情報科学概論

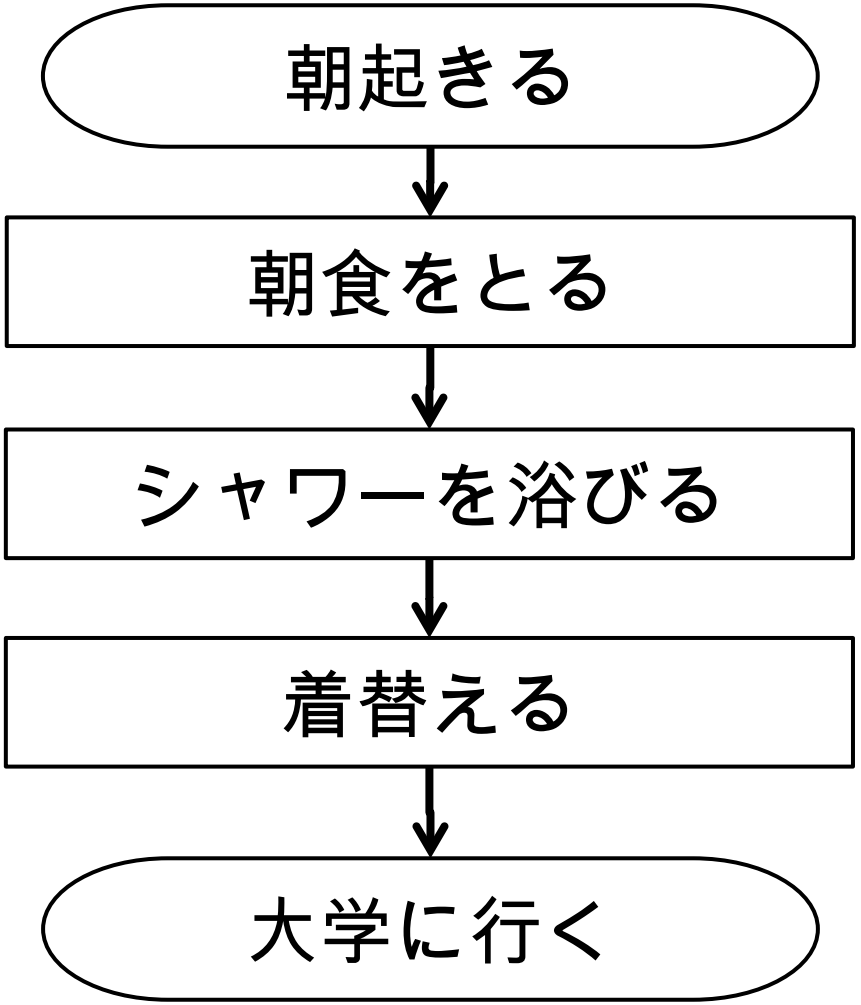
プログラムとアルゴリズム

アルゴリズム

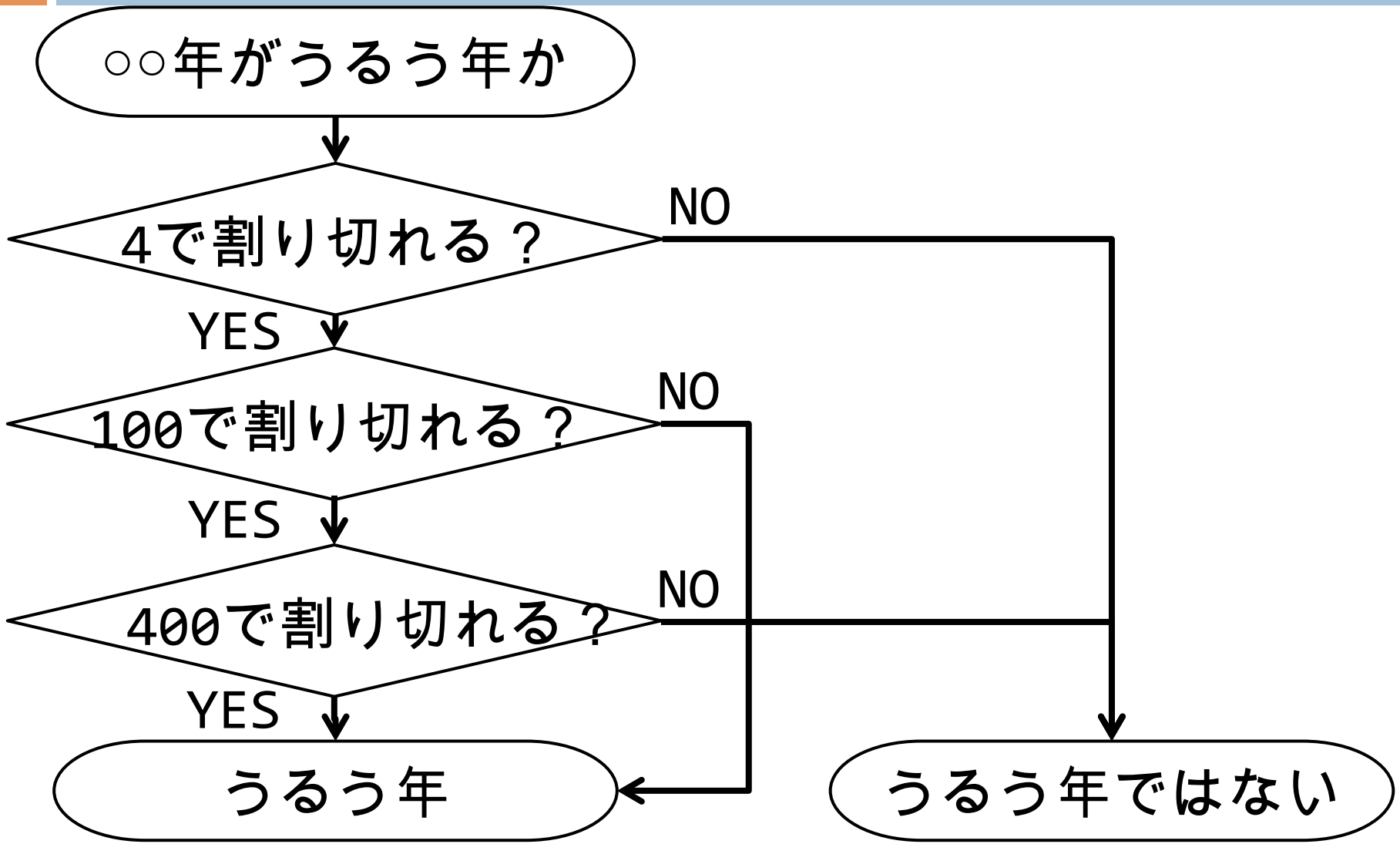
- 何かをする(効率的な)手順
- 情報科学の本質はアルゴリズム
  - 賢いアルゴリズムによる性能向上
    >> ハードウェアの改良による性能向上

1. フローチャート(流れ図)を使って図示
   - 直線型・分岐型・繰り返し型
2. プログラミング言語で記述

フローチャートの例



分岐のあるアルゴリズムの例



対応するプログラム(簡略版)

```
if(year % 4 == 0){
  print("うるう年")
}
else{
  print("うるう年ではない")
}
```

原始プログラム
(プログラミング言語)

コンパイル

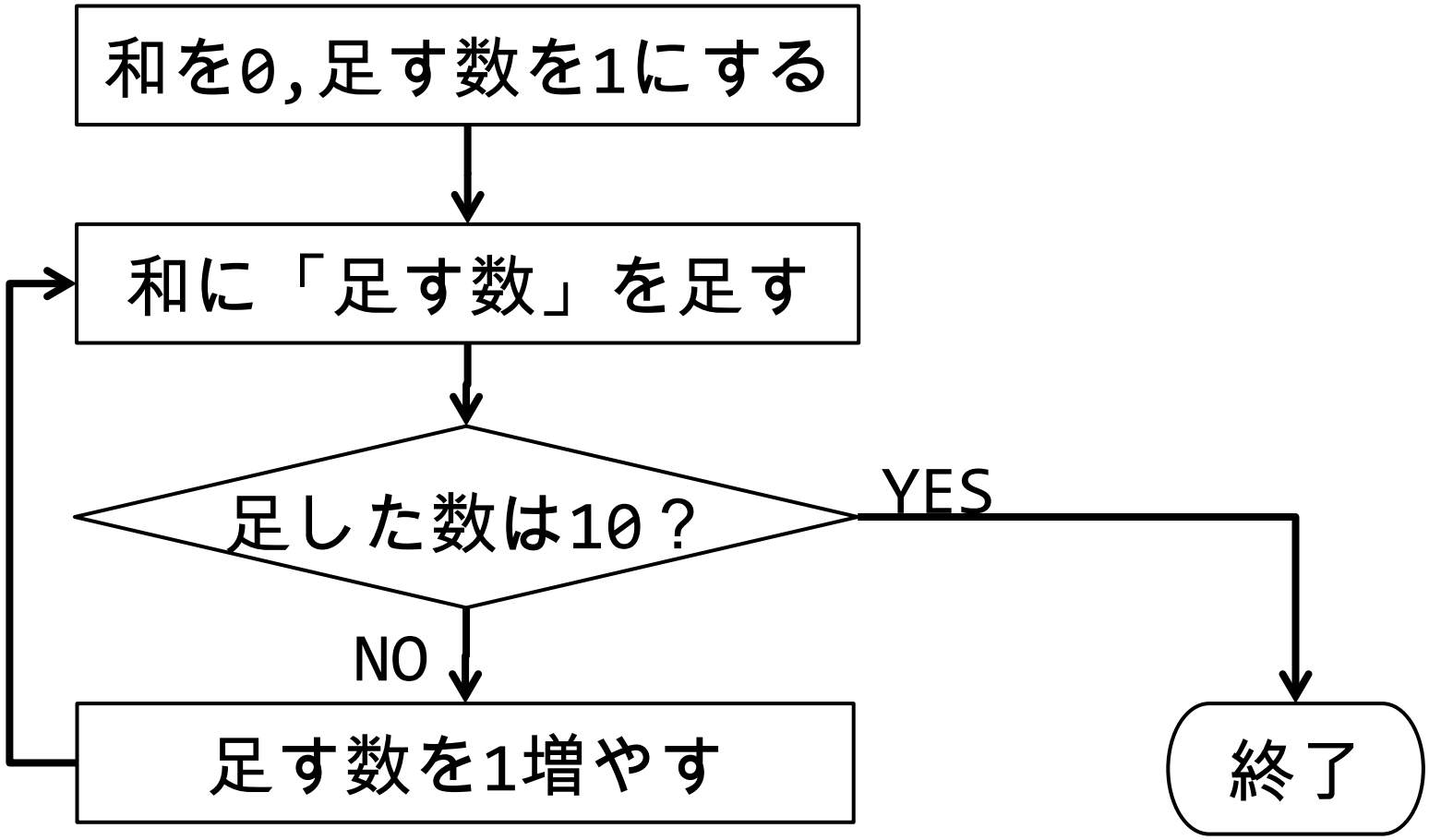
目的プログラム
(機械語)

001	レジスタ1の値を4で割った余りをレジスタ2に保存
002	レジスタ2の値と0を比較
003	等しかったら 006 へジャンプ
004	print("うるう年ではない")
005	007 にジャンプ
006	print("うるう年")
007	つづく...

実行プログラム

ライブラリとリンク
(ここでは print など)

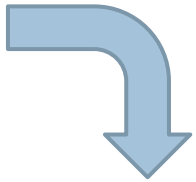
繰り返しのあるアルゴリズムの例



対応するプログラム

```
sum = 0
n = 1
while(n != 10){
   sum = sum + n
   n = n + 1
}
print(sum)
```

コンパイル



001	レジスタ1の値を0にする
002	レジスタ2の値を1にする
003	レジスタ2の値と10を比較する
004	等しかったら008にジャンプ
005	レジスタ1にレジスタ2を加えて保存
006	レジスタ2に1を加えて保存
007	003にジャンプ
008	print(レジスタ1)

コンパイラのしてくれること

- 対応する機械語の命令に変換する

＋

- レジスタを割り当てる
  - 例：sumはレジスタ1，nはレジスタ2
- 制御を複数の分岐命令に置き換える

レジスタの番号
命令のアドレス
} などを意識しないでプログラムを書ける

アルゴリズムの良し悪し

- 高速
  - いつでも速い
  - 速いときが多い（運が悪いと遅い）
- メモリの使用量が少ない

実行効率

- シンプル
- いろいろな問題に使える

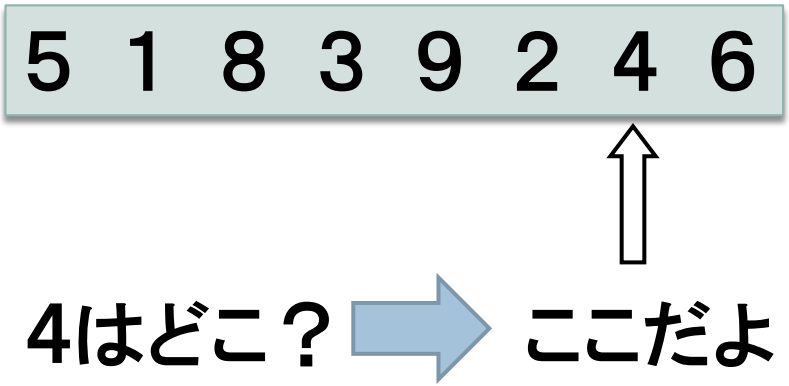
開発効率

- 計算結果の誤差が小さい

代表的なアルゴリズム

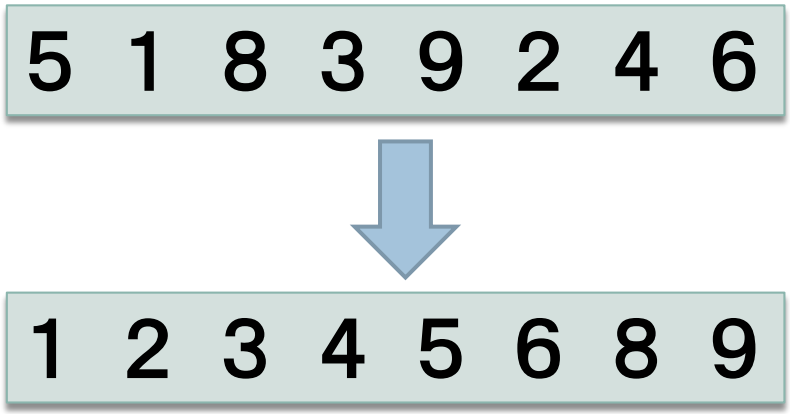
探索

- たくさんのデータから目標物を見つける



整列（ソート）

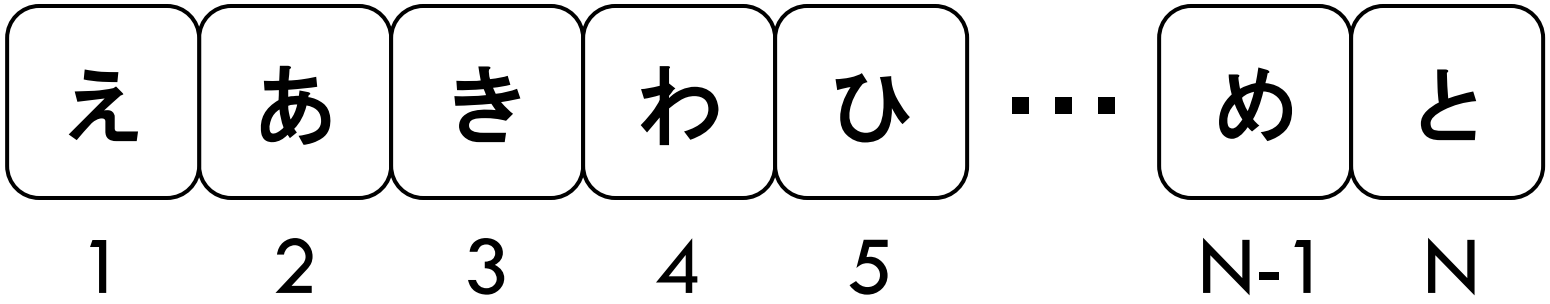
- 大きい順、小さい順などで並びかえる



探索アルゴリズム

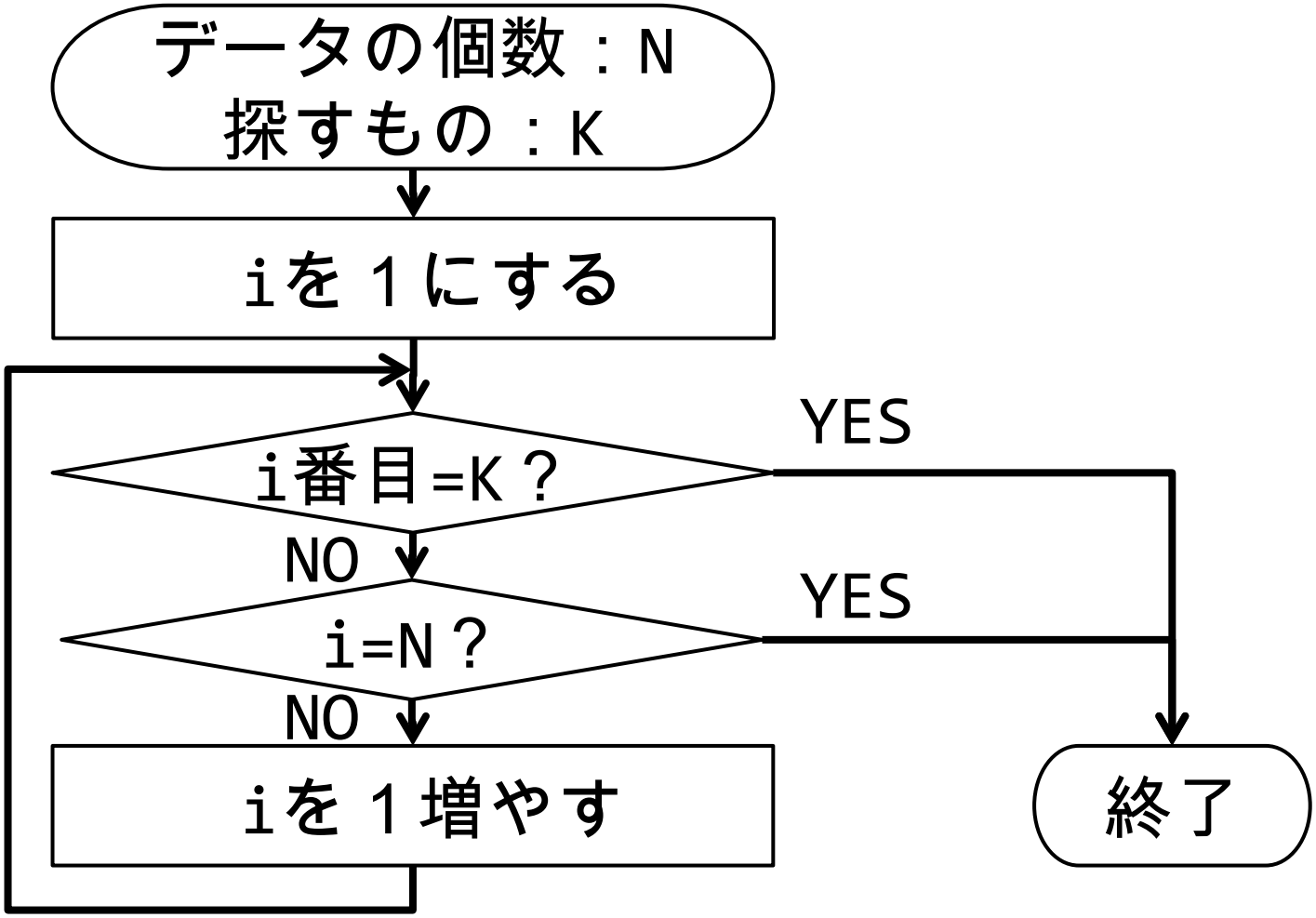
探索

- データの個数：N個
  - データには1～Nの番号が振られている



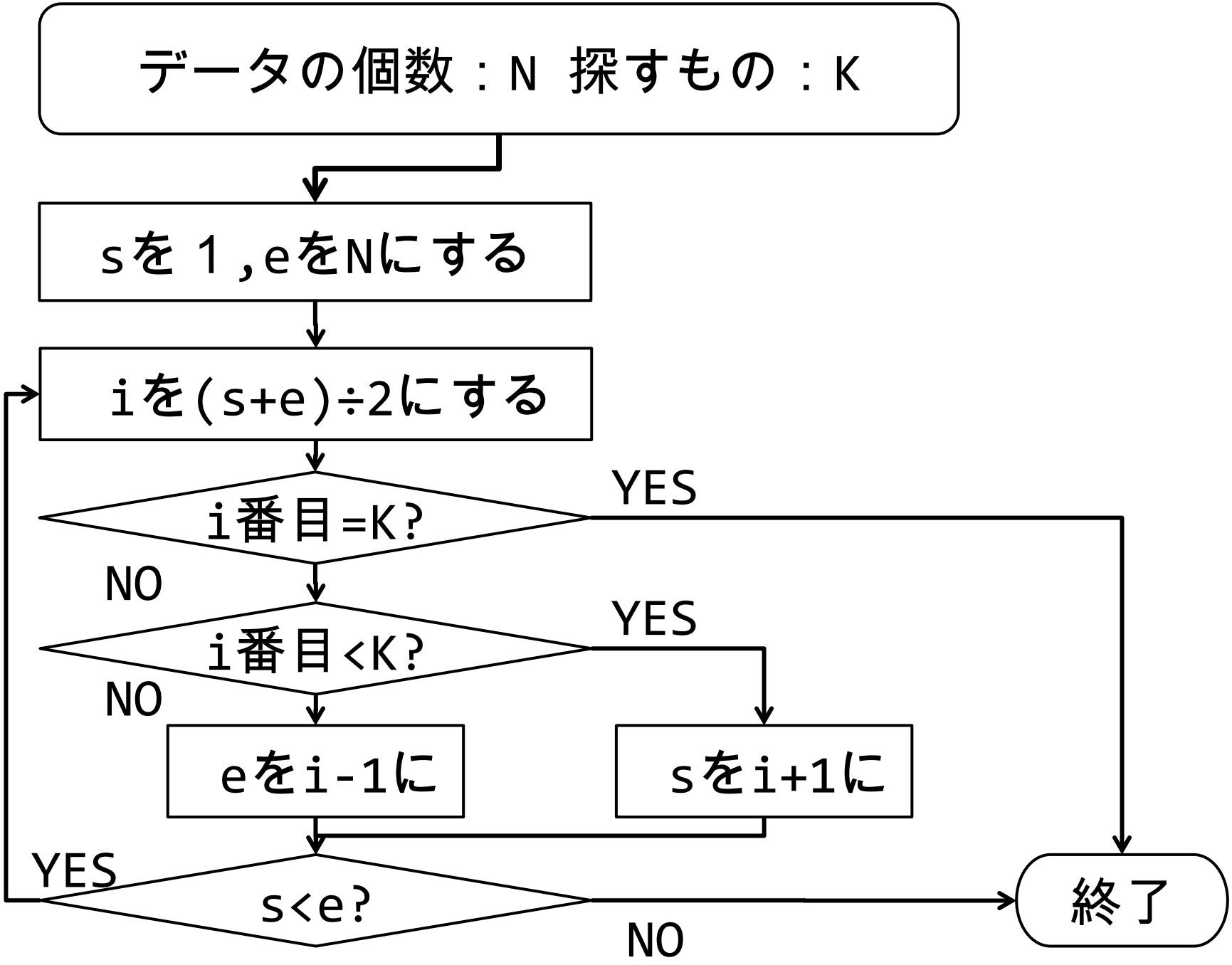
- あるデータがどの位置（何番目）にあるか探す
  - 例：「き」を探す→3番目にある
- 1度に可能な処理
  - 2つのデータを比較する

線形探索（Linear search）



二分探索（binary search）

- あらかじめデータを整列しておくと、速く見つけることができる
- 整列したデータの真ん中にあるデータと比較する
→真ん中より手前にあるか後にあるかわかる


データの個数：N 探すもの：K
sを１,eをNにする
iを(s+e)÷2にする
i番目=K?
YES
NO
i番目<K?
YES
NO
eをi-1に
sをi+1に
YES
s<e?
NO
終了

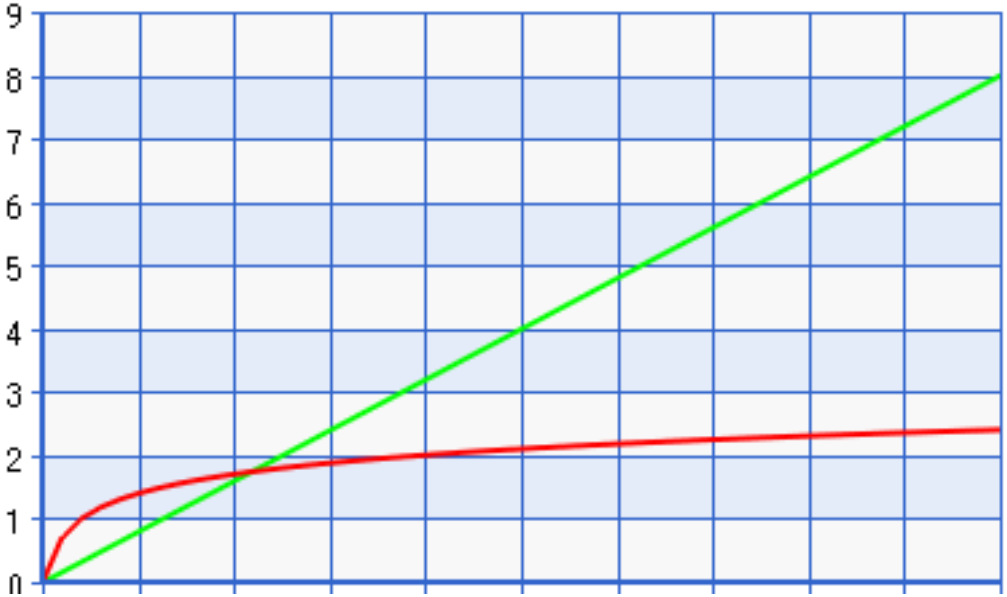
線形探索と二分探索の比較

線形探索

- 最悪の場合 n 回比較
- 下準備が不要

二分探索

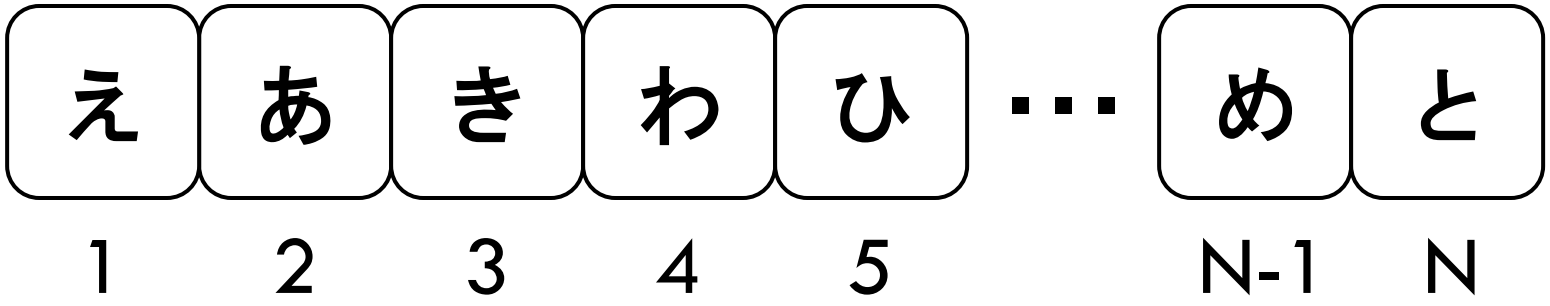
- 最悪の場合 $\log_2(n)$
- 下準備(整列)が必要



整列アルゴリズム

整列

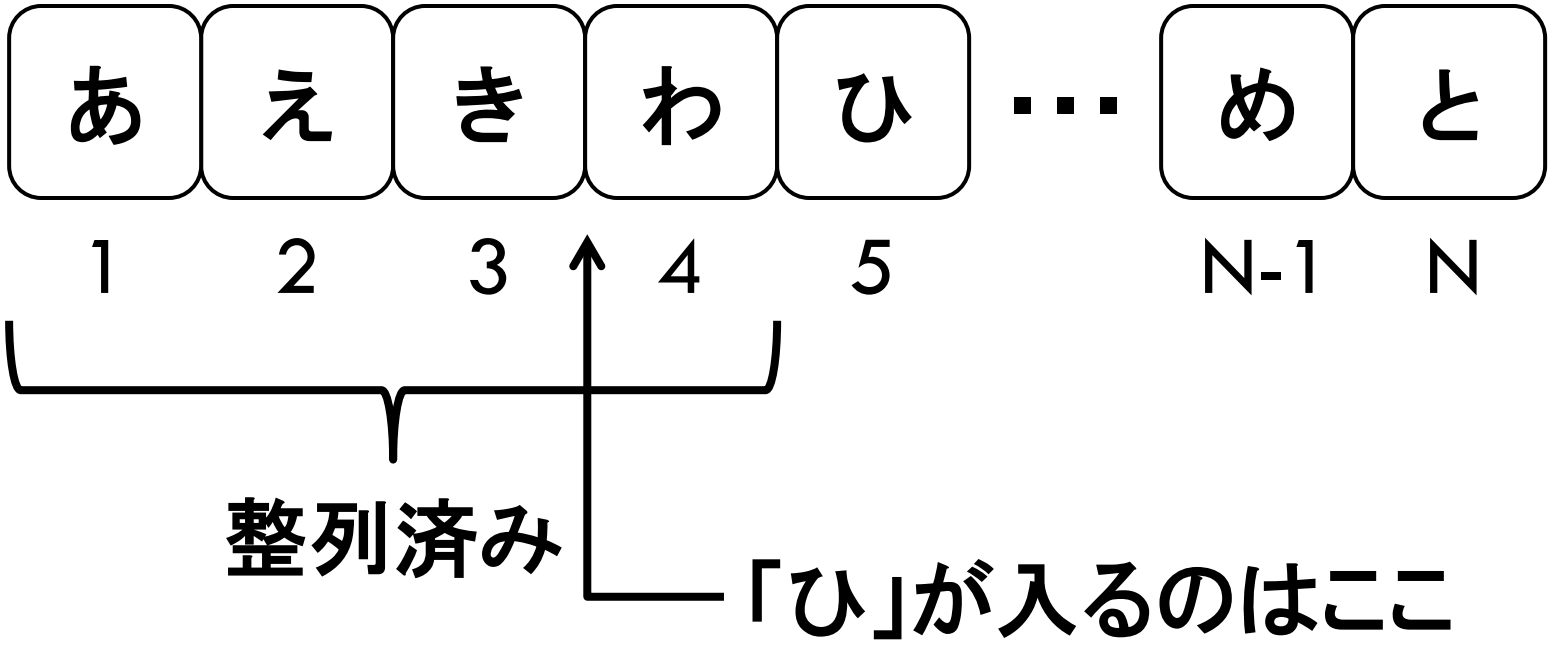
- データの個数：N個
  - データには1～Nの番号が振られている

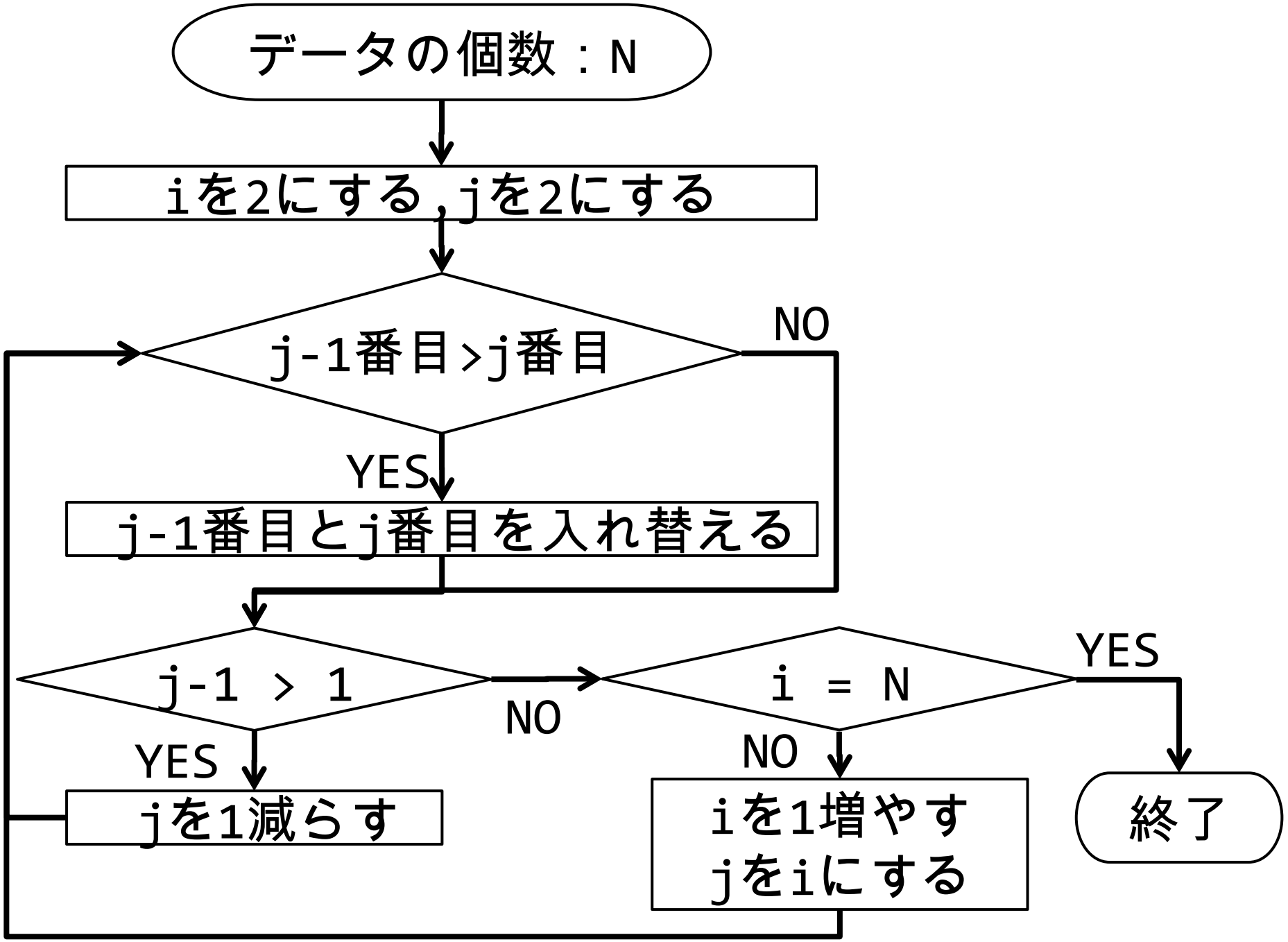


- 1度に可能な処理
  - 2つのデータを比較する
  - 2つのデータの位置を入れ替える

挿入ソート（insertion sort）

- 整列済み部分に1つずつデータを挿入して、整列済み部分を広げていく




データの個数：N
iを2にする，jを2にする
j-1番目>j番目
NO
YES
j-1番目とj番目を入れ替える
j-1 > 1
NO
i = N
YES
YES
jを1減らす
NO
iを1増やす
jをiにする
終了

クイックソート

- もっとも高速な整列アルゴリズムのひとつ

1. 適当な数を選択する(ピボット)
2. ピボットより小さい数と大きい数に分割する
3. 二分割されたデータを、それぞれソートする

挿入ソートとクイックソートの比較

挿入ソートの実行時間

- n^2 に比例

クイックソートの実行時間

- 平均 $n\log_2(n)$
- 最悪 n^2 かかる