第６章．整列（ソート）のアルゴリズム

【学習のねらい】

① 整列（ソート）を行う基本的なアルゴリズム（バブルソート、選択ソート、挿入ソート）を学習し、その処理の流れを理解する。
② ３つのソートアルゴリズムの効率について考察する。
③ ソートアルゴリズムを応用したプログラムを学習する。

幾つかのデータを、値の大きい順や小さい順などのように、一定の基準に従って並べ替える操作を整列（ソート）と言います。ソートは応用範囲の広い処理であることから様々なアルゴリズムが考案されており、アルゴリズムの宝庫とも呼ばれていいます。本章では、その内、最も基本的あるいは代表的な３つのソートアルゴリズムを学習し、それらを幾つかの例に応用してみます。本章は、アルゴリズム学習のクライマックスとなる内容であり、特に将来、情報技術関係の道に進みたい学生にとっては、必須の内容です。

６－１　バブルソート

まず、最も基本的であり、（アルゴリズム関係の）どのような教科書にも出てくるバブルソートから学習を始めることにしましょう。バブルソートとは、隣り合う２つのデータ（の大小関係）を比較し、**並べたい順序になっていなければ入れ替える**、という操作を繰り返すことで整列を行う手法です。

具体的なケースで考えてみましょう。今、配列 A[0]～A[4]に数値（整数）が入力されているものとします。これを昇順に並べ替える場合を考えましょう。

＜ソート前＞

要素番号	0	1	2	3	4
配列 A	10	9	12	2	5

⇒

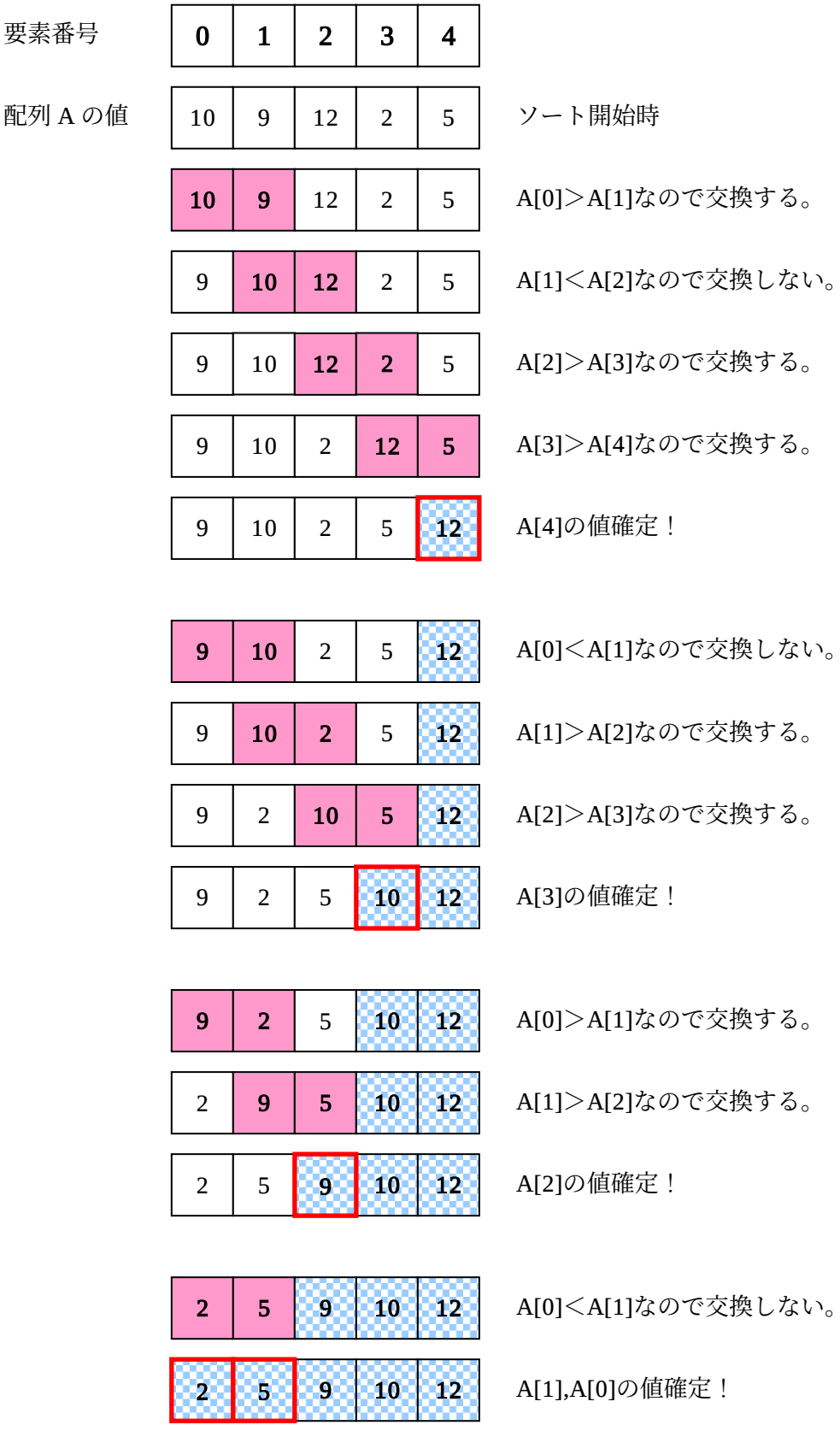
＜ソート後＞

0	1	2	3	4
2	5	9	10	12

ここに、データを小さい方から順に並べる場合を**昇順**と呼び、逆に大きい順に並べる場合を**降順**と言います。例えば「1,2,3,4,5」は昇順であり、「5,4,3,2,1」は降順になります。

次ページに、上例（のデータをバブルソートで昇順に並べ替えた場合）の処理の流れをまとめておきます。１ステップずつじっくりと確認して行って下さい。

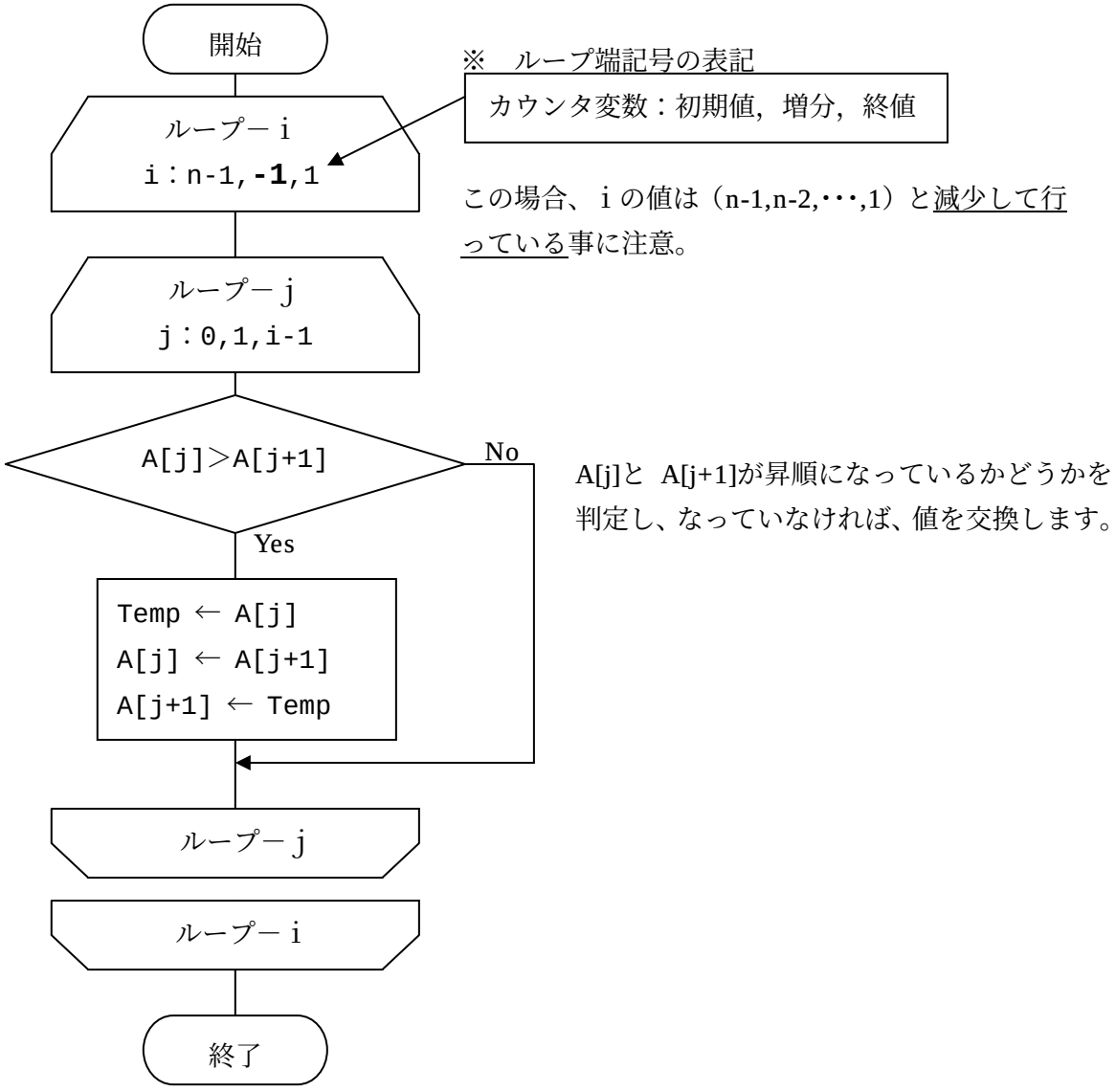
＜バブルソートの処理の流れ＞



ソート完了！

上の処理を、流れ図で表すと次のようになります。ただし、以下では配列の大きさをｎとして、配列 A[0]～A[n-1]に適当な値が入力されているものとしています。

＜バブルソート（昇順）＞



【基礎課題 6-1】

HP の該当部分から、ある科目の得点が記録されたファイル「tokuten.txt」をダウンロードし、これまで同様、フォルダ「IOFile」にコピーして下さい。

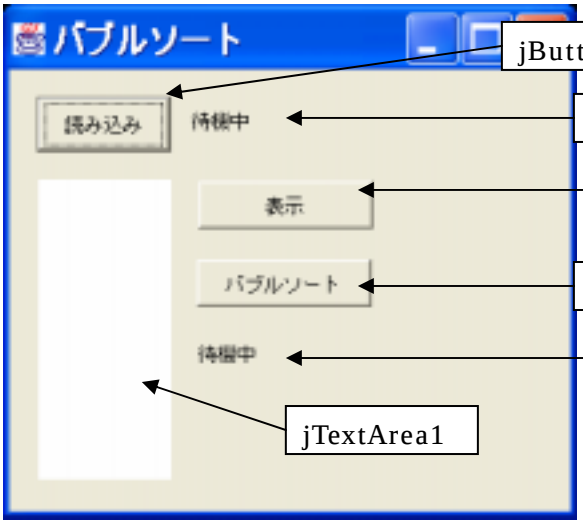
＜tokuten.txt＞

55
63
39
87
48
70
35
77
59
44

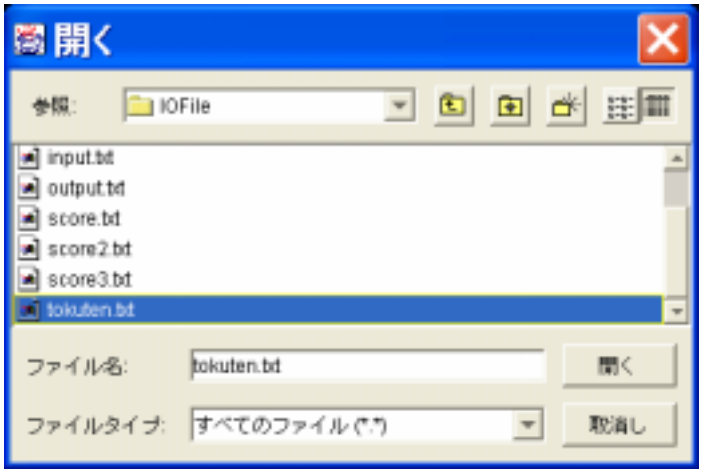
バブルソートを適用する練習として、このファイルから得点を読み込み、昇順にソートするプログラムを作成しましょう。

作成するプログラムの動作内容は次の通りです。

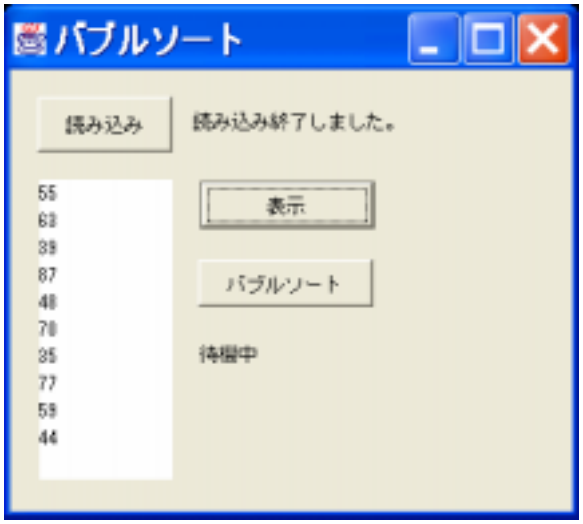
プログラムを起動すると、次のような画面が現れます。



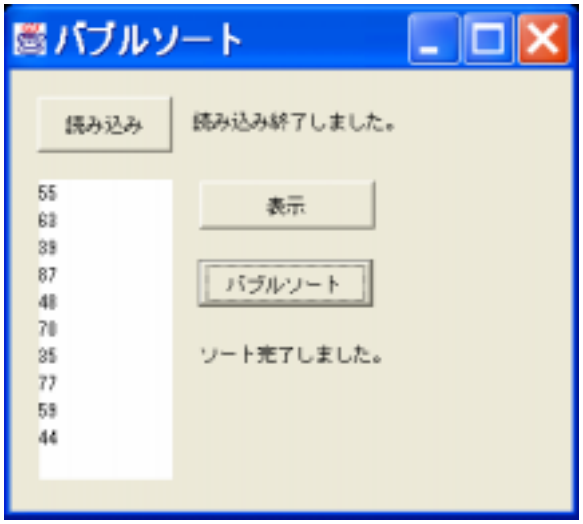
※　入力ファイル指定のダイアログを用いるので、UI フォルダに jFileChooser コンポーネントを追加しておいて下さい。



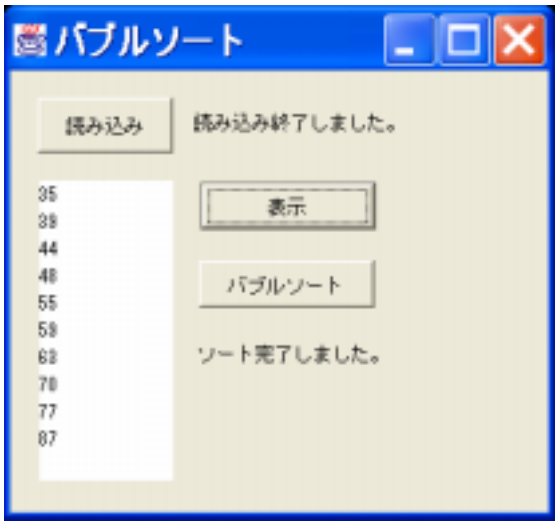
ここで、［読み込み］ボタンをクリックします。そして現れたダイアログボックスで「tokuten.txt」を指定すると、入力データの読み込みが完了します。



ここで、［表示］ボタンをクリックすると、今読み込んだ得点が（そのまま）表示されます。



次に［バブルソート］ボタンをクリックすると、バブルソートによる昇順ソートが完了します（データのソートを行うだけなので、jTextArea1 に表示された内容はまだ変わりません）。



続いて［表示］ボタンをクリックすると、昇順にソートされた得点リストが表示されます。

それでは、作成にとりかかりましょう。以下の要領でプログラムを作成して下さい。

＜プログラムの作成＞

1．まずファイルを読み込むので、いつもの通り波線部のインポート文を加えます。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
```

2．［読み込み］ボタンクリック時のイベントハンドラは次のようになります。これは、これまでと同じ要領です。

＜［読み込み］ボタン＞

```
int Tokuten[]= new int[100]; //得点を保管する配列
int Num; //データの個数

void jButtonRead_actionPerformed(ActionEvent e) {
  String Data;
  try {
    jFileChooser1.showOpenDialog(this);
    File FName=jFileChooser1.getSelectedFile();
    BufferedReader fin=new BufferedReader(new FileReader(FName));
    int i=0;
    while ((Data=fin.readLine())!=null) {
      Tokuten[i]=Integer.parseInt(Data);
      i++;
    }
    Num=i;
    jLabelRead.setText("読み込み終了しました。");
    fin.close();
  }
  catch (Exception em) {
    jLabelRead.setText("エラー発生："+em);
  }
}
```

3．［表示］ボタンクリック時のイベントハンドラは次のようになります。これは【基礎課題 3-7】と同じ要領です。　＜［表示］ボタン＞

```
  void jButtonDisplay_actionPerformed(ActionEvent e) {
    String Data="";
    for (int i=0;i<Num;i++) {
      Data=Data+Tokuten[i]+"¥n";
    }
    jTextArea1.setText(Data);
  }
```

４．［バブルソート］ボタンクリック時のイベントハンドラは次のようになります。空欄を埋めてプログラムを完成させて下さい。p.89 の流れ図を参照しながら考えれば分かるはずです。

＜［バブルソート］ボタン＞

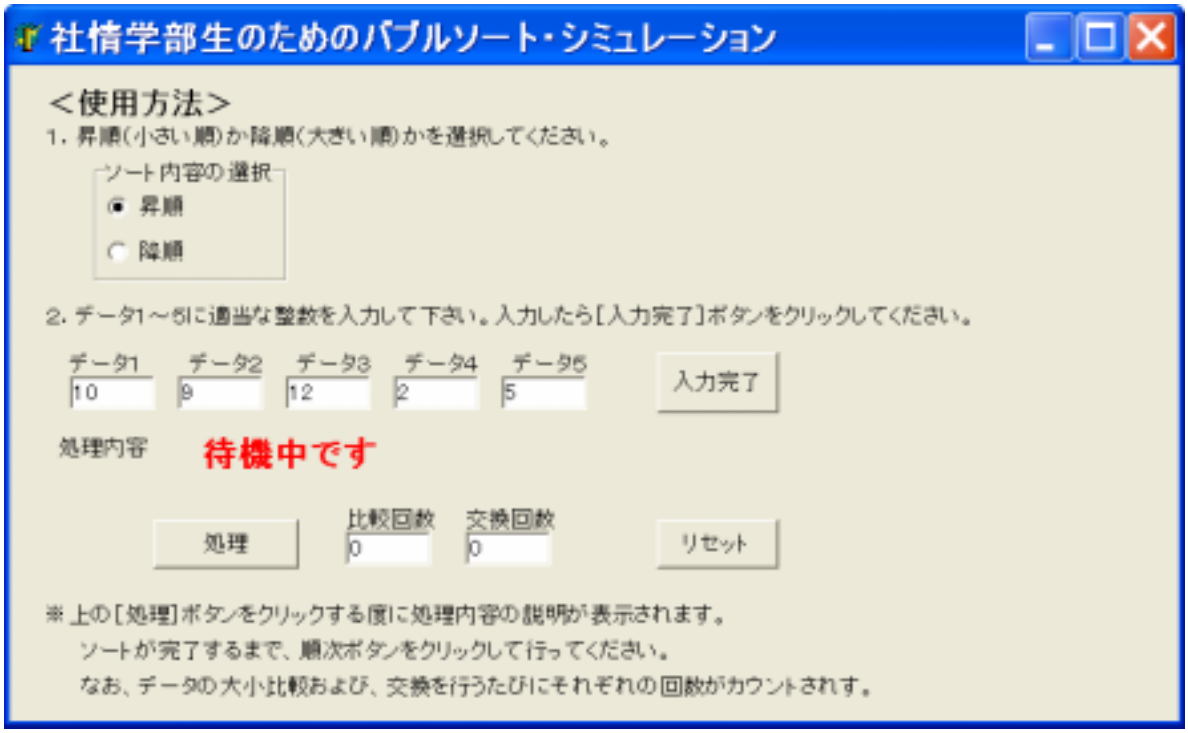
```
void jButtonBubbleSort_actionPerformed(ActionEvent e) {
  int Temp;
  for (int i=Num-1;        ;i--) {

    for (int j=0;          ;j++) {

      if(Tokuten[j]>Tokuten[j+1]) {
        Temp=Tokuten[j];
        Tokuten[j]=Tokuten[j+1];
        Tokuten[j+1]=Temp;
      }
    }
  }
  jLabelSort.setText("ソート完了しました。");
}
```

作成したら実行し、動作を確認してください。

＜　バブルソート・シミュレーションプログラム　＞

バブルソートの処理を（視覚的に）確認できるデモプログラムを受講生用に作成しました。HP の該当部に、「BubbleSort.exe」の名前で掲載しています。このプログラムをダウンロードして、処理の流れを今一度確認してください。使い方は、次の通りです。

プログラムを起動すると以下のような画面が現れるので、ソート内容が昇順か、降順かを選択し、５つのデータ欄に適当な数値を代入します。



数値入力後、［入力完了］ボタンをクリックすると準備完了です。あとは、［処理］ボタンをクリックする毎に、処理内容が表示され、変数の交換等の処理が行われます。また、２つの変数の大小比較や交換（入れ替え）を行う毎に、その回数がカウントされます。

なお、ソート完了後［リセット］ボタンをクリックすると、また最初からやり直すことができます。各自、何セットか実行し、処理の流れをじっくりと確認してください。

【基礎課題 6-2】

バブルソートの場合、（隣り合う）データの比較を行う回数は、データ数によって決まっています。データ数が 5 個の場合は、比較回数は幾つになるでしょうか？また、最大交換回数は幾つでしょうか？各自、アルゴリズムに従って１ステップずつ処理の流れをトレースしてみて下さい。上の「BubbleSort.exe」を用いて確かめても結構です。

（データ数５個の場合の）比較回数	（データ数 5 個の場合の）最大交換回数

※　例えば降順に並んだ {5,4,3,2,1} を昇順にソートするとき、データの交換回数は最大になります。

６－２　選択ソート

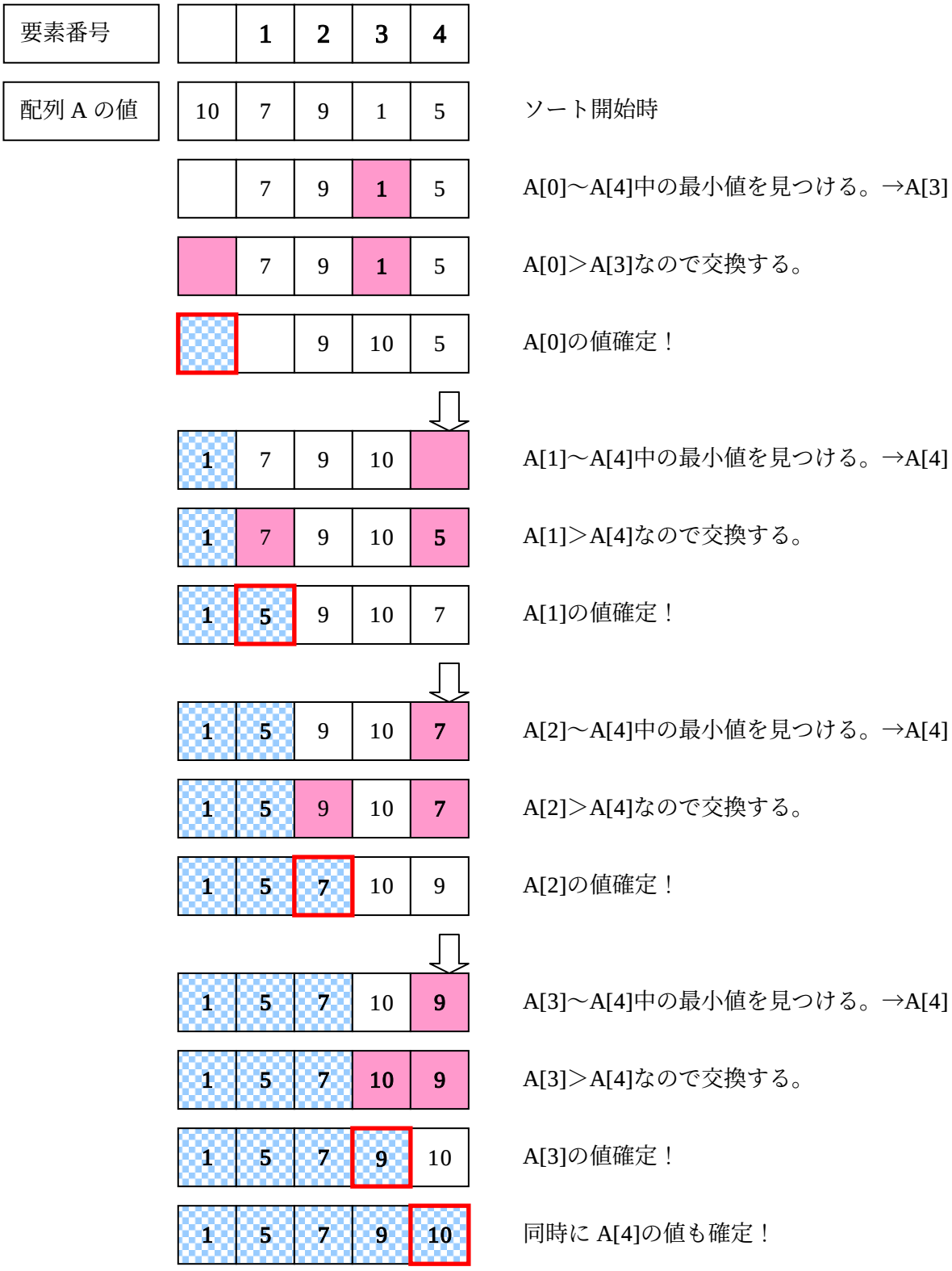
前節と同じく、配列 A[0]～A[4]に入力された数値（整数）を昇順に並べ替える場合を考えましょう。選択ソートとは、

① ５つの中から最小値を探し出し、先頭（1 番目の）データと交換する　→　1 番目のデータ確定。
② 次に、残った 4 つの中から最小値を探し出し、それを 2 番目のデータと交換する。 → 2 番目のデータ確定。

・・・

という操作を繰り返すものです。つまり、データの中から最小値あるいは最大値を選択し、それを順次データ（の未整列部分）の先頭に移動させる、という処理を繰り返すことで、整列を実現しているわけです。これが、選択ソートのアルゴリズムです。前節同様、次ページに具体的な処理の流れをまとめておきます。一つ一つの処理の流れを確認して行ってください。

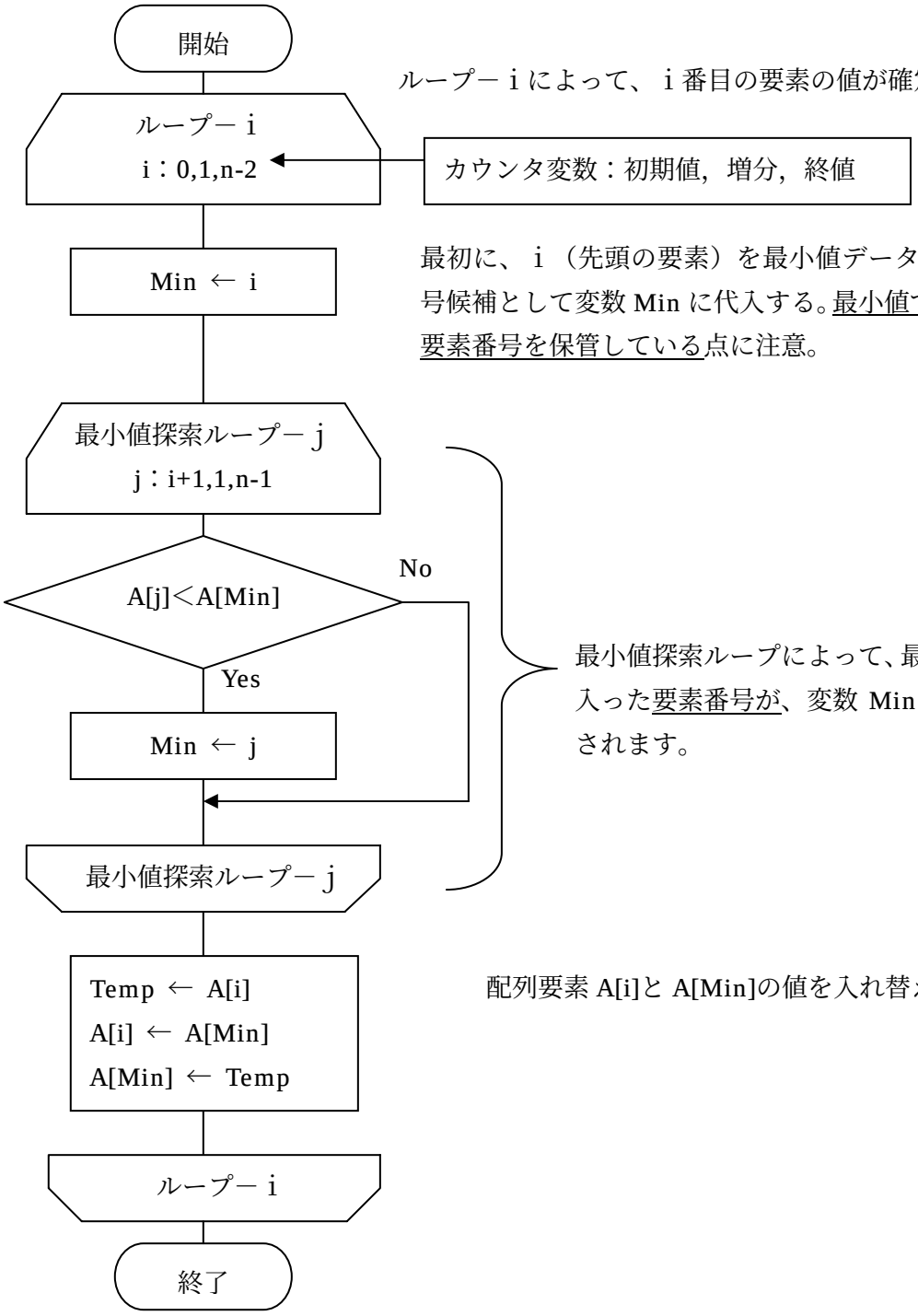
＜選択ソートの処理の流れ＞



ソート完了！

上の処理を流れ図の形に整理してみましょう。今、大きさｎの配列 A[0]～A[n-1]に、整数が入力されているものとします。この配列要素を昇順に並べ替える場合の流れ図は次のようになります。

＜選択ソート（昇順）＞



ループー i によって、 i 番目の要素の値が確定します。

最初に、 i （先頭の要素）を最小値データの要素番号候補として変数 Min に代入する。最小値ではなく、要素番号を保管している点に注意。

最小値探索ループによって、最小値の入った要素番号が、変数 Min に格納されます。

配列要素 A[i]と A[Min]の値を入れ替えます。

【基礎課題 6-3】

アルゴリズムは”書く”ことによって頭に刻みこまれます。そこで流れ図を書く練習をしましょう。前頁の流れ図を参考にして、配列要素 A[0]～A[n-1]を選択ソート法で降順に並べ替える流れ図を、以下に記述してください。

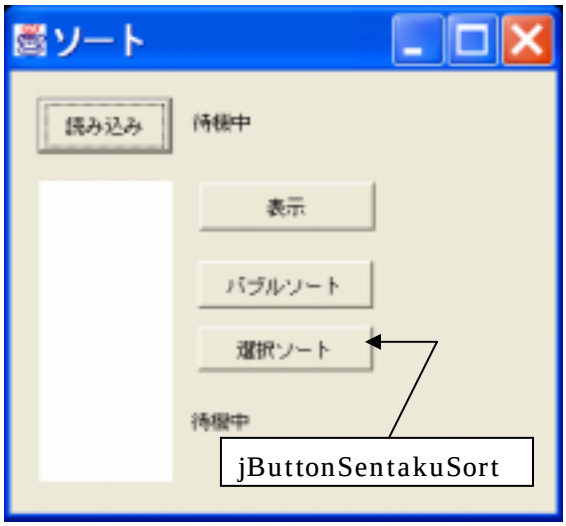
開始

終了

【基礎課題 6-4】

【基礎課題 6-2】と同様に、「tokuten.txt」から得点を読み込み、昇順にソートするプログラムを作成しましょう。ただし、今度は、選択ソートを用います。【基礎課題 6-2】のプログラムに次のように［選択ソート］ボタンを追加して下さい。

下の空欄を埋めて［選択ソート］ボタンのイベントハンドラを完成させてください。




```
void jButtonSentakuSort_actionPerformed(ActionEvent e) {
  int Temp,Min;
  for (int i=0;i<=Num-2;i++) {
    Min=i;
    for (int j=i+1; j<=Num-1;j++) {
      if(Tokuten[j]<                ) {
        Min=j;
      }
    }
    Temp=Tokuten[i];
    Tokuten[i]=Tokuten[Min];
                    =  Temp;

  }
  jLabelSort.setText("ソート完了しました。");
}
```

作成したら実行し、動作を確認して下さい。

【基礎課題 6-5】

前節と同じく、選択ソートの処理の流れを観察できるプログラムを HP の該当部に、「SentakuSort.exe」の名前で掲載しています。このプログラムをダウンロードして、適当なデータを入力することにより、処理の流れを視覚的に確認してください。

選択ソートにおいても、ソートに必要な比較回数は、入力データに関わらず一定です。入力データ数が５つの場合、比較回数は幾つかを、「SentakuSort.exe」を用いて確認してください。

比較回数＿＿＿＿＿＿＿＿

６－３　挿入ソート

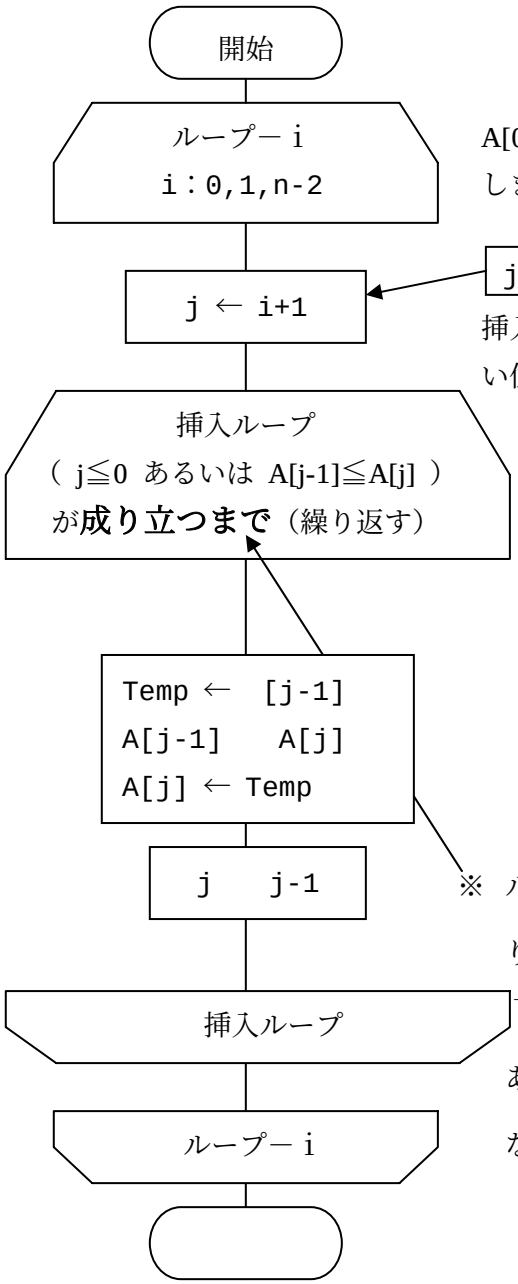
本章で学習する最後のソート法として挿入ソートを取り上げましょう。これは、部分的に整列されたデータの場合に威力を発揮するソート法です。ここまで来ると皆もソートアルゴリズムに慣れてきた頃だと思いますので、前置きなしで、以下に処理の流れをまとめておきます。流れを確認して下さい。

＜挿入ソートの処理の流れ＞

要素番号		1	2	3	4	
配列 A の値		6	2	9	7	開始時：A[0]までがソート済みと考える。
	10	6	2	9	7	A[1]を整列済み部分に追加する。
	6	10	2	9	7	A[1]を A[0]～A[1]中の正しい位置に送る。→A[0]>A[1]なので入れ換える。
			2	9	7	A[0]～A[1]が整列済み。
	6	10	2	9	7	A[2]を整列済み部分に追加する。
		6	10	9	7	A[2]を A[0]～A[2]中の正しい位置に送る。→A[2]の値を A[1]、A[0]の値と順に比較し、降順なら入れ換える、という操作を繰り返す。
	2	6	10	9	7	A[0]～A[2]が整列済み。
	2	6	10	9	7	A[3]を整列済み部分に追加する。
	2	6	9	10	7	A[3]を A[0]～A[3]中の正しい位置に送る。→A[3]の値を A[2]、A[1]の値と順に比較して行き、A[1]<9 なのでそこで挿入位置確定。
		6	9	10	7	
	2	6	9	10	7	A[4]を整列済み部分に追加する。
	2	6	7	9	10	A[4]を A[0]～A[4]中の正しい位置に送る。
	2	6	7	9	10	A[0]～A[4]が整列済み。→ソート完了！

これまで同様、上の処理を流れ図で表すと次のようになります。ただし以下では配列の大きさをｎとして、配列 A[0]～A[n-1]に適当な値が入力されているものとしています。

＜挿入ソート（昇順）＞



A[0]～A[i]までが整列済みであるとしてスタートします。

ｊは挿入位置を意味します。

挿入ループによって、A[i+1]を A[0]～A[i+1]中の正しい位置に送ります。

終了条件に注意して下さい。A[j-1]≦A[j]が成立する場合は、その時点で A[j]の値が確定します。なぜなら、A[j-1]より前方はすでに昇順に整列済みなので、全て A[j]より小さいか等しいからです。そのため、これ以降は大小関係を比較する必要がないので、ループを終了します。

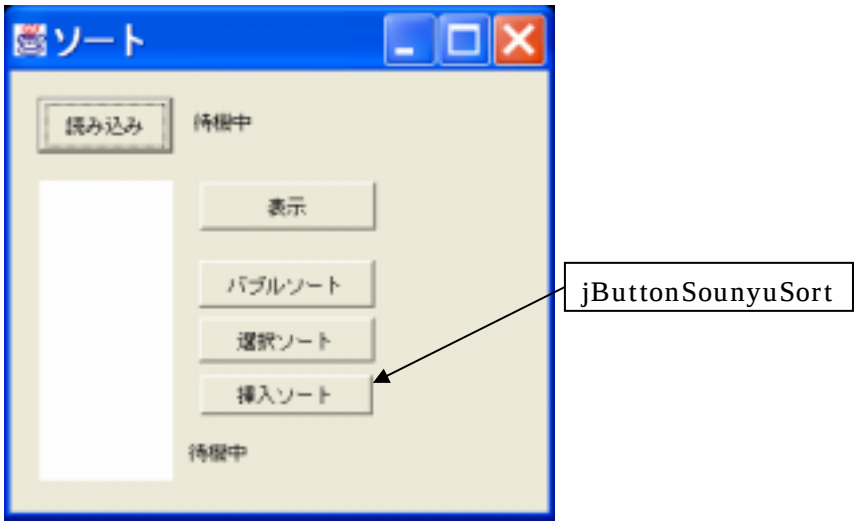
※ ループ端記号の約束では、ある条件が成り立つ**まで**繰り返す、という表記になっています。
一方、プログラミング言語で while 文を用いる場合は、ある条件が成り立っている**間**繰り返す、という書き方になります。
したがって、上の条件は

```
while（ j≧1 かつ A[j-1]＞A[j] ）
```

という条件の書き方になります。
次ページの【基礎課題 6-6】を考える際注意してください。

【基礎課題 6-6】

【基礎課題 6-2】および【基礎課題 6-4】と同様に、「tokuten.txt」から得点を読み込み、昇順にソートするプログラムを作成しましょう。もちろん、今度は挿入ソートを用います。【基礎課題 6-4】のプログラムに、次のように［挿入ソート］ボタンを追加して下さい。



下の空欄を埋めて［挿入ソート］ボタンのイベントハンドラを完成させてください。

```
  void jButtonSounyuSort_actionPerformed(ActionEvent e) {
    int Temp;
    for (int i=0;i<=Num-2;i++) {
      int j=i+1;
      while (                                  ) {

        Temp=Tokuten[j-1];
        Tokuten[j-1]=Tokuten[j];
        Tokuten[j]=Temp;
        j--;
      }
    }
     jLabelSort.setText("ソート完了しました。");
}
```

作成後、プログラムを実行し、動作を確認してください。

【基礎課題 6-7】

前節までと同じく、挿入ソートの処理の流れを観察できるプログラムを HP の該当部に、「SounyuSort.exe」の名前で掲載しています。このプログラムをダウンロードしてください。そしてこれを用いて処理の流れを確認して下さい。

挿入ソートとバブルソートに要する比較回数および交換回数には、次のような大小関係があります。実際にシミュレーションプログラムで確認して、空欄に入る記号を解答群から選んでください（よく分からない人は、先に 6-4 節を読んで下さい)。

＜解答群＞

比較回数（挿入ソート）		比較回数（バブルソート）	＜　＞ ≦　≧
交換回数（挿入ソート）		交換回数（バブルソート）	

６－４　アルゴリズムの効率

本章で学んだ３つのソートアルゴリズムは、いずれを使っても問題なくソートを行うことができます。しかし、その効率には違いがあります。アルゴリズムの効率については、本講義ではその詳細は扱いませんが、ソート処理に関して言えば、それに要する比較回数および交換回数が少ないほど効率が良い、と理解しておいてください。本章で用意した、ソートシミュレーションプログラムを用いて、色々な入力データについて動作を確かめてみると分かるのですが、以下の点が知られています。

１．バブルソートと、選択ソートの比較回数は、同じです。データ数が５つの場合は 10 となっていたと思います。種明かしをすると、これは、５つの中から任意の２つの組み合わせを選ぶときの数です。データ数がｎ個であれば、n(n-1)/2 となります。つまり、この２つのソートでは、全てのペアについて大小関係を比較している訳です。一方、挿入ソートの場合は、すでに整列済みの部分については比較する必要がないので、n(n-1)/2 以下になります。

２．交換回数については、一般に選択ソートが最も少なくなります。一方、バブルソートと挿入ソートの交換回数は等しくなります。

以上を整理すると次のようになります。

- ◆ 比較回数：　（バブルソート）＝（選択ソート）≧ （挿入ソート）
- ◆ 交換回数：　（バブルソート）＝（挿入ソート）≧ （選択ソート）

これを見ると、バブルソートは最も効率が悪い、ということになります。では、一般に挿入ソートと選択ソートではどちらの効率が良いのでしょうか？それは、ソート対象となるデータに依存することになりますが、一般には、部分的に整列したデータが含まれることが多いので、挿入ソートが最も効率が良くなることが多いようです。

ここで、「どれを用いても正しくソートできるのなら、どうして効率などにこだわるの？」と疑問に思う人がいるかもしれません。もっともな疑問ですが、ソートプログラムが使われている現場では、数万個程度のデータを扱うことが少なくありません。例えば、センター入試の全受験生の成績をソートする場合などを考えてみたらどうでしょう（今年度の場合志願者は約 59 万人です！）。このような場合、ソートアルゴリズムの効率が決定的にモノを言うのです。

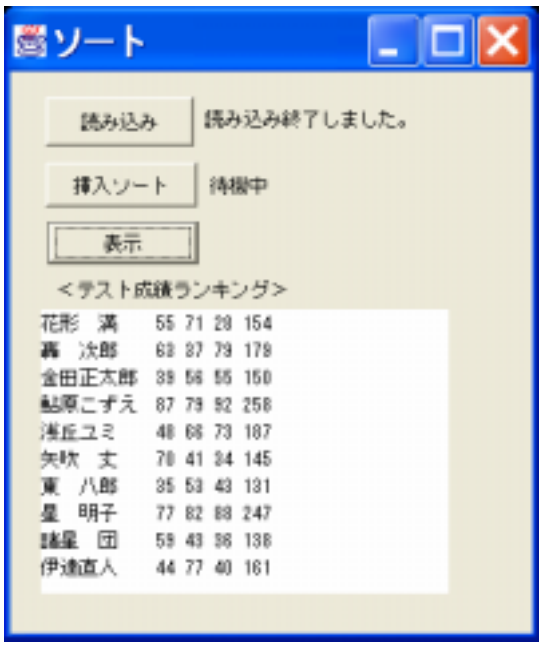
以上はやや専門的な内容なので、参考までに知っておいてくれれば十分です。しかし、将来情報処理関係の技術者を目指す人にとっては、”常識”として要求される知識です。

６－５　応用問題

【応用課題 6-A】

【応用課題 5-B】を発展させて、今度は「score3.txt」のデータを読み込んで、下の様に 3 科目の合計点順に（降順に）ソートして表示させるプログラムを作成してください。ソート法としては挿入ソートを用いてください。

入力ファイルを指定しデータの読み込み完了後、［表示］ボタンをクリックすると、受験生の成績が表示される。



［挿入］ボタンのクリック後、再度［表示］ボタンをクリックすると、合計点の高い順にソートされて表示される。