替えたものをプリントする