# アルゴリズムの比較から
# 効率的なアルゴリズムの理解の仕方

大阪電気通信大学
教授・副学長　兼宗 進

# 「情報Iプログラミング」の解説動画

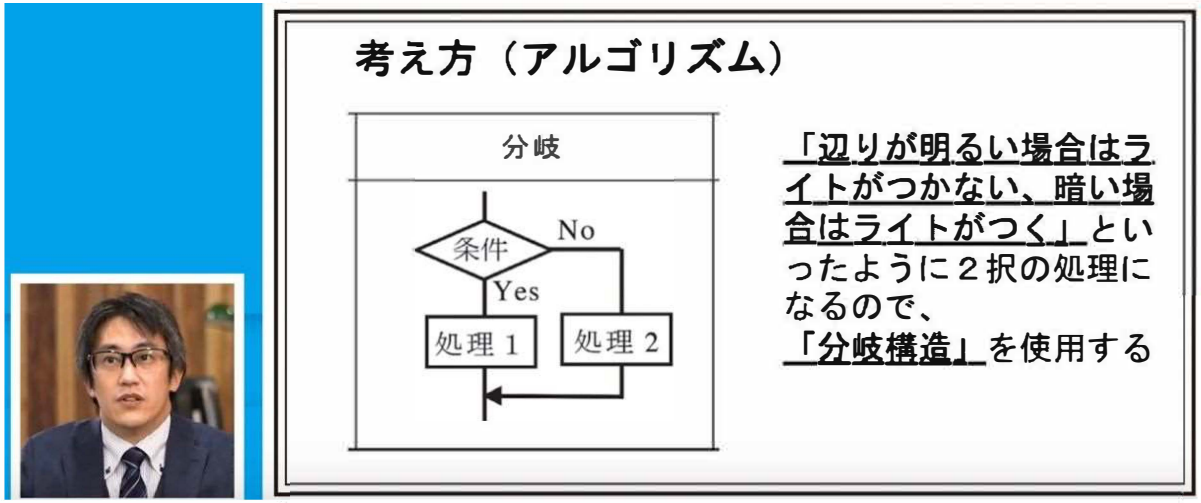
- 高校の先生方（三井先生/鎌田先生）が説明
  - [1]センサーライトを作ろう!（モデル化と外部機器を活用したシミュレーション ）
  - [2]100連ガチャをプログラムして作ろう!（アルゴリズムの基本とプログラミング ）
  - [3]公平な方法で発表順番を決めよう!（データ構造　外部プログラムの連携 ）
  - [4]天気予報表示マシーンを作ろう!（情報通信ネットワークを活用したプログラミング ）
- 今日の内容
  - アルゴリズムの用途と性能の重要性
  - 教科書で扱われているアルゴリズム
  - 「小さなプログラムで大きな仕事」を体験しよう

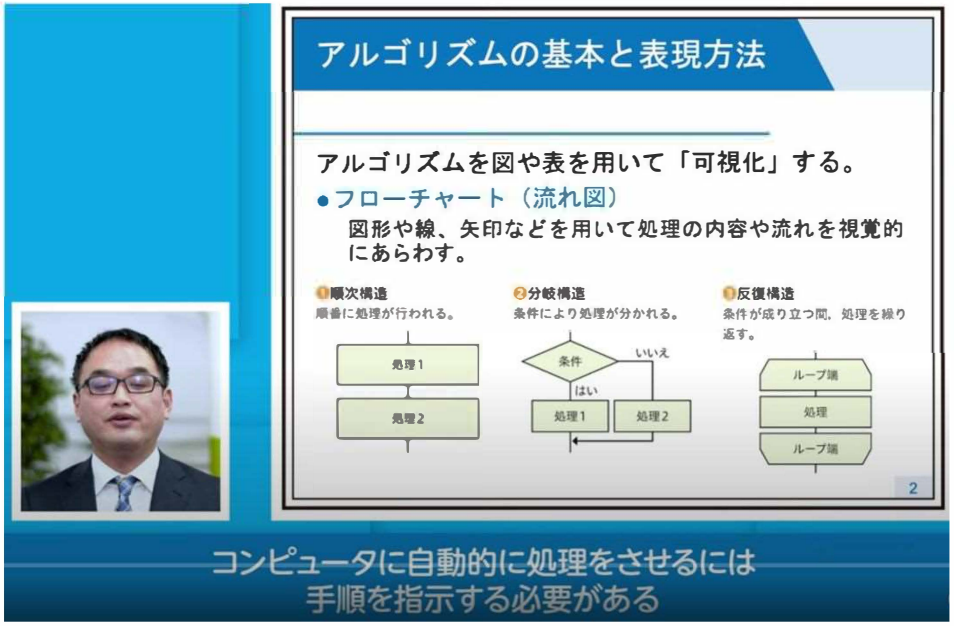
# ソフトウェア設計としてのアルゴリズム

- 研修動画では「プログラムを作る前に考える手順の設計」の意味で使用
- ソフトウェア開発の詳細設計に相当

ソフトウェア開発：　概要設計　→　**詳細設計**　→　**実装**　→　動作確認

（機能・デザイン）（アルゴリズム）（プログラム）　（テスト）



# アルゴリズムの用途と性能の重要性

# 問題解法としてのアルゴリズム

- **問題の解を求める定石や公式のような手順**
- さまざまなアルゴリズムが毎日の生活で利用されている
  - パズルの解法（ルービックキューブ、将棋、…）
  - 最短経路（鉄道の乗換検索、自動車のナビゲーション）
  - Web検索、書籍検索
  - 検索結果のランキング表示、おすすめ商品のレコメンデーション
  - データ暗号化、データ圧縮
  - 画像認識、音声認識
- 「情報I」の教科書の例（太字は複数教科書が使用）
  - 数値計算（平方根を求める、フィボナッチ数列、**素数を求める**）
  - **データの探索、データの整列、最大値/最小値を求める**

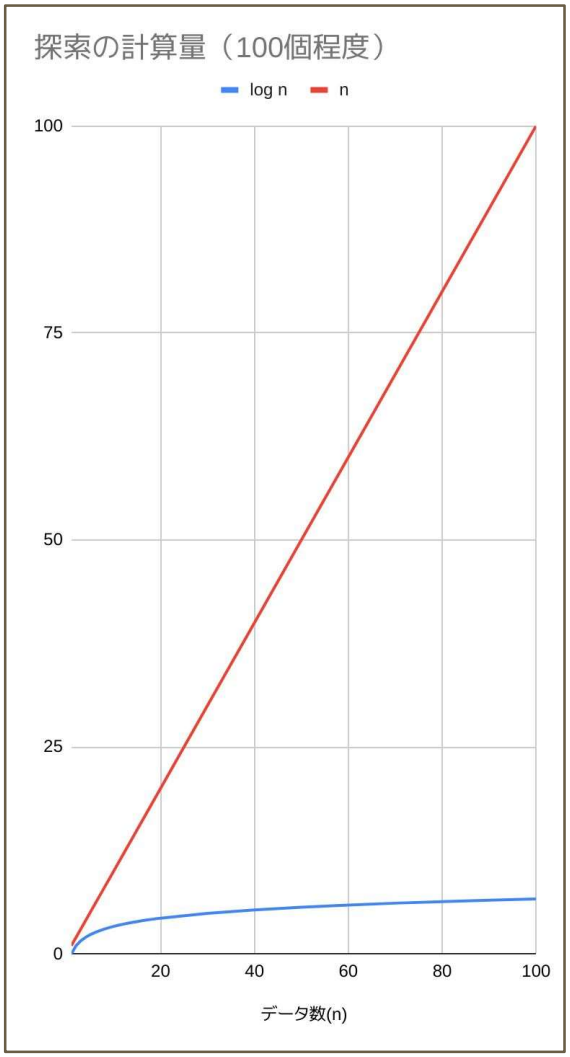
# 大量のデータを扱うアルゴリズム（探索）

- データが増えると計算量（計算時間）が爆発的に増える
- 探索の例
  - 線形探索はデータ数と比例して時間がかかる（赤線）
  - 二分探索はデータ数が増えても時間がかからない（青線）
    - ただし、データはソート済である必要がある
  - データが増えると、さらに差が開く

```
# 線形探索
a=[3,7,5,4,1,6,2]
n=4
p=-1
for i in range(5):
  if a[i]==n:
    p=i
    break
print(p)
```

```
# 二分探索
a=[1,2,3,4,5]
n=5
p=-1
l=0
r=len(a)
while l<r:
    c=int((l+r)/2)
    if a[c]==n:
        p=c
        break
    if a[c]<n:
        l=c+1
    else:
        r=c
print(p)
```

プログラムは複雑だが、
二分探索は使う価値あり？

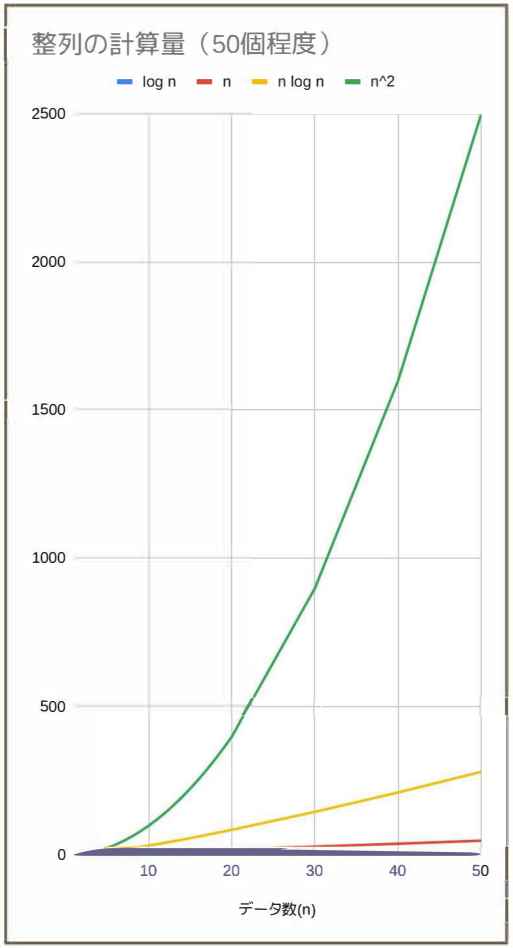


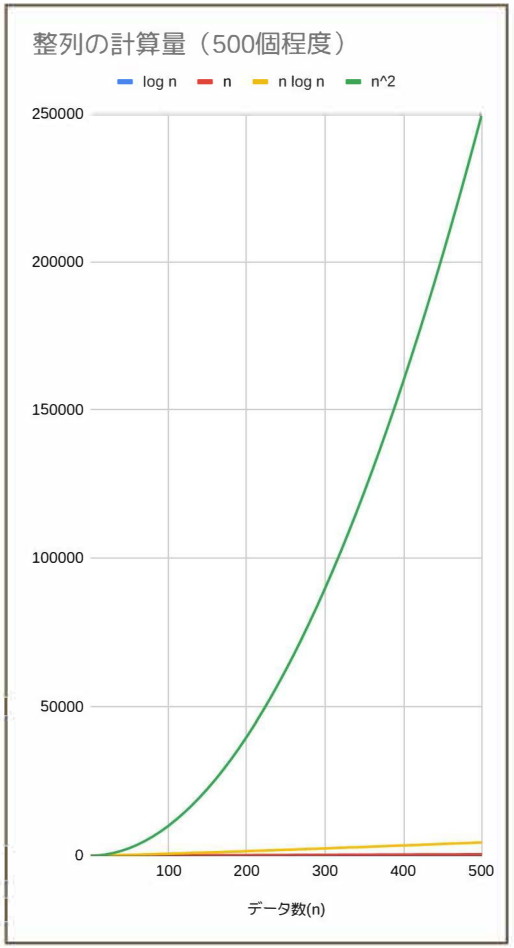
# 大量のデータを扱うアルゴリズム（整列）

- データが増えると計算量（計算時間）が爆発的に増える
- 探索は比例なのでまだマシ

- 整列は$n^2$になると実用的でない
  - $n^2$（緑線）バブル/選択/挿入など
  - n log n（黄線）クリック/マージなど
- データによって計算量が変わることも
  - ほぼソート済は速い：バブル/挿入
  - ほぼソート済は遅い：クイック

- 実際の式は「$an^2+bn+c$」のようになるが、緑線の増加は$n^2$が大きく影響するため、a,b,c,nは無視して$O(n^2)$のように考える



# 教科書で扱われているアルゴリズム

# (1)最大値/最小値を求める

- 最大値を求めるアルゴリズム
  - 変数maxを作り、リストa[0]の値を入れる
  - (1)リストaの要素について、maxと比較する
  - (2)要素がmaxより大きければ、その値をmaxに代入する
  - (1)(2)を、リストaの各要素について繰り返す

```
a=[3,2,5,4,1]
max=a[0]
for i in range(1,5):
    if a[i]>max:
        max=a[i]
print(max)
```

```
5
```

- 考え方はシンプルで、プログラムも簡潔
- 自分で考えさせる授業も可能
- 最大値maxの初期値に0を入れて「max=0」のプログラムで説明した後、データに負の数が含まれる可能性を伝えて改良させることも有効
- 最大値を説明した後で、最小値を求めるように改良させることも有効

# (2)探索(サーチ):線形探索

- アルゴリズム
  - 変数nに検索する値を入れる。変数pに-1を入れる。
  - (1)リストaの要素について、nと比較する
  - (2)要素がnと等しければ、その添字をpに代入する(そして探索処理を中断する)
  - (1)(2)を、リストaの各要素について繰り返す

- 性能(データ数7個の場合)
  - 3は1回で見つかる(最小)
  - 2は7回で見つかる(最大)
  - 平均すると3.5回(平均)

```
a=[3,7,5,4,1,6,2]
n=4
p=-1
for i in range(5):
  if a[i]==n:
    p=i
    # break
print(p)
```

```
3
```

# 二分探索

- データは値の順に並んでいる
- アルゴリズム
  - 変数nに検索する値を入れる。変数pに-1を入れる。
  - 変数lに0、変数rにリストaの要素数を入れる
  - (1)変数cに、lとrの真ん中の添字を入れる
  - (2)a[c]がnと等しければ、その添字をpに代入する（そして探索処理を中断する）
  - (3)「a[c]<n」なら、c+1をlに代入する
  - (4)そうでなければ、cをrに代入する
  - (1)から(4)を、「l>r」の間、繰り返す

アルゴリズム、プログラムは、両方とも難しく授業で扱いにくい？
→　違う教え方を工夫したい

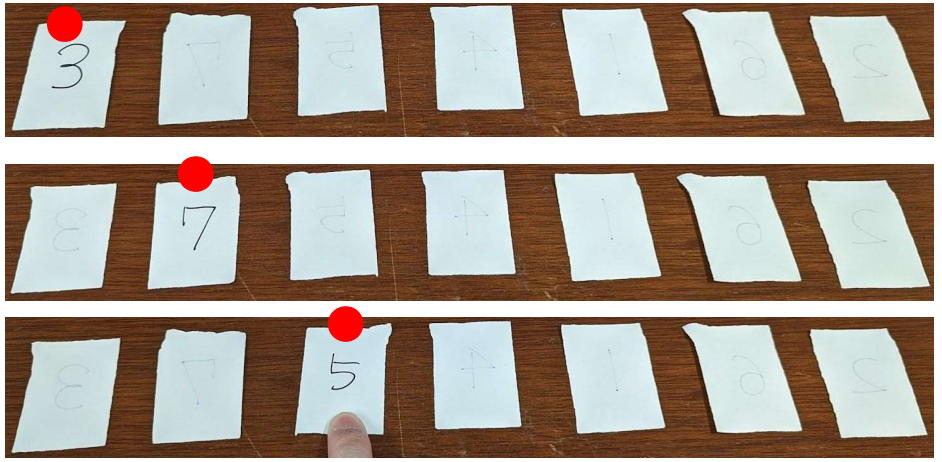
```
a=[1,2,3,4,5]
n=5
p=-1
l=0
r=len(a)
while l<r:
    c=int((l+r)/2)
    if a[c]==n:
        p=c
        break
    if a[c]<n:
        l=c+1
    else:
        r=c
print(p)
```

```
4
```

# カードで線形探索

- カードを使うとわかりやすく実習できる
  - おもてにすると見えてしまうので裏返す


- 7枚のカードから5を探す場合
  - 左端のカードを見ると3なので違う
  - となりのカードを見ると7なので違う
  - となりのカードを見ると5なので見つかった！



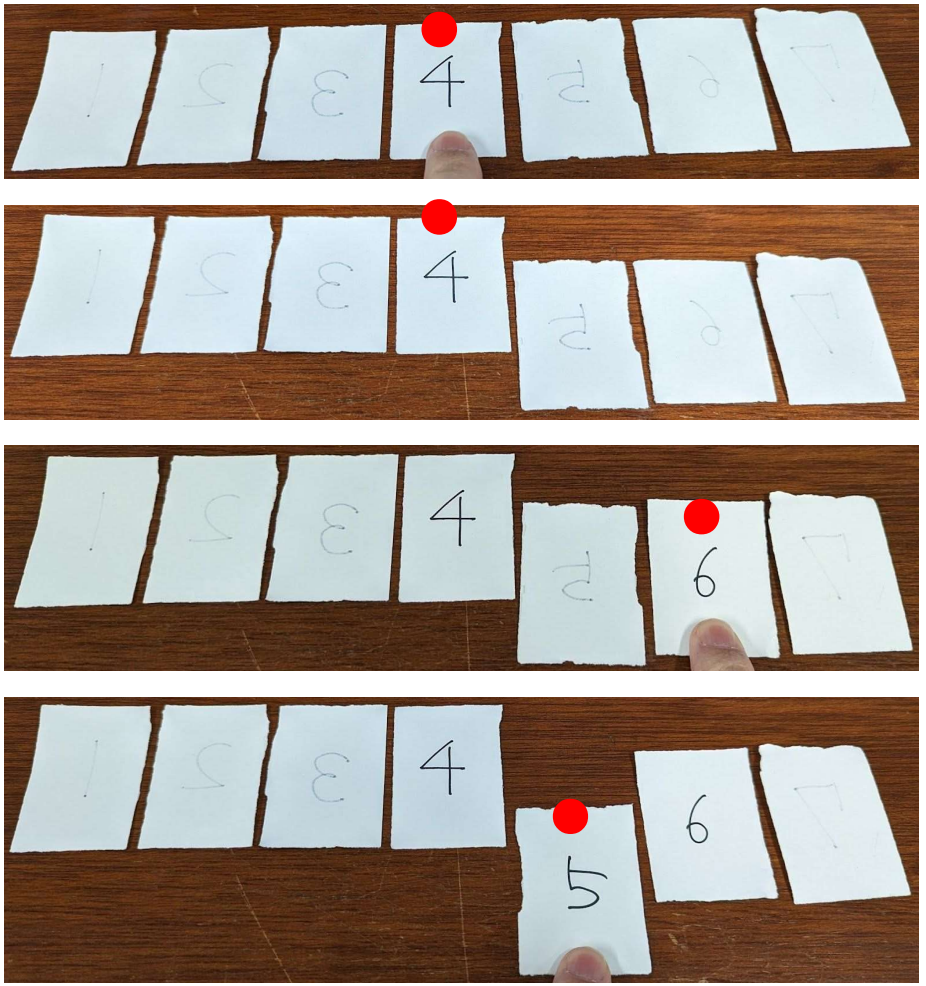

- 性能を考えてみよう
  - 5は3回目で見つかったが、2は7回見る
    - 最小： 4は**1回**で見つかる
    - 最大： 2は**7回**で見つかる
    - 平均： (1+2+3+4+5+6+7)/7= **4回**

# カードで二分探索

- カードを使うとわかりやすく実習できる！

- 7枚のカードから5を探す場合
  - 真ん中のカードを見ると4なので5より小さい
    → 右側の3枚のカードを調べる
  - 真ん中のカードを見ると6なので5より大きい
    → 左側の1枚のカードを調べる
  - 真ん中のカードを見ると5なので見つかった！

- 性能を考えてみよう
  - 5は3回目で見つかった
    - 最小： 4は**1回**で見つかる
    - 最大： 1と3と5と7は**3回**で見つかる
    - 平均： (3+2+3+1+3+2+3)/7= **2.43回**

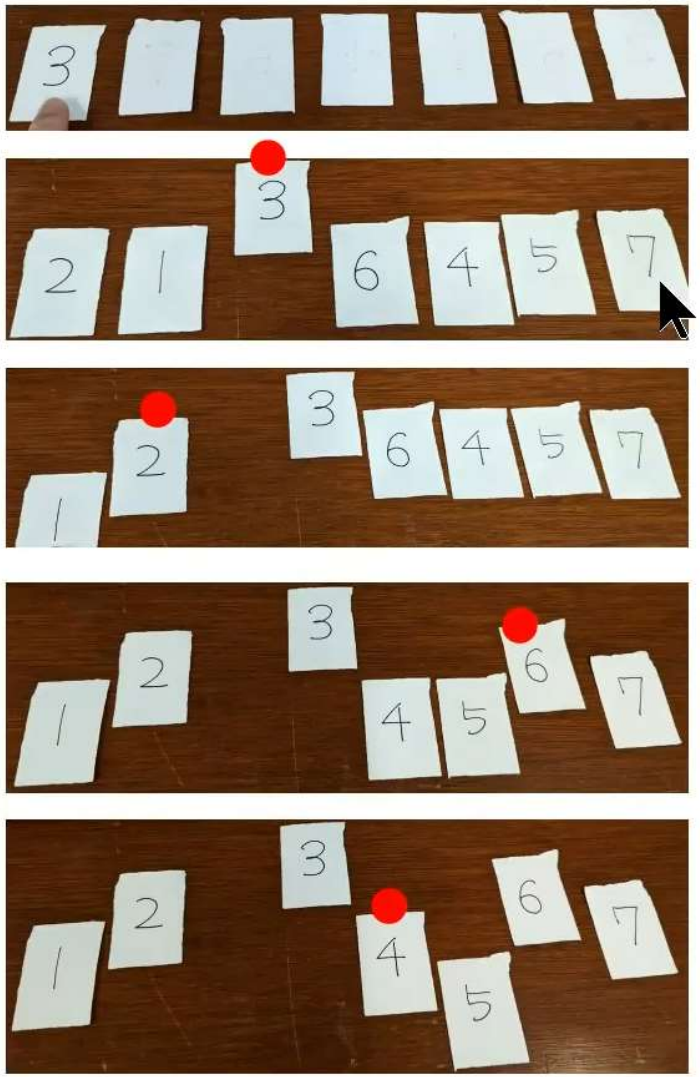


# カードでクイックソート

- カードを使うとわかりやすく実習できる！

- 7枚のカードを並べ替える場合
  - 適当な1枚（今回は左端）を選ぶ
  - その1枚と大小を比較して残りを左右に分ける
  - 左右のグループに対して、これを何度も繰り返す

- 性能を考えてみよう
  - 全部で何回、基準のカードと比較した？

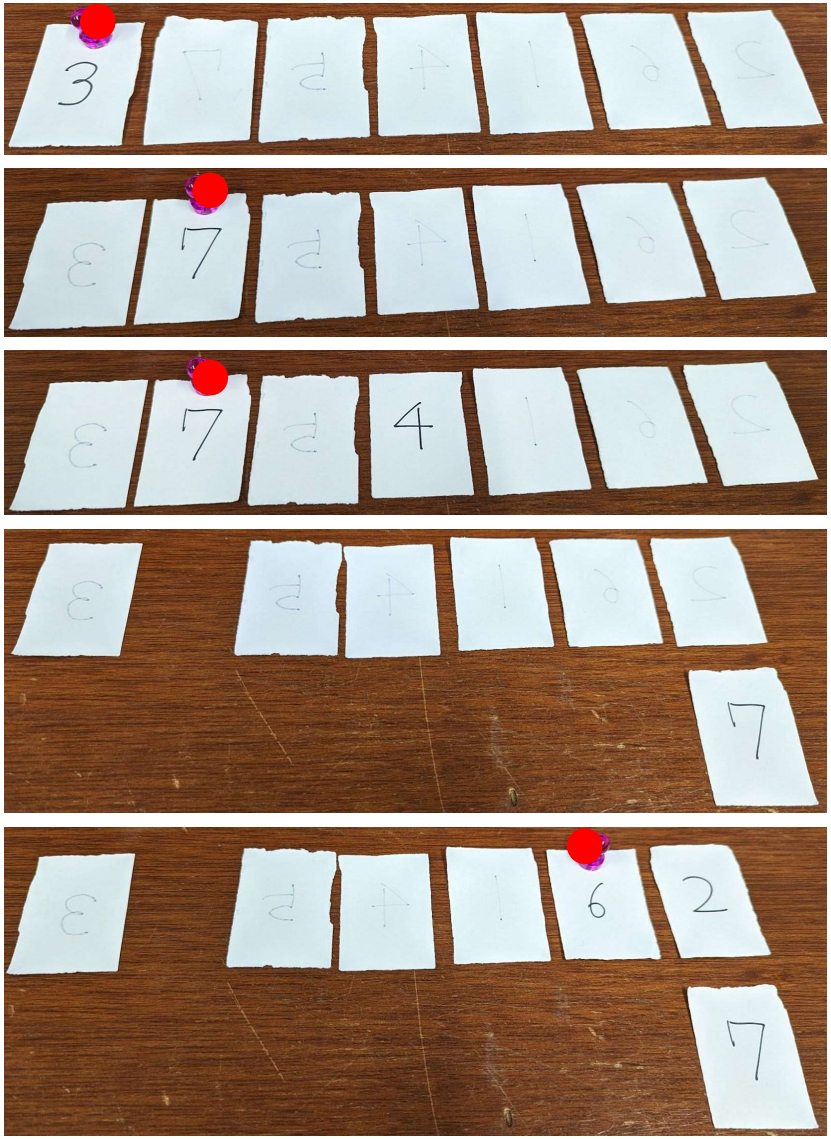


# カードで選択ソート

- カードを使うとわかりやすく実習できる！

- 7枚のカードを並べ替える場合
  - 左端から順に見て、最大値を表にして覚えておく
  - 右端まで見たら、最大値を外に出す
  - これを何度も繰り返す

- 性能を考えてみよう
  - 全部で何回、カードの値を見た？

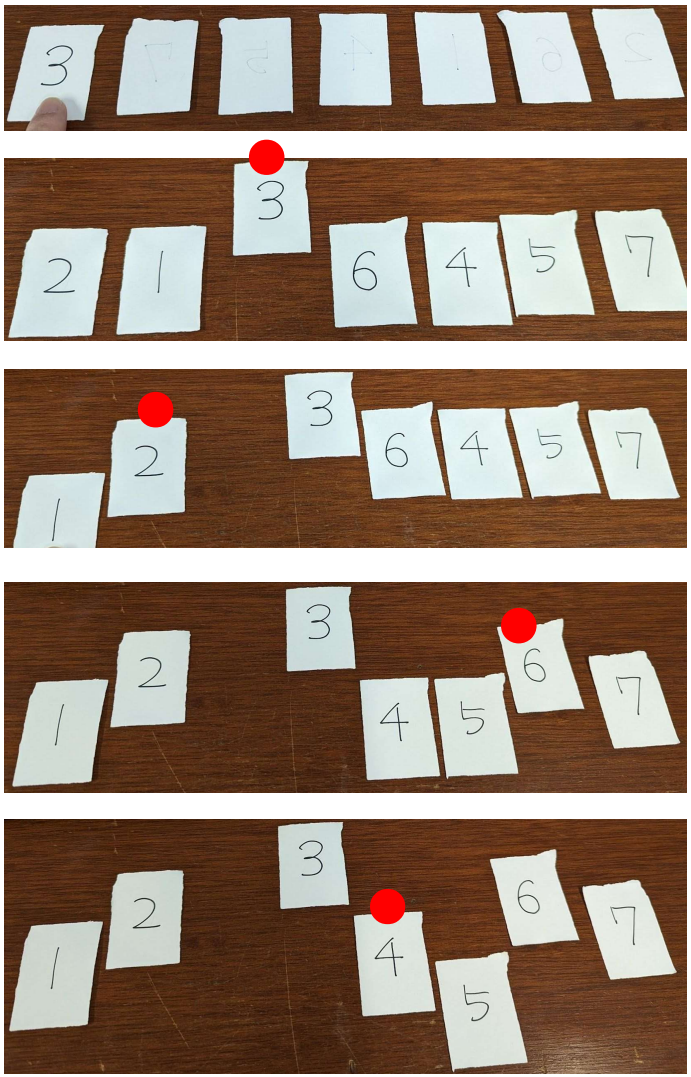


# カードでクイックソート

- カードを使うとわかりやすく実習できる！

- 7枚のカードを並べ替える場合
  - 適当な1枚（今回は左端）を選ぶ
  - その1枚と大小を比較して残りを左右に分ける
  - 左右のグループに対して、これを何度も繰り返す

- 性能を考えてみよう
  - 全部で何回、基準のカードと比較した？

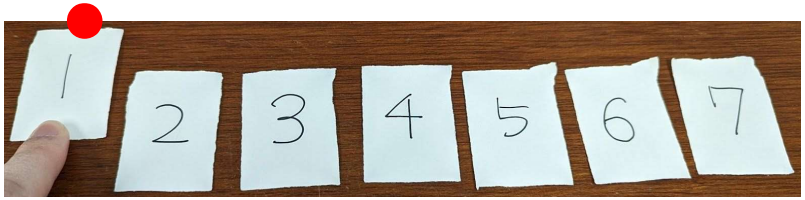


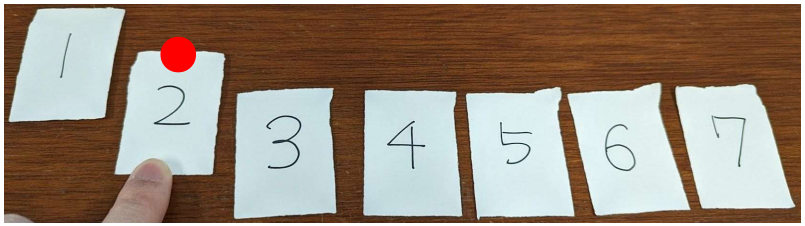
# 残念なクイックソート

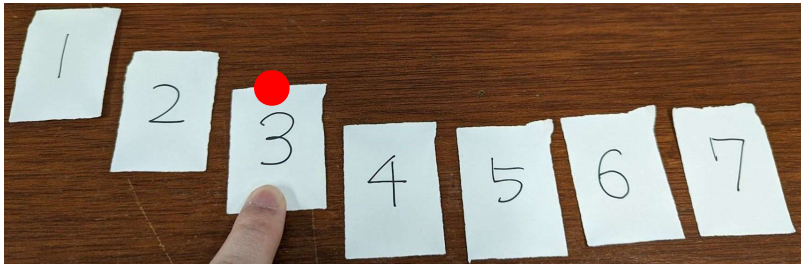
- ソート済みデータをクイックソートしてみた

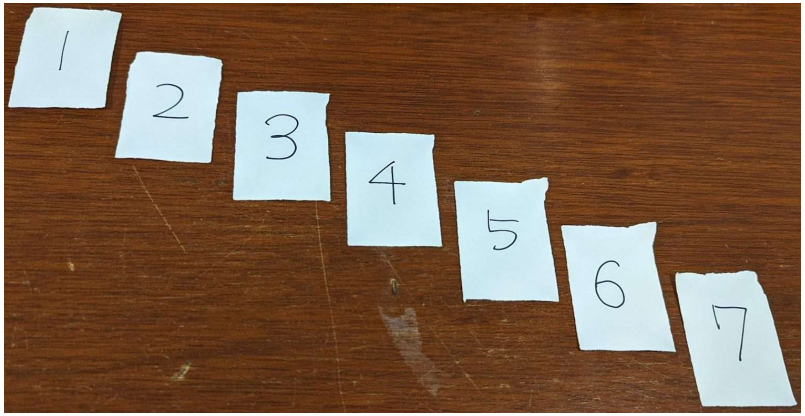
- その結果
  - 適当な1枚（今回は左端）を選んだ
  - 左右に分けようとしたが全部右に行った
  - 何度も繰り返した。結果は木でなく棒になった…

- 性能を考えてみよう
  - 全部で何回、基準のカードと比較した？



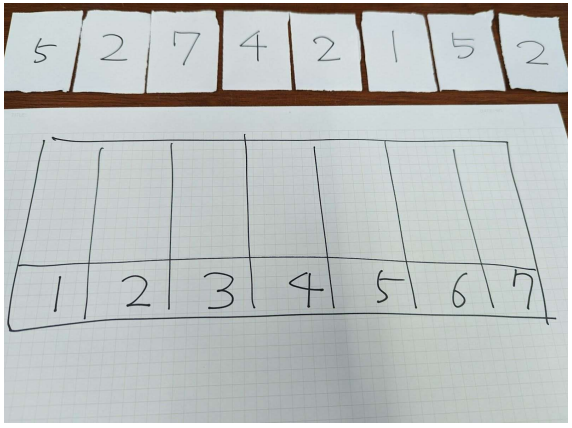



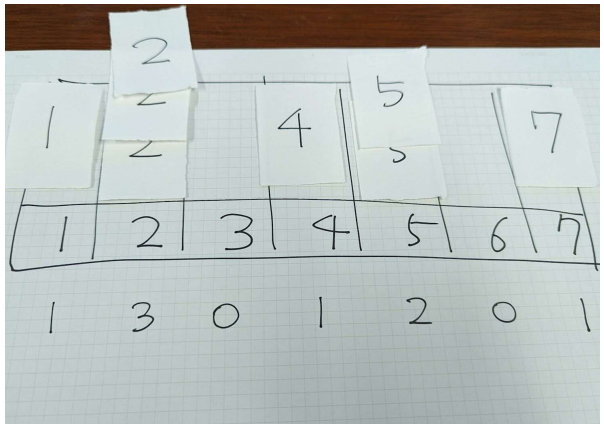
# バケットソート

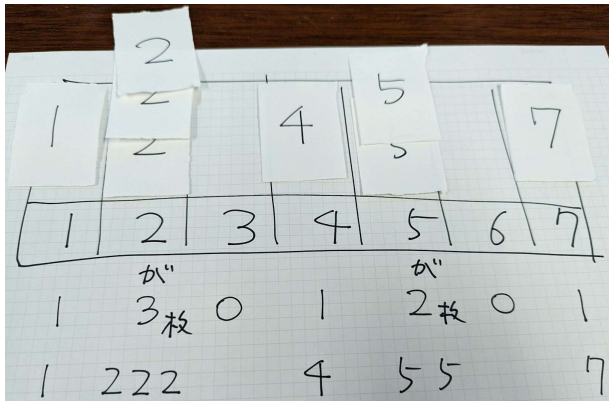
- あまり知られていないアルゴリズムをひとつ紹介
- 手順
  - 値ごとの箱を作り、そこにカードを入れる
  - 箱ごとの枚数を数える
  - 枚数分だけカードの数を書き出す
- 特徴
  - 理解しやすい。超高速（O(n)）
  - データの制約が厳しい。整数値のみ、値の範囲限定

```
max=8
data=[5, 2, 7, 4, 2, 1, 5, 2]
out=[0, 0, 0, 0, 0, 0, 0, 0]
for n in data:
    out[n]=out[n]+1
for i in range(max):
    for n in range(out[i]):
        print(i, end=" ")
```

```
1 2 2 2 4 5 5 7
```



# (4)素数の計算

- アルゴリズムを改良しながら比較できる題材
  - (1)「その数と1以外で割り切れない」という定義のとおりに、**2からその数n-1で割ってみる**。ひとつでも割り切れたら素数でないと判定
  - (2)4,6,8,10など2より大きい偶数は素数でないことに気づき、**2の倍数を除外**して調べる工夫を入れる
  - (3)3の倍数も除外、5の倍数も除外、と考えいくうちに、9の倍数の除外は意味がないことに気づき、**それまで求めた素数の倍数を除外**する工夫を入れる
  - (4)「49であれば7まで調べれば十分」ということに気づき、**その数の平方根まで調べる**工夫を入れる

- 数について考えられる生徒は、このステップを楽しんで進められる
  - そうでない生徒にはヒントを出して作業させる
- 改良前のプログラムを残しておき、(1)から(4)のプログラムで、「100までの計算」「500までの計算」のように剰余の計算回数を比較する

# 「小さなプログラムで大きな仕事」を体験しよう

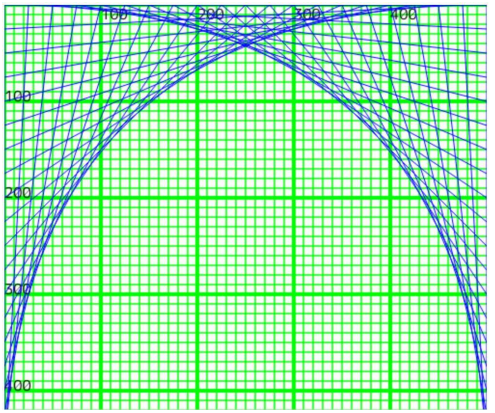
# 学習環境の例（Bit Arrowなど）

- いくつか授業用ツールを開発して公開しています。ご利用ください
- Bit Arrow：プログラミング学習ツール
  - ブラウザでプログラムを記述して実行
  - 3大学で共同開発（大阪電気通信大学、東京農工大学、明星大学）
  - 言語は、Python、DNCL、C、ドリトル、Processing、教育用JavaScript
  - 生徒へのプログラム配布、生徒プログラムのダウンロードなど
  - 無料で教員登録可能（「授業利用に向けた準備」から「教員登録フォーム」）
    - https://bitarrow.eplang.jp/
- サクセス：データベース学習ツール
  - 大阪電気通信大学、大阪大学で共同開発
    - https://saccess.eplang.jp
- Connect DB：データ活用ツール
  - 大阪電気通信大学高等学校と共同開発
    - https://cdb.eplang.jp

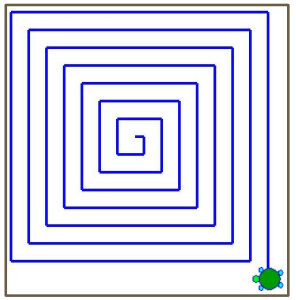
# 小さなプログラムで大きな仕事

- アルゴリズムは数学の美しい公式のようなイメージ
  - 先人が改良を続けて残してくれた宝物
- 自分で自由に書くプログラムも、冗長で長いプログラムよりは、簡潔なほうが読みやすく理解しやすい（よいプログラム）
- 数や文字だけでなく、グラフィックスを描いたほうが実感しやすい。

  Bit ArrowのC言語とドリトル言語で例を紹介
  - C言語：　図形や線で描くグラフィックス
  - ドリトル：　3,4行のプログラムで意味のある仕事




```
かめた＝タートル！作る（青）線の色。
「｜ｎ｜
　かめた！（ｎ＊10）　歩く 90　右回り。
」！30回　繰り返す。
```

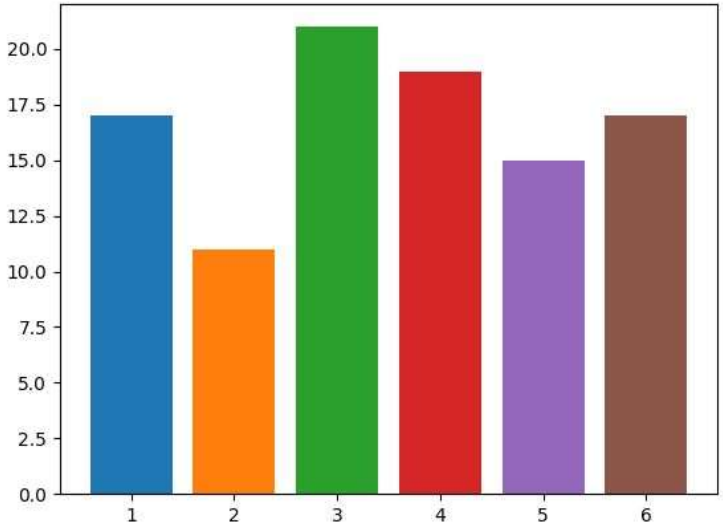


# グラフによる可視化

- 棒グラフで表示する

```
import random
import matplotlib.pyplot as plt
deme=[]
for i in range(100):
    deme.append(random.randint(1,6))
for i in range(1,7):
    plt.bar(i, deme.count(i))
plt.show()
```

→



# 各種の画像処理

- Bit Arrowの素材管理で画像ファイルを登録しておく

- 画像を表示

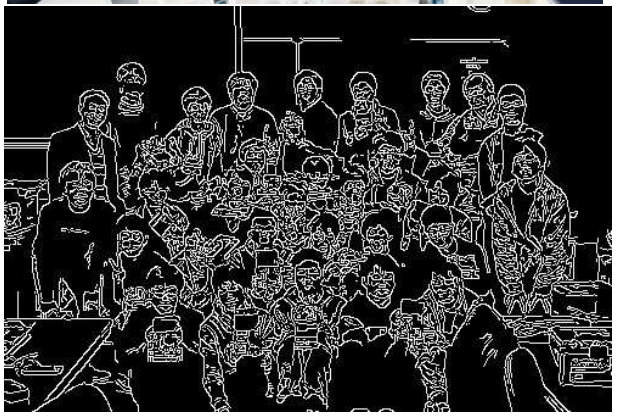
```
import cv2
img = cv2.imread("class/7-IMG_6564.jpg")
cv2.imshow("", img)
```

- 色を反転

```
rev = cv2.bitwise_not(img)
```

- 輪郭抽出

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
canny = cv2.Canny(gray, 200, 100)
```



# オープンデータの利用例

- 自治体（寝屋川市）の例
  - 利用条件は「CC BY」（原作者名を表記）

**寝屋川市オープンデータについて**

寝屋川市では、情報の利活用を推進し、地域の課題解決を図り、社会経済の発展に寄与するよう、市の保有する情報のオープンデータ化に取り組んでいます。

寝屋川市オープンデータサイトのデータの利用に関しては、CC（クリエイティブ・コモンズ）ライセンスによるCC－BYにより提供します。

| データ名 | カテゴリ | 更新日 | データ形式 |
|---|---|---|---|
| AED設置個所一覧 | 安全・安心 | 2021年12月24日 | csv |
| 指定緊急避難場所一覧 | 安全・安心 | 2021年07月01日 | csv |
| 消防水利施設一覧 | 安全・安心 | 2021年07月01日 | csv |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | NO | 名称 | 名称_カナ | 住所 | 方書 | 緯度 | 経度 | 標高 |
| 2 | 1 | 東小学校 | ヒガシショ | 大阪府寝屋川市太秦 | | 34.76139 | 135.6338 | |
| 3 | 2 | 第一中学校 | ダイイチチ | 大阪府寝屋川市高宮 | | 34.76233 | 135.6317 | |
| 4 | 3 | 市民会館 | シミンカイ | 大阪府寝屋川市秦町 | | 34.76288 | 135.6314 | |
| 5 | 4 | 東コミュニ | ヒガシコミ | 大阪府寝屋川市高宮 | | 34.76135 | 135.6313 | |
| 6 | 5 | 中央小学校 | チュウオウ | 大阪府寝屋川市初町 | | 34.76379 | 135.6257 | |
| 7 | 6 | 寝屋川高等 | ネヤガワコ | 大阪府寝屋川市本町 | | 34.7651 | 135.6259 | |
| 8 | 7 | 府立大学コ | フリツダイ | 大阪府寝屋川市幸町 | | 34.77043 | 135.629 | |
| 9 | 8 | 池田小学校 | イケダショ | 大阪府寝屋川市池田 | | 34.7732 | 135.6171 | |
| 10 | 9 | 桜小学校 | サクラショ | 大阪府寝屋川市池田 | | 34.77298 | 135.6115 | |
| 11 | 10 | 第二中学校 | ダイニチュ | 大阪府寝屋川市池田 | | 34.76956 | 135.611 | |

  - CSV形式のテキストデータをBit Arrowに登録してプログラムから利用可能

# 避難場所のデータを地図に表示する

```
import pandas as pd
import folium

# 避難場所の位置情報を取得
shelters = pd.read_csv("class/272159_evacuation_space.csv", encoding="shift-jis")

# 地図作成
map = folium.Map(location=[shelters["緯度"][0], shelters["経度"][0]], zoom_start=12)

# 避難場所を1か所ずつマーク
for i in range(len(shelters)):
    latlng = shelters.loc[i, ["緯度", "経度"]].tolist()
    name = shelters.loc[i, "名称"]
    folium.Marker(latlng, popup=name).add_to(map)

# 地図表示
map.show()
```

# まとめ

# 全体のまとめ

- 代表的なアルゴリズムをいくつか見た
  - 最大値、探索（サーチ）と整列（ソート）。カードでの学習の様子を紹介した
- アルゴリズムの学習
  - プログラムを書いて学べる：　最大値、線形探索
  - カードで楽しく学べる：　二分探索、バブル/選択、クイック、バケット
  - プログラムを読んで理解できる：　二分探索、（バブル/選択）、バケット
- 他のアルゴリズムも調べてみてください
  - 探索：　ハッシュ探索はO(1)の超高速アルゴリズム
  - 整列：　マージソート、ヒープソートも速い。挿入ソートはトランプと同じ
    基数ソートは桁ごとの数に色を付けてカードで学べる
- アルゴリズムの改良も楽しい
- グラフィックスで可視化することも有効です
- 学習システムを公開しています。ご利用ください
  Bit Arrow、サクセス、ConnectDB、ドリトル