情報処理技法
(Javaプログラミング)2

第3回
アルゴリズム

人間科学科コミュニケーション専攻

白銀 純子

第3回の内容

- アルゴリズム

前回の出席課題の解答

- 下記のプログラムを実行したとき、確実に標準出力に出力されるメッセージは何かを答えなさい。

```
try {
   String line[] = new String[5];
   int i;

   FileReader  fr = new  FileReader("sample.txt");
   BufferedReader  br = new  BufferedReader(fr);

   for(i = 0; i < 100; i = i + 1) {
     line[i] = br.readLine();
   }
   br.close();
   fr.close();
}
catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("指定された範囲外の添え字を使おうとしています。");
}
catch (NumberFormatException e) {
    System.out.println("数値に変換できない文字列を数値に変換しようとしています。");
}
catch (FileNotFoundException e) {
    System.out.println("指定されたファイルが存在しません。");
}
catch (IOException e) {
    System.out.println("指定されたファイルを読み込むことができません。");
}
```

確実に標準出力に
出力されるメッセージ

アルゴリズム

アルゴリズムとは

- アルゴリズム: ある問題を解決するときに必要な処理手順
  - プログラムを書くときには、必ずアルゴリズムを考える必要
  - プログラム: アルゴリズムをプログラミング言語を使って記述したもの

Ex. 焼きそば作りのアルゴリズム:

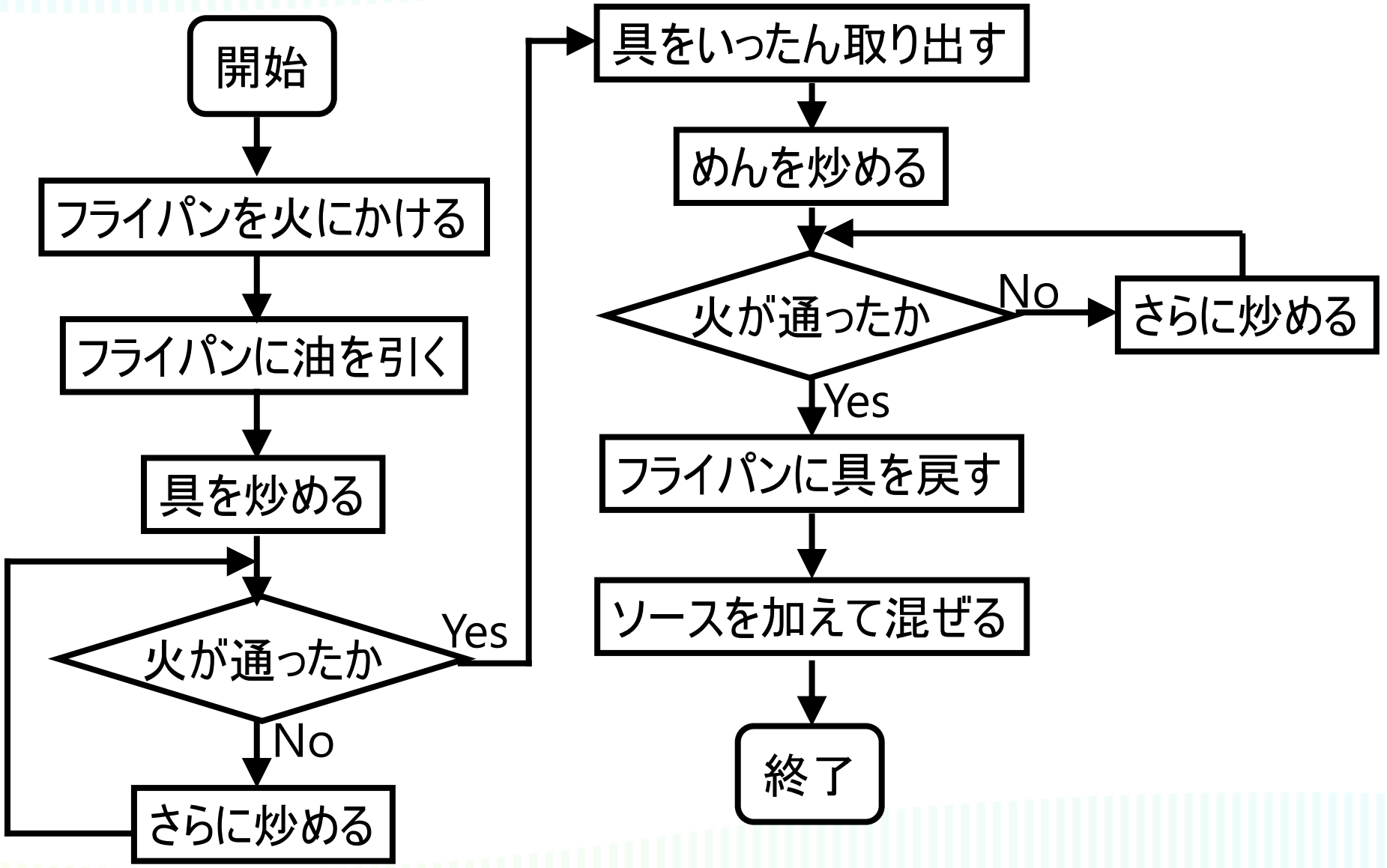
1. キャベツや肉などの具をフライパンで炒めなさい。
2. フライパンからいったん具を取り出しなさい。
3. めんをフライパンで炒めなさい。
4. 具をめんの入っているフライパンに戻しなさい。
5. 焼きそばソースを加えてさらに炒めなさい。

アルゴリズムの表現方法

- 文章で書く
  - 箇条書きで書くことも多い
- 図で書く
- プログラムで書く
  - プログラムが最終段階

プログラムで行う処理手順がわからなければ、まず文章で書き出して、
それを詳細にしていく

焼きそば作成(図)



焼きそば作成(プログラム1)

焼きそばのアルゴリズムをプログラム的に表現

```
public static void main(String[] args) {
  // 開始
  フライパンを火にかける;
  フライパンに油を引く;
  具を炒める;
  while (火が通っていない) {
    // Yes(「火が通ったか?」に対して「No」)
    さらに炒める
  }
  //「火が通ったか?」に対して「Yes」
  具をいったん取り出す;
  めんを炒める;
```

焼きそば作成(プログラム2)

焼きそばのアルゴリズムをプログラム的に表現(続き)

```
while (火が通っていない) {
   // Yes(「火が通ったか?」に対して「No」)
   さらに炒める;
 }
 //「火が通ったか?」に対して「Yes」
 フライパンに具を戻す;
 ソースを加えて混ぜる;
 // 終了
}
```

最大公約数のアルゴリズム

素因数分解

1. 1つ目の数を素因数分解する
2. 2つ目の数を素因数分解する
3. それぞれの素因数の共通項を掛け合わせる

Ex. 136と24の最大公約数
$136 = 2 \times 2 \times 2 \times 17$
$24 = 2 \times 2 \times 2 \times 3$

➡ 共通項: $2 \times 2 \times 2 = 8$

➡ 最大公約数: 8

最大公約数

- 2つの数の最大公約数を求めるアルゴリズムは2通り
  - 素因数分解で求める
  - ユークリッドの互助法で求める

ユークリッドの互助法

1. 2つの数のうち、大きい方を小さい方で割る
2. 1. の余りが0でない場合、その余りと小さい方の数を改めて2つの数とし、1. へ戻る
3. 1. の余りが0の場合、割った数が最大公約数となる

Ex. 136と24の最大公約数

136÷24 = 5 … 16

24÷16 = 1 … 8

16 ÷8 = 2 … 0

最大公約数: 8

筆算のアルゴリズム

自然数の加算(1)

- 2つの自然数の和を求める = 筆算
  - n桁目の数の和を求め、10以上になった場合にはn+1桁目に1を加える

nが1から自然数の桁数まで繰り返す

- 準備
  - 2つの自然数は同じ桁数とする
    - 50と200の場合、050と200として考える
  - 加えた結果を入れる変数(配列が便利)を用意する
  - 繰り上がりを入れる変数を用意し、0を代入しておく

自然数の加算(2)

1. 1の位から最上位の位まで、a. ～d. を繰り返す
   a. その位の2つの数の和を求める
   b. 下位からの繰り上がりがあればそれも加える
   c. その和の1桁目を、求める和のその位の結果とする
   d. その和の2桁目を、次の桁への繰り上がりとする
2. 最上位からの繰り上がりがあった場合は、求める和のその次(上位)の位の数とする

ATMのアルゴリズム

ATM(1)

- 人間から見える手順
  1. 「引き出し」ボタンを押す
  2. キャッシュカードを入れる
  3. 暗証番号を入力する
  4. 引き出し金額を入力する
  5. 現金を取り出す

ATM(2)

- ATM側での処理手順

1. 初期画面を表示してボタンが押されるのを待つ
2. 押されたボタンによってサービスの種類を判別する
   a. 「引き出し」ボタンの場合、3. へ進む
   b. 「預け入れ」ボタンの場合、4. へ進む

   ……
3. サービスの種類が「引き出し」の処理
4. サービスの種類が「預け入れ」の処理

   ……
5. 「ありがとうございました」の画面を表示して、1. へ戻る

ATM(3)

- 「引き出し」の処理手順
  a. カードを入れてくださいと表示する
  b. 適切なカードかどうかを調べ、キャッシュカードではないカードであれば一. へ進む
     一. 入れられたカードを排出し、「キャッシュカードを入れなおしてください。」と表示する
     二. b. へ戻る
  c. 暗証番号を調べ、番号が間違っていたら一. へ進む
     一. 「暗証番号を入力しなおしてください。」と表示する
     二. c. へ戻る
  d. 「引き出す金額を入力してください。」と表示し、引き出し金額を入力してもらう
  e. 引き出し金額を調べる

ATM(3)

- 「引き出し」の処理手順の続き

f. 引き出し金額が残高よりも多ければ、一. へ進む

一. 「残高を超えています。引き出し金額を入力しなおしてください。」と表示する

二. f. へ戻る

g. 出金処理を行い、預金残高を更新する

h. 預金残高を表示し、5. へ進む

有名なアルゴリズム

並べ替え(ソート)

- ソート: 複数の数を小さい or 大きい順に並べること
  - バブルソート
  - 選択ソート
  - 併合ソート
  - クイックソート
  - etc.

バブルソート(1)

- 前から2つずつ、数の大きさを比較して、小さい数を後ろに送っていく
  - 最後まで調べると、最も小さな数が一番後ろにある
  - この作業を、並べ替える数の個数だけ繰り返すと、数が大きい順に並ぶ
- $x_1 < x_2$ならば、x_1とx_2を入れ替える
- $x_2 < x_3$ならば、x_2とx_3を入れ替える
- …
- $x_{n-1} < x_n$ならば、x_nとx_{n-1}を入れ替える

  - n個の数でこの処理が終わった後にn番目に最も小さな数
  - n-1回この処理を繰り返すことで、大きい順に数を並べ替え

バブルソート(2)

- 最も小さな数が一番後ろに来る
- 次は、一番後ろの1つ前まで　(8, 4, 5)で同じようにする

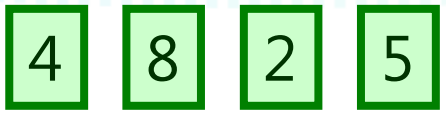
選択ソート(1)

- 変数t: 並べ替えをする数の中で最も小さい数を入れておく変数
- 変数i: 並べ替えをする数の中で、tが最初から何番目の位置にあるかを表す変数
- 変数j: 何回繰り返したかを数えるための変数
- 並べ替えをする数はn個

選択ソート(2)

1. tに並べ替えをする数の一番最初の数を代入する
2. iに1を代入する
   - 1: 並べ替えをする数の一番最初の数の位置
3. jに2を代入し、1ずつ増やしながらnになるまで以下を繰り返す
   - tがj番目の数より大きいならば、j番目の数をtに代入し、iにjの値を代入する
     - この処理を1回するごとにjの値を1増やす
     - jの値は、現在調べている数の位置になる
4. 並べ替えをする数の一番最後の数とtを入れ替える

選択ソート(2)



ステップ1:
- tに4を代入する
- iに1を代入する

ステップ2:
- jに2を代入する
- tの値(4)と8を比べる
- tの値の方が小さいのでそのまま

ステップ3:
- jの値を1増やす(3になる)
- tの値(4)と2を比べる
- tの値の方が大きいのでtに2を代入し、iにjの値(3)を代入する

ステップ4:
- jの値を1増やす
- tの値(2)と5を比べる
- tの値の方が小さいのでそのまま

ステップ5:
- tの値(2)を最後に置く
- これまで最後だった数(5)をi番目に入れる

- 最も小さな数が一番後ろに来る
- 次は、一番後ろの1つ前まで(8, 4, 5)で同じようにする

探索(1)

- たくさんのデータから目的のデータを見つけること

Ex. 高校の生徒の得点を管理するプログラム

- 出席番号5番の生徒の英語の点数を知りたい
  →高校の生徒のデータから、出席番号が「5」というものを探す
- 「東京子」という生徒の国語の点数を知りたい
  →高校の生徒のデータから、名前が「東京子」というものを探す

探索(2)

- 探索のアルゴリズム
  - 逐次探索
  - 2分探索
  - 自己組織化探索
  - 2次元探索
  - 補間探索
  - ハッシュ法

逐次探索

- データを前から順番に比較して探していく方法

1. データが配列a[0]～a[N]に入っている
2. 「a[i] ＝ x」(xは探したいデータ)となる「i」を探す
   - 「i」は0～Nの整数

- 欠点: データが配列の後ろのほうにある場合に見つけるまでに時間がかかる

2分探索(1)

- ソートされたデータの中から目的のデータを探す方法

1. データが配列a[0]～a[N]に入っている
2. 中央のデータ「a[(0 + N)/2]」と「x」を比較する
   - x: 探したいデータ
3. 2. の結果、a[m]<xの場合、データは右半分にx<=a[m]の場合、データは左半分に存在
   - m: (0 + N)/2
   - 右半分: a[m+1]～a[N]
   - 左半分: a[0]～a[m]

2分探索(2)

4. データが右半分にある場合、a[m+1]～a[N]で2. をする
5. データが左半分にある場合、a[0]～a[m]で2. をする
6. 「a[left]～a[right]」での探索で、「left = right」となると探索終了
7. 目的のデータは「a[left]」

2分探索(3)

「1, 2, 3, 4, 5, 6, 7, 8, 9, 10」の中から「9」を見つけたい

- a[(0+9)/2](値: 5)と「9」を比較し、「9」の方が大きいので、「9」は右半分(a[(0+9)/2+1]～a[9])の範囲にある
- a[(5+9)/2](値: 8)と「9」を比較し、「9」の方が大きいので、「9」は右半分(a[(5+9)/2 + 1]～a[9]の範囲にある
- a[(8+9)/2](値: 9)と「9」を比較し、「9」と同じなので、「9」は左半分(a[(5+9)/2]～a[(8+9)/2])の範囲にある
- a[(7+8)/2](値: 8)と「9」を比較し、「9」の方が大きいので、「9」は右半分(a[(7+8)/2+1]～a[8])の範囲にある
- 探索する配列の添え字が同じになったので、探索終了
- 結果: a[8]が回答

やってみよう!

- 逐次探索
  1. 配ったカードをバラバラの順序にする
  2. その中から「18」を探してみる
- 2分探索
  1. 配ったカードを昇順に並べる
  2. カードの中から「18」を探してみる

アルゴリズムの良し悪し

良いアルゴリズム

- そのアルゴリズムによるプログラムを実行するときの計算時間や記憶領域の使用量の少なさ
- アルゴリズムのわかりやすさ・作りやすさ・修正の容易さ

アルゴリズムの計算時間

- CPUそのものの速さや、コンパイラに大きく依存
- ファイルの入出力時の記憶装置とのアクセスの速度に依存
- メインメモリが少ない場合、メインメモリとHDDの間でデータが頻繁に行き来(スラッシング)

これらを除いても、同じ結果を出す複数のアルゴリズムで、計算時間に違いが出る

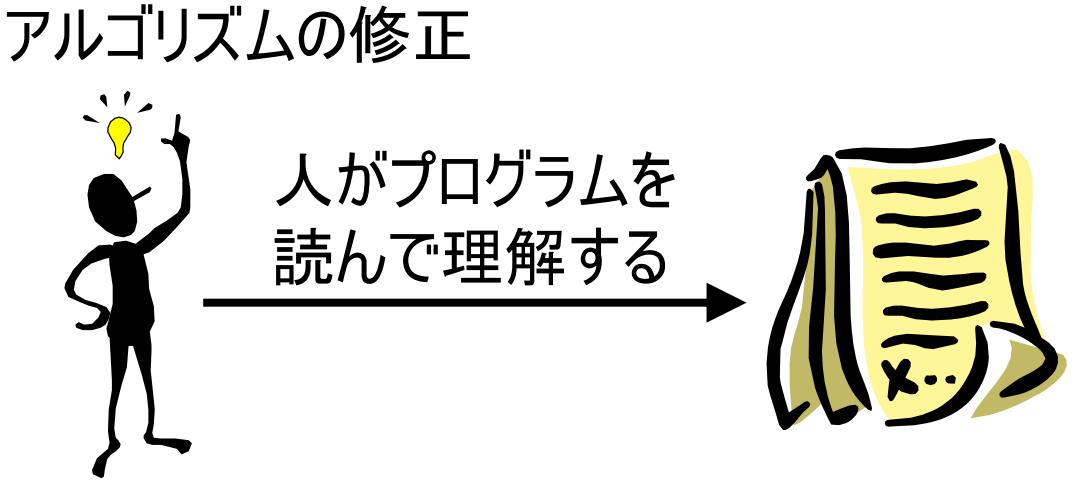
➡ アルゴリズムの計算量

アルゴリズムの計算量

- CPUの速さなど、アルゴリズムには関係ない要因を除いた、アルゴリズムそのものの計算時間
  - Ex. for文で繰り返す回数, 足し算・かけ算などの回数, etc.

アルゴリズムのわかりやすさ

- 一旦完成したプログラム: 機能の追加などのためにプログラムの修正が必要なことも多い



プログラムを読んで理解する = 書かれてあるアルゴリズムの理解が必要

- アルゴリズムが難解 = 修正が難しい
- アルゴリズムが簡単 = 修正が容易

アルゴリズムを使う状況

- 多くの場合、計算時間の速いアルゴリズムとわかりやすいアルゴリズムは対立関係
  - 計算時間が速ければ、わかりにくいアルゴリズム
  - わかりやすければ、計算時間が遅い

速さをとるか、わかりやすさをとるかは、状況に応じて判断

- めったに使わないプログラムや頻繁に修正するプログラム: わかりやすいアルゴリズム
- よく使うプログラムや計算時間に制約があるプログラム: 速いアルゴリズム

並べ替えアルゴリズムの比較(1)

- 結果が出るまでの基本処理の回数(アルゴリズムの計算量)
  - バブルソート: $N(N-1)/2$
  - クイックソート: $N\log_2(N)$

  ※$\log_2(N)$: Nを2^kとしたときの「k」

N: 並べ替える数の個数

N	$N(N-1)/2$(バブル)	$N\log_2(N)$(クイック)
8	28	24
32	496	160
64	2016	384
128	8128	896

Nが大きければ大きいほど、クイックソートの方が速い

並べ替えアルゴリズムの比較(2)

- 計算量: 入力(N: 並べ替えの場合は数の個数)に対して行われる基本処理の回数
  - Nが十分に大きなとき: 計算式の中の最も大きな項だけに着目して、大まかに計算

    = 各項の比例定数や次数の低い項は無視
    - バブル: $N(N-1)/2 = N^2/2 - N/2$
      →N^2のみに注目
    - クイック: $N\log_2 N$
      →$N\log_2 N$のみに注目

アルゴリズムの計算量は、正確な計算量ではなく、Nが大きくなればどの程度の割合で計算量が増えるかを大まかに知ることが重要なため

並べ替えアルゴリズムの比較(3)

- 計算時間の速いアルゴリズム: NやlogNなどのみで計算量が計算できるアルゴリズム
- 計算時間の遅いアルゴリズム: N^2, N^3, ..., N^kやN!(1からNまでをかけあわせた数), 2^Nなど、多くのかけ算を計算に必要とするアルゴリズム
  - N^2, N^3などの計算を必要とするアルゴリズム: 多項式時間アルゴリズム
  - N!や2^Nなどの計算を必要とするアルゴリズム: 指数時間アルゴリズム

準備

- 授業のページから2つのファイルをダウンロード
  - BubbleSort.java
    - バブルソートをするプログラム
  - QuickSort.java
    - クイックソートをするプログラム

BubbleSort.java(前半1)

並べ替えをする
数の個数

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
  random[i] = (int) (num * Math.random());
}
try {
  File file = new File("BubbleSort.txt");
  FileWriter fw = new FileWriter(file);
  PrintWriter pw = new PrintWriter(fw);

  pw.println("並べ替え前");
  for (i = 0; i < num; i++) {
    pw.write(Integer.toString(random[i]) + ", ");
  }
  Calendar cal1 = Calendar.getInstance();
  long beforeTime = cal1.getTimeInMillis();
```

BubbleSort.java(前半2)

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
  random[i] = (int) (num * Math.random());
}
try {
  File file = new File("BubbleSort.txt");
  FileWriter fw = new FileWriter(file);
  PrintWriter pw = new PrintWriter(fw);

  pw.println("並べ替え前");
  for (i = 0; i < num; i++) {
    pw.write(Integer.toString(random[i]) + ", ");
  }
  Calendar cal1 = Calendar.getInstance();
  long beforeTime = cal1.getTimeInMillis();
```

並べ替えをする数を
乱数で作成

BubbleSort.java(前半3)

```
int num = 10000;
int i;
int[] random = new int[num];
for (i = 0; i < num; i++) {
  random[i] = (int) (num * Math.random());
}
try {
  File file = new File("BubbleSort.txt");
  FileWriter fw = new FileWriter(file);
  PrintWriter pw = new PrintWriter(fw);

  pw.println("並べ替え
  for (i = 0; i < num; i++) {
    pw.write(Integer.toString(random[i]) + ", ");
  }
  Calendar cal1 = Calendar.getInstance();
  long beforeTime = cal1.getTimeInMillis();
```

数を並べ替える直前の時間を計測

BubbleSort.java(後半1)

```
 for (i = 0; i < num; i++) {
   for (j = 0; j < num - i - 1; j++) {
     if (random[j] > random[j + 1]) {
       temp = random[j + 1];
       random[j + 1] = random[j];
       random[j] = temp;
     }
   }
 }
 Calendar cal2 = Calendar.getInstance();
  long afterTime = cal2.getTimeInMillis();
  System.out.println("かかった時間(バブルソート): " + (afterTime - beforeTime) + "ミリ秒");

  pw.println("¥n¥n並べ替え後");
  for (i = 0; i < num; i++) {
    pw.write(Integer.toString(
  }
  fw.close();
  pw.close();
}
catch(IOException e) {  }
```

バブルソートをする処理部分

BubbleSort.java(後半2)

```
 for (i = 0; i < num; i++) {
   for (j = 0; j < num - i - 1; j++) {
     if (random[j] > random[j + 1]) {
       temp = random[j + 1];
       random[j + 1] = tar
       random[j] = temp;
     }
   }
 }
 Calendar cal2 = Calendar.getInstance();
 long afterTime = cal2.getTimeInMillis();
 System.out.println("かかった時間(バブルソート): " + (afterTime - beforeTime) + "ミリ秒");

 pw.println("¥n¥n並べ替え後");
 for (i = 0; i < num; i++) {
   pw.write(Integer.toString(random[i]) + ", ");
 }
 fw.close();
 pw.close();
}
catch(IOException e) { }
```

数を並べ替えた直後の時間を計測

QuickSort.java(前編)

```
public static void sort(int[] rand, int first, int last) {
  int i, j, temp, x = rand[(first + last) / 2];

  i = first;
  j = last;
  while (true) {
    while (rand[i] < x) {
      i = i + 1;
    }
    while (x < rand[j]) {
      j = j - 1;
    }
    if (i >= j) {
      break;
    }
    temp = rand[j];
    rand[j] = rand[i];
    rand[i] = temp;
    i = i + 1;
    j = j - 1;
  }
```

クイックソートの
処理部分
(メソッドとして定義)

QuickSort.java(中編1)

```
    if (first < i - 1) {    sort(rand, first, j - 1);  }
    if (j + 1 < last) {    sort(rand, j + 1, last);  }
}
public static void main(String[] args) {
  int i, j, temp, num = 1
  int[] random = new in
  for (i = 0; i < num; i++
    random[i] = (int) (nu
  }
  try {
    File randFile = new File("QuickSort.txt");
    FileWriter fw = new FileWriter(randFile);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
```

クイックソートの
処理部分
(メソッドとして定義～続き～)

QuickSort.java(中編2)

```
    if (first < i - 1) {    sort(rand, first, j - 1);  }
    if (j + 1 < last) {    sort(rand, j + 1, last);  }
}
public static void main(String[] args) {
  int i, j, temp, num = 10000;
  int[] random = new int[num];
  for (i = 0; i < num; i++) {
    random[i] = (int) (num * Math.r
  }
  try {
    File randFile = new File("QuickSort.txt");
    FileWriter fw = new FileWriter(randFile);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
```

並べ替えをする数の個数

QuickSort.java(中編3)

```
    if (first < i - 1) {    sort(rand, first, j - 1);  }
    if (j + 1 < last) {    sort(rand, j + 1, last);  }
}
public static void main(String[] args) {
  int i, j, temp, num = 10000;
  int[] random = new int[num];
  for (i = 0; i < num; i++) {
    random[i] = (int) (num * Math.random());
  }
  try {
    File randFile = new File("QuickSort.txt");
    FileWriter fw = new FileWriter(randFile);
    PrintWriter pw = new PrintWriter(fw);

    pw.println("並べ替え前");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
```

並べ替えをする数を
乱数で作成

QuickSort.java(後編1)

```
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();

    sort(random, 0, num - 1);

    Calendar cal2 = Calendar.getInstance();
    long afterTime = cal2.getTimeInMillis();
    System.out.println("かかった時間(クイック): " + (afterTime - beforeTime) + "ミリ秒");

    pw.println("¥n¥n並べ替え後");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
    pw.close();
    fw.close();
  }
  catch(IOException e) {
  }
}
```

数を並べ替える直前の時間を計測

QuickSort.java(後編1)

```
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();

    sort(random, 0, num - 1);

    Calendar cal2 = Calendar.getInstance();
    long afterTime = cal2.ge
    System.out.println("かかった時間(ミリ秒): " + (afterTime - beforeTime) + "ミリ秒");

    pw.println("¥n¥n並べ替え後");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
    pw.close();
    fw.close();
  }
  catch(IOException e) {
  }
}
```

クイックソートのメソッドを呼び出す処理

QuickSort.java(後編3)

```
    Calendar cal1 = Calendar.getInstance();
    long beforeTime = cal1.getTimeInMillis();;

    sort(random, 0, num

    Calendar cal2 = Calendar.getInstance();
    long afterTime = cal2.getTimeInMillis();
    System.out.println("かかった時間(クイック): " + (afterTime - beforeTime) + "ミリ秒");

    pw.println("¥n¥n並べ替え後");
    for (i = 0; i < num; i++) {
      pw.print(random[i] + " ");
    }
    pw.close();
    fw.close();
  }
  catch(IOException e) {
  }
}
```

数を並べ替える直後の時間を計測

ファイルの役割

- BubbleSort.java
  - 「BubbleSort.txt」に、並び替え前と後の数を保存
- QuickSort.java
  - 「QuickSort.txt」に、並び替え前と後の数を保存

やってみよう!(1)

- ユークリッドの互助法のプログラム
  - 2つの数を標準入力で
- ATMの「引き出し」と「預け入れ」の機能のプログラム
  - 預金残高と暗証番号は、あらかじめプログラム中に書いておく
  - カードに関する処理はなしでOK
  - 出金処理と入金処理は、標準入力で処理
    - 出金: 「xx円出金し、yy円残っています。」と表示
    - 入金: 入金する金額を標準入力で入力

やってみよう!(2)

- 「アルゴロジック」という教材に挑戦してみよう
  - *http://home.jeita.or.jp/is/highschool/algo/index.html*
    - アルゴリズムの考え方を身につけるのに良い教材

やってみよう!(3)

- 配ったカードを、順番をバラバラにして…
  1. バブルソートで昇順に並べ替え(練習)
  2. バブルソートで昇順に並べ替え、かかった時間を計測(本番)
  3. 自分のやりやすい方法で昇順に並べ替え、かかった時間を計測
  4. 2. と3. の時間を比較(どちらが時間がかかっているか??)

やってみよう!(4)

- 授業のページから2つののファイルをダウンロードし、コンパイルと実行
  - BubbleSort.java
  - QuickSort.java
- 実行した結果、かかった時間が表示されるので、どちらが速いかを比較
- 2つのファイルの中に書いてある、「並べ替えをする数の個数」を
  いろいろな数に変更し、実行しなおして比較

第1回課題

- プログラム作成課題
  - 提出先: junko@cis.twcu.ac.jp
    - 第1回の授業で連絡した、連絡用のメールアドレスとは異なるので注意すること
    - このメールアドレスへの質問は受け付けないので注意すること
  - 提出期限: 11月2日(月) 18:00
  - 詳細は授業のページに
    - http://www.cis.twcu.ac.jp/~junko/Programming/Java2/

次回

- 10月27日: 出張のため休講