課題２：整列化（ソート）の計算

講義ノート

例題 2.2（教科書より）整列化

与えられた n 個の数値データを小さい順に並び替えよ.

教科書では，この問題に対して次の 2 つのアルゴリズムを与えている.

```
1  program sortS (float a[1..n])
2     以下を i ← 1 ∼ n − 1 に対して実行 {
3        a0 ← a[i] ∼ a[n] の最小値
4        i0 ← 配列 a 中の a0 の場所
5        a[i] ↔ a[i0]
6     }
7     return(a)
8  program end.
```

プログラム 2.3（教科書より）： 選択整列法

```
1  program sortM (float a[1..n])
2     配列 a を前半部 b1 と後半部 b2 に分ける
3     b1 ← sortM(b1)
4     b2 ← sortM(b2)
5     b ← merge(b1,b2)
6     return(b)
7  program end.
```

プログラム 2.5（教科書より）： マージソート法

```
 1  program merge (float b1[1..m1], b2[1..m2])
 2     i ← 1;   j ← 1;
 3     k ← 1;
 4     以下を繰り返す {
 5        if (b1[i] ≤ b2[i]) {
 6           b[k] ← b1[i];
 7           k ← k + 1;   i ← i + 1;
 8           if (配列 b1 が終わった) break;
 9        } else {
10           b[k] ← b2[j];
11           k ← k + 1;   j ← j + 1;
12           if (配列 b2 が終わった) break;
13        }
14     }
15     b[k..] ← b1 または b2 の残り;
14     return(b)
15  program end.
```

プログラム 2.7（教科書より）: マージ（教科書のとは若干違う計算方法）

講義（演習）では，前者を ssort，後者を msort と呼ぶことにする．また，$t_{\rm s}(n), t_{\rm m}(n)$ で各々の最悪時間計算量（比較回数を基準とした）を表わすことにする．つまり，

$$\begin{aligned} t_{\rm s}(n) &= \text{ssort に } n \text{ 個の数を与えたときの最悪時比較回数} \\ t_{\rm m}(n) &= \text{msort に } n \text{ 個の数を与えたときの最悪時比較回数} \end{aligned}$$

とする．

実験ノート

実験課題（レポートの〆切は 12 月 5 日午後 4:00）

整列化を行う 4 つのプログラム (ssort, msort, xsort, ysort) に対し，（ほぼ）最悪時間計算量，（いろいろな状況のもとでの）平均時間計算量を求め，その比較回数，実際の計算速度を比較せよ．

(1) まず，苦手なデータについて考える．これは比較回数に基づく計算量で議論する．ssort, msort, xsort に対し（ysort はやらなくてもよい），相性の良い・悪いデータはあるかを調べる．（**注意：比較的小さなデータで実験した方がよい．**）

(2) 次に，いろいろな状況の元での「平均」を考える．単純な平均だけではなく，ある種の偏りを持ったデータ群の平均など，複数のデータの分布を考えて実験してみるとよい．これも比較回数に基づく計算量で議論する．（**注意：これも比較的小さなデータで実験した方がよい．**）

(3) 最後に計算時間について調べる．比較回数だけでは，msort の方が xsort よりすぐれているように思える．では，実際の計算時間ではどうだろうか？これは，比較的大きなデータまで使ってで実験すべき．（**注意：データサイズが大きくなると ssort は時間がかかるので，ssort はこの実験の対象外にすべき．**）

(4) オプショナル：ysort が最も苦手とするようなデータを実験から推定してみよう．

使用ソフト

実験のための次のソフト（コマンド）等は，ディレクトリ `~owatanab/pub/sort` に用意してある．

実験データ生成用

randnum（プログラムのソースは randnum.c）

[自分のプロンプト] `./randnum` *a b seed*

- *a*, *b* は $a \leq b$ となる整数．*seed* は乱数の初期値．適当な値を使う．
- $[a, b]$ に入る整数をランダムに生成する．

randfloat（プログラムのソースは randnum.c）

[自分のプロンプト] `./randfloat` *a b seed*

- *a*, *b* は $a \leq b$ となる整数．*seed* は乱数の初期値．適当な値を使う．
- $[a, b]$ に入る小数をランダムに生成する．

rands（プログラムのソースは rands.c）

[自分のプロンプト] `./rands` *a b seed n*

- *a*, *b* は $a \leq b$ となる整数．*seed* は乱数の初期値．適当な値を使う．*n* は生成したい数字の個数．
- $[a, b]$ に入る小数をランダムに *n* 個生成する．（同じ数が複数個出る場合もあるが，可能性はかなり低い．）

実験対象プログラム

msort, ssort, xsort, ysort, msortT, ssortT, xsortT, ysortT

[自分のプロンプト] `./msort` *n*

- ソートする数値の個数 *n*．
- 与えられる *n* 個のデータを整列化する．
- 計算中に行なわれた（データ同士の）比較回数も出力．

使い方の例

(1)
```
% ./msort 5
3            <--- 手で入力
2.5
8.91
0
-1.3
```

答えが出力される

(2)
```
% ./rands 0 1 1234 100 > data.100
% ./msort 100 < data.100
```

答えが出力される

シェルスクリプトについての補足

今回の実験では，必要なデータを自分で作成して欲しい．そのため，データを作成するのに必要最小限のコマンド randnum, randfloat しか用意しなかった．データファイルは，シェルスクリプトで作ることができる．たとえば，次のようにすれば，100 個の $[0,1]$ 区間に入る乱数からなるファイル data.100 が生成できる．

```
#!/bin/tcsh −f
@ x = 1
@ seed = 1234
echo > file1
while ( $x <= 100 )
   ./randfloat 0 1 $seed >> file1
   @ x = $x + 1
   @ seed = $seed + 1
end
```

プログラム J2r.sh： 乱数を 100 個生成する

ただし，残念ながらこの方法は遅い．そこで，多数の乱数を生成する場合を考え，n 個を高速に生成するコマンド rands も用意しておいた．多くデータを作る場合には，これを利用しても欲しい．

その他，便利なシェルスクリプトの記述法

（例１）
```
if ( $x <= $y ) then
   ···          ⇐ 条件が成り立つ時，実行される
   ···
endif
```

（例２）
```
while ( ··· )
   ···
   if ( ··· ) break        ⇐ 条件が成り立つ時，繰り返しから抜け出る
   @ ···
end
···
```

（例３）
```
foreach hensu ( ls file.* )
   ···          ⇐ ディレクトリ内の file.xxx という名のファイル
   ···             すべてに対し，以下を実行．各ファイル名は $hensu へ
end
```