SORT

N 個のデータ $x = \{x(i) \in X \mid i = 1, 2, \cdots, N\}$, を小さい順に並べなおすアルゴリズム．$N$ のことを **データの大きさ（サイズ）** と呼ぶ[1]．X はデータの集合（一般には半順序集合）だが，ここでは $X = \mathbf{R}$ の場合だけを扱う．

並べ直すとは，まず入力データを配列に代入した後に (配列も $x(1)$, $x(2)$, $\cdots$, $x(N)$ と書く),

(i) 任意の二つの変数の値の比較,

(ii) ある変数の値の別の変数への代入 (コピー)，特に，二つの変数の値の交換,

という操作だけを用いて，$\{x(i) \mid i = 1, 2, \cdots, N\} = \{y(i) \mid i = 1, 2, \cdots, N\}$, かつ，$y(1) \leq y(2) \leq \cdots y(N)$ を満たす出力データ (または配列) $x(1)$, $x(2)$, $\cdots$, $x(N)$ を得ることである．

§0.1. 一般論

§0.1.1. 目安となる量

回数． 理論的な研究のしやすい量として考えられるもの：

(i) 可能な初期配列のうちで最悪の比較回数.（初期条件によらないこと（安定性）の目安.）

(ii) $N!$ 個の可能な初期配列の集合上に一様確率測度を入れたときの終了までの比較回数の期待値（平均比較回数）.（アルゴリズムの平均的な性能，速さ，の目安.）

(iii) 最悪の交換回数.

(iv) 初期配列に一様確率測度を入れたときの交換回数の期待値.

ここでは比較回数を問題にするが，交換回数も目安になる．交換 $(swap(a, b))$ は計算機のハード（メモリ）レベルでは 3 回の代入 $c \leftarrow a$, $a \leftarrow b$, $b \leftarrow c$ として実行されるので代入の回数を目安にするのが適切.

問． 使用環境に応じて入れ替え回数と比較回数のどちらが目安として重要か（あるいは同程度か）は変わりうる．その判基準は何か？

記憶容量． 入れ替えを行うために必要な補助的な記憶場所（作業領域）が限られている場合には記憶容量も性能の目安になる．個々のデータが大きな記憶場所を占める場合（単なる数字ではなく，数字をキーとする複数フィールドのデータなど）には実際の入れ替えをしない方法が有利な場合もある．

§0.1.2. Uniform bound

定理 1． 最悪の回数 $k_{max}(N)$ は

$$\liminf_{N\to\infty} \frac{1}{N \log_2 N} k_{max}(N) \geq 1$$

を満たす．即ち，$k_{max}(N)$ は N が大きいときほぼ $N \log_2 N$ 以上である．

証明. 整列したデータ $x(i) = i$, $i = 1, 2, \cdots, N$, から出発して k 回の操作を行うことを許すと，1 回毎に操作を行うか行わないかで最大 2 倍の異なる配列が生成できるので 2^k 種類の配列を生成できる．ソートはこの逆操作である． N 個のデータがあるとき，$N!$ 種類の配列をソートしなければならないから， $2^{k_{max}} \geq N!$ でないと最悪の場合をカバーできない． Stirling の公式

$$\lim_{N\to\infty} \frac{N!}{N^N e^{-N} \sqrt{2\pi N}} = 1$$

を用いれば定理を得る． □

[1] 大きさといってもデータ自体の数値の大小ではないことに注意

定理 1 の気持ち． $\lim_{N\to\infty} \frac{1}{N\log_2 N} k_{max}(N) = 1$ を満たすアルゴリズムがあれば，k_{max} に関する漸近的に最善のアルゴリズム[2]である．

定理 2． 平均回数 $k_{ave}(N)$ は

$$\liminf_{N\to\infty} \frac{1}{N\log_2 N} k_{ave}(N) \geq 1$$

を満たす．

証明．定理 1 で考察した状況の逆操作が最も効率がよい．即ち k_{ave} の下限を与える．定理 1 の k_{max} を $k_m = k_{max}(N)$ と書くと，

$$2^{k_m} \geq N! > 2^{k_m} - 1,$$
$$N!\, k_{ave}(N) \geq \sum_{k=1}^{k_m-1} k\, 2^{k-1} + k_m\, (N! - 2^{k_m-1}) = k_m\, N! - 2^{k_m} + 1\,.$$

これより，$k_{ave}(N) > k_m(N) - 1$ を得る． □

定理 2 の気持ち． $\lim_{N\to\infty} \frac{1}{N\log_2 N} k_{ave}(N) = 1$ を満たすアルゴリズムがあれば，k_{ave} に関する漸近的に最善のアルゴリズムの一つである．

補題 3． n が自然数のとき

$$\sum_{k=1}^{n} k\, 2^{k-1} \quad = \quad (n-1)\, 2^n + 1\,,$$
$$\sum_{k=1}^{n} k\, 2^{-k} \quad = \quad 2 - (n+2)\, 2^{-n}\,.$$

証明．前半は

$$\mathrm{lhs} = \sum_{k=1}^{n} k\, 2^{k-1} = \sum_{k=1}^{n} k\, 2^k - \sum_{k=1}^{n} k\, 2^{k-1} = \sum_{k=1}^{n} k\, 2^k - \sum_{k=0}^{n-1} (k+1)\, 2^k = n\, 2^n - \sum_{k=0}^{n-1} 2^k = \mathrm{rhs}\,.$$

同様に後半は

$$\mathrm{lhs} = \sum_{k=1}^{n} k\, 2^{-k+1} = \sum_{k=1}^{n} k\, 2^{-k+1} - \sum_{k=1}^{n} k\, 2^{-k} = \sum_{k=0}^{n-1} 2^{-k} - n\, 2^{-n} = \mathrm{rhs}\,.$$

□

§0.2. 主なソートアルゴリズム

一番速いアルゴリズム（複数ある時はどれか一つ）だけを使えばいいように見えるが，

(i) データサイズ N が小さいときはすぐ終了するので， N が大きいとき，即ち漸近的な評価に主な興味がある．逆に言えば，データの大きさが小さい場合は漸近的に速いプログラムより，簡単なプログラムがかえって速く，またプログラムの間違いが少ない，

(ii) 実用上，使用目的によって初期配列が一定の傾向がある場合には初期配列として起こりやすいものに重みをつけた確率測度を入れて期待値を考えるべきであり，一般的には遅いアルゴリズムであっても有利になることがある．例えば，部分的に並んでいるデータが多いのが普通，

[2]複数あるかもしれない．

(iii) 値域の集合が限られているときも速い方法がある． $X = S_N(1, 2, \cdots, N)$ （$(1, 2, \cdots, N)$ の置換）のときは x を自然数上の写像と見てその逆写像 y を求めるのが速い（ハッシュソートは値域の集合を逆写像の計算が速い集合にいったん変換するハッシュ関数を用意する方法である），

(iv) 使用可能記憶容量が少ないときは遅くても作業領域が少なくてすむアルゴリズムが必要である，

(v) データが全順序集合（実数）のキーを持たない（半順序集合である）場合にソートしたい（この場合，一列に並べる「自然な」方法は複数あり得るので，解が一意でない問題を解くことになる），

などの理由によっていくつものアルゴリズムが提案されている．これらの問題の検討は省略するが，$X = \mathbf{R}$ の場合の比較的普遍的なソートを列記する．

名前	平均的比較回数	最悪のケース	特徴・制限・利点等
基本選択法	$N(N-1)/2$	$N(N-1)/2$	
基本交換法[3]	$N(N-1)/2$	$N(N-1)/2$	
基本挿入法	$N^2/4\,(1+o(1))$	$N(N-1)/2$	
heap sort	$N\log N\,(1+o(1))$		二分木 (再帰的手続き)．選択法．小さい N では不利
quick sort	$N\log N\,(1+o(1))$	N^2	再帰的手続き．交換法
merge sort	$N\log N\,(1+o(1))$	同程度?	再帰的手続き．挿入法的．作業領域大 ($2N$?)
Shell sort[4]	$O(N^{1.5})$?	同程度?	挿入法

最後の３つは遅いが自明な方法．最初の４つは自明な方法を改良したものといえる．速くするためには複雑な改良 (§0.2.4) が必要であり，それ故にアルゴリズムという概念が重要になる．

アルゴリズムという概念がなぜ重視されるか？

殆どの場合，理論的に無駄のない（速い）プログラムを書くには直感のききにくい方法（アルゴリズム）を工夫しないといけない．

基本選択法． $x(k),\ 1 \le k \le n-1$ $(n = 1, 2, \cdots, N)$ が整列済みの最初の $n-1$ 個の小さいデータになっているとき，残りのデータを順番 $(x(k),\ k = n+1, \cdots, N)$ に配列上の n 番目のデータ（最初は初期値 $x(n)$）と比較して，小さいときは交換することで n 番目のデータまで終了する．(最小値候補用作業領域 xm を用意して比較・代入したほうが代入回数換算の交換回数は少なくなるが，こだわらないことにする．)

比較回数は常に $N(N-1)/2$.

基本交換法（バブルソート）． $x(k),\ n+1 \le k \le N$ $(n = 1, 2, \cdots, N-1)$ が整列済みの最後の $N-n$ 個の大きいデータになっているとき，順番 $(k = 1, \cdots, n-1)$ に $x(k)$ と $x(k+1)$ を比較して大きいほうを右に持っていく（交換が起きると大きい方のデータは続けて右隣と比較するので非常に大きなデータはバブル（泡）のように右に続けて移動する）と n 番目のデータまで整列が終了する．

比較回数は常に $N(N-1)/2$.

基本挿入法． $x(k),\ 1 \le k \le n-1$ $(n = 1, 2, \cdots, N)$ が整列 $(i < j \longrightarrow x(i) \le x(j))$ している（小さい方から $n-1$ 個とは限らない）とき， $x(n)$ をこれらと小さい方から順に比較して，最初に大きいのが出てきたときにその直前に挿入すると n 番目のデータまで整列する．「順に比較して挿入する」とは，

$w \leftarrow x(n)$
$\boxed{k:\ n-1 \sim 1}$
| $x(k) \gtrless w$
| YES $\Longrightarrow x(k+1) \leftarrow x(k)$

[3] シェル (Shell) のみ人名．
[4] 通常バブルソートと呼ぶ．

NO $x(k+1) \leftarrow w \implies$ next n

交換回数も比較回数も最悪で $N(N-1)/2$.

命題 4. 交換回数も比較回数も平均は約 $N(N-1)/4$.

証明. データ $x(i)$ の総移動回数は自分より左 $(j<i)$ にある自分より大きい $(x(i)<x(j))$ データと右にある小さいデータの和．これを i について加えて（交換は二つのデータの移動だから）2で割れば交換回数を得る．

まず $i \neq j$ ならば $\mathrm{Prob}[\, x(j) > x(i)\,] = 1/2$ であることに注意．これは二つのデータは一方が大きいか他方が大きいかどっちかで，どのデータもどの値を取るかは平等だから．(具体的に定義にしたがってやってもできる．) 期待値は加法性を使えば,

$$\tfrac{1}{2}\sum_i \Big(\sum_{j<i} \mathrm{Prob}[\, x(j) > x(i)\,] + \sum_{j>i} \mathrm{Prob}[\, x(j) < x(i)\,]\Big)$$
$$= \sum_{i,j<i} \mathrm{Prob}[\, x(j) > x(i)\,] = N(N-1)/2/2\,.$$

□

$0 \leq$ 比較回数 $-$ 交換回数 $\leq N-1$ なので，平均比較回数の漸近形は平均交換回数のそれに等しい．

問.

(i) 表の空欄や曖昧な欄を検討して表を完成せよ.

(ii) 基本的な３つの方法について，説明を参考にしてアルゴリズムをフローチャートで書け.

(iii) 各方法について交換回数の最悪の場合と平均交換回数を求めよ.

§0.2.1. シェルソート

挿入操作と再帰的手続き．短いプログラムで基本法より速い(最善の漸近形ではない)．In place (必要な補助記憶容量が $O(1)$).

$h_1 > h_2 > \cdots > h_e = 1 > 0$ なる自然数の有限列を固定する．通常 $h_1 = [N/2]$, $h_{k+1} = [h_k/2]$, $k = 1, 2, \cdots, e$, となるように選ぶ．$k = 1, 2, \cdots, e$, $i = 1, 2, \cdots, h_k - 1, h_k$, について，部分データセット $\{x(i+\ell h_k),\ \ell = 0, 1, 2, \cdots, [(N-i)/h_k]\}$ を基本挿入法でソートする.(小さい k における操作で，部分的に整列していることに注意．基本挿入法の最悪のケースは起こらない.)

第２種情報処理技術者試験の教科書には約 $O(N^{1.5})$ となっているが，よく分からない．基本挿入法で逆順のとき最悪だが，シェルソートで逆順，かつ，N が 2 のべき乗ならば $O(N \log N)$ だと思う.

§0.2.2. マージソート

極端に長くはないプログラムで理論的に最善の漸近的な速度を得る．但し in place ではない．再帰的手続きで，二つの部分列を merge するときに挿入操作を行う.

x を配列，$p \leq q < r$ を配列の添字とし，部分配列 $x[p, \cdots, q] = x(p), \cdots, x(q)$ と部分配列 $x[q+1, \cdots, r]$ が整列済みのとき，これらを整列して $x[p, \cdots, r]$ を整列する演算 (merge) を $\mathtt{MERGE}(x, p, q, r)$ で表す．$p \leq r$ に対して $x[p, \cdots, r]$ をマージソートするアルゴリズムを $\mathtt{MERGE-SORT}(x, p, r)$ とおくと,

$\boxed{\mathtt{MERGE-SORT}(x, p, r)}$

$p \lneq r$

YES $\implies q \leftarrow [(p+r)/2]$

$\texttt{MERGE}-\texttt{SORT}(x,p,q)$
$\texttt{MERGE}-\texttt{SORT}(x,q+1,r)$
$\texttt{MERGE}(x,p,q,r)$
NO $\Longrightarrow$ 終わり

および，補助配列 $y[1,\cdots,N]$ を用意して，

$\boxed{\texttt{MERGE}(x,p,q,r)}$
$j \leftarrow p,\ k \leftarrow q+1$
$\boxed{i:\ 1 \sim r-p+1}$
| $j > q$ ならば下記 YES の右へ飛ぶ
| $k > r$ ならば下記 NO の右へ飛ぶ
| $x(j) \lesseqgtr x(k)$
| YES $\Longrightarrow y(i) \leftarrow x(k),\ k \leftarrow k+1$
| NO $\Longrightarrow y(i) \leftarrow x(j),\ j \leftarrow j+1$
———

で定義される．

再帰的手続きを使わずに書くことはもちろんできるが，MERGE-SORT の２回の部分MERGE-SORT について flag を立てるなどめんどう．

作業ステップ数 (比較+交換) $T(N)$ は Cormen etal., p.14–15 によれば，データに依存しない定数で，$T(n) = 2T(n/2) + O(n)$ を満たす．$O(n) = n$ ならば，帰納法により $T(n) = n\log_2 n$ となる．

§0.2.3. クィックソート

短いプログラムで in place かつ理論的に最善の平均の漸近的な速度を得る．漸近形の係数は小さい (即ち，プログラムが簡単) ので，とりあげたソートの中では平均的に最も速い．最悪の場合は基本法並に遅いという欠点があるが，もっとも実用的な方法．交換法の改良型．

アイディア：$x[p,\cdots,r]$ を，$x[p,\cdots,q]$ と $x[q+1,\cdots,r]$ に分割する．但し，前者の要素は全て最初に与えられた $x(p)$ 以下に，後者は $x(p)+1$ 以上になるように，配列を並べ替える ($q = \texttt{PARTITION}(x,p,r)$)．各部分列を quick sort すれば，全体がソートされたことになる．$x(p)$ を基準 (pivot) にとったのは根拠がない．Pivot データの順位によって所要時間が変わる．

$\boxed{\texttt{PARTITION}(x,p,r)}$
$w \leftarrow x(p)$ $\quad$ $x(p)$ がピボット (基準)
$i \leftarrow p-1$
$j \leftarrow r+1$
(A) repeat $j \leftarrow j-1$ until $x(j) \le w$
repeat $i \leftarrow i+1$ until $x(i) > w$
$i \lesseqgtr j$
YES $\Longrightarrow x(i) \leftrightarrow x(j)$, (A) に戻る
NO $\Longrightarrow j$ を値として終わる．但し $i > r$ ならば $x(p) \leftrightarrow x(r)$ として $j-1$ を値とする

§0.2.4. ヒープソート

ヒープソートは二分木に基づくヒープ (heap) という概念を用いる高度なアルゴリズム上の概念を含んでいる．しかし，極端に長くはないプログラムで in place かつ理論的に最善の漸近的な速度を得る．選択法の改良型．代表的なソートの例として，また，アルゴリズムという概念の重要性の指摘を込めて，特に取り上げる．

データ構造．　データ $X = \{x(i) \mid i = 1, 2, \cdots, N\}$ の間に何らかの２項関係（$X \times X$ の部分集合 R のこと）が定義されているとき，データの集合とその間の関係を，データ一つ一つを点と見て関係がある点同士を線でつないだグラフとみなすこと．$(x, y) \in R \ \rightarrow \ (y, x) \in R$ のとき（無向）グラフ，それ以外，特に，$(x, y) \in R \ \rightarrow \ (y, x) \notin R$ のとき有向グラフという．グラフが連結であるとは，任意の二つの異なる点 x, y に対して，$(x_i, x_{i+1}) \in R$ または $(x_{i+1}, x_i) \in R$ を満たす点列 $x = x_1, x_2, \cdots, x_n = y$ が取れること．

木 (tree)．

$$
\begin{aligned}
&\exists k \in \mathbf{N}, x(a_i) \in X,\ i = 1, 2, \cdots, k; \\
&(x(a_i), x(a_{i+1})) \in R \text{ or } (x(a_{i+1}), x(a_i)) \in R,\ i = 1, 2, \cdots, k-1, \\
&(x(a_k), x(a_1)) \in R \text{ or } \ (x(a_1), x(a_k)) \in R, \\
&a_i \neq a_j \ (i \neq j)\,,
\end{aligned}
$$

を満たす列 $x(a_1), x(a_2), \cdots, x(a_k)$ をループと呼ぶ．ループのない連結なグラフを木と呼ぶ．有向木グラフでは矢印の元を親，先を子と呼ぶことがある．

二分木 (binary tree)．　各点に高々２つの子しかない木．どの親の子でもない点が１つだけある．それをルートと呼ぶ．ルートを level 0 呼び，一般に level i の子を level $i+1$ とよぶ．

Binary tree の配列上の表現．　最後の2 level 以外の点は子を２つずつ持っている binary tree は，ルートのデータを $x(1)$ とし， $x(s)$ の子を $x(2 * s)$ と $x(2 * s + 1)$ とする（木の左の子を先にする），という対応で，一対一に対応する：

$\texttt{PARENT}(i) := [i/2]$
$\texttt{LEFT}(i) := 2i$
$\texttt{RIGHT}(i) := 2i + 1$

以下，この対応に基づいて，実際のデータは配列に並んでいるにも関わらず，あたかも木になっているかのようにアルゴリズムを表現する．

Heap．　最後の段の level 以外の点は子を２つずつ持っている二分木において，木の各点のデータが，どの親子関係についても親のデータのほうが大きくなっているとき heap であると呼ぶ： $x(\texttt{PARENT}(i)) \geq x(i)$.

かってな二分木のデータからヒープを作るアルゴリズムは次の通り．子を持つ最後（配列上最後）の親 $x([N/2])$ から始め，$[N/2], \cdots, 1$ の順に以下を行う[5]：親のほうが子より小さければ，２つの子のうち大きい方を親と交換する．交換した子を新たな親としてそこから下に向かって親の方が小さければ大きい方の子と交換して親子の逆転がなくなるまでくり返す.

$\texttt{HEAPIFY}(x, i)$ は点 i を root とし，その右側と左側はそれぞれヒープになっている部分二分木をヒープにする.

$\boxed{\texttt{HEAPIFY}(x, i)}$
$\ell \leftarrow \texttt{LEFT}(i)$
$r \leftarrow \texttt{RIGHT}(i)$
$\ell \lesseqgtr \texttt{heap} - \texttt{size}(x)$
　YES $\Longrightarrow$ 終わり
　NO $\Longrightarrow$ $x(\ell) \lesseqgtr x(i)$
　　　　YES $\Longrightarrow max \leftarrow \ell$
　　　　NO $\Longrightarrow max \leftarrow i$
　　$r \lesseqgtr \texttt{heap} - \texttt{size}(x)$

[5]つまり，ある親に注目するとき，その二つの子を親とする部分二分木はヒープになっているとする．

YES $\Longrightarrow x(r) \gtrless x(max)$

YES $\Longrightarrow max \leftarrow r$ (NO は何もしない)

$max \neq$ i

YES $\Longrightarrow$ 終わり

NO $\Longrightarrow$ $x(i) \leftrightarrow x(max)$

$\texttt{HEAPIFY}(x, max)$

Heap sort のアルゴリズム． 配列の初期値に対応する二分木から始めて，heap を作る．ルート $(x(1))$ は最大の数なので最後のデータ $(x(N))$ と交換する．N 番目のデータは正しい位置に来たので以後固定し，$N-1$ 個のデータからなる配列とみなして以上をくり返す．帰納的に木のサイズが 1 になれば対応する配列は昇順になっている．

$\boxed{\texttt{BUILD}-\texttt{HEAP}(x)}$

$\texttt{heap}-\texttt{size}(x) \leftarrow N$

$\boxed{i:\ [N/2] \sim 1}$

| $\texttt{HEAPIFY}$(x,i)

———

$\boxed{\texttt{HEAPSORT}(x)}$

$\texttt{BUILD}-\texttt{HEAP}(x)$

$\boxed{i:\ N \sim 2}$

| $x(1) \leftrightarrow x(i)$

| $\texttt{heap}-\texttt{size}(x) \leftarrow \texttt{heap}-\texttt{size}(x) - 1$

| $\texttt{HEAPIFY}(x, 1)$

———

問．

(i) 以上の説明に基づいて heap sort のアルゴリズムをフローチャートにせよ．

(ii) Heap sort の平均及び最悪の比較回数と交換回数の漸近形を求めよ．

Heap sort プログラム具体例．

```
/* heap sort (昇順) */
/* C program from C言語によるはじめてのアルゴリズム入門,  */
/* 河西朝雄, 技術評論社, 1992年, p.297--p.299 */
/* 19970413 */
/* ---------------------------------------------------------------- */
#include <stdio.h>

void swap(int *, int *);
void shiftdown(int, int, int *);

void main(void)
{
    static int heap[100];
```

```
    int i,n,m;

    n=1;                          /* データを木に割り当てる */
    while (scanf("%d",&heap[n])!=EOF)
         n++;
    n--;                          /* データ数 */
    for (i=n/2;i>=1;i--)          /* 初期ヒープの作成 */
         shiftdown(i,n,heap);
    m=n;                          /* n の保存 */
    while (n>1){
         swap(&heap[1],&heap[n]);
         n--;                     /* 木の終端を切り離す */
         shiftdown(1,n,heap);
    }
    for (i=1;i<=m;i++)
         printf("%d ",heap[i]);
}

void shiftdown(int p, int n, int heap[])   /* 下方移動 */
{
    int s;
    s=2*p;
    while (s<=n){
         if (s<n && heap[s+1] > heap[s])   /* 左と右の子の小さい方 */
              s++;
         if (heap[p]>=heap[s])
              break;
         swap(&heap[p],&heap[s]);
         p=s; s=2*p;                       /*親と子の位置の更新*/
    }
}

void swap(int *a, int *b)
{
    int w;
    w=*a; *a=*b; *b=w;
}
```

スプレッドアウト法．　値の範囲が限られているなどの，入力データに関する情報があれば，もっと速いアルゴリズムが可能であることは一般に知られている．

高速大容量の作業領域（記憶装置）があれば交換（代入）を N 回にできると思う[6]．

サイズ 2^N の作業領域配列 z を用意し，データの値にならない値 (0 とする) で初期化しておく．$z(2^{N-1}) = x(1)$, $z(2^{N-2}k) = x(2)$ ($k = 1$ または 3), のように，配列の中へデータを昇順になるように順次代入していく．比較作業は大小関係の正しいほうの領域の中点をたどっていく．非データ要素にたどりついたらそこに

[6]哲弥流．スプレッドアウト方という名前も勝手に付けた．971016: 速いことの理由は本質的に counting sort と呼ばれる方法と同じ．但し，作業配列をこのように極端に大きくとる発想は今までなかっただろう．Introduction to Algorithms, T.H.Cormen, C.E.Leiserson, R.L.Rivest, MIT, p.175

データを代入する．

$k=0$ から出発して $j=1,2,\cdots,2^N$ に対して順に

$$z(j)\neq 0 \implies k\leftarrow k+1,\ y(k)=z(j)\,,$$

を実行すると，$y(i),\ i=1,2,\cdots,N,$ は整列したデータになる．

アイディアはカード（年賀状やトランプなど）の整理で，床の上に広げて順番に並べ，最後に端から重ねていく方法．これは私のオリジナルだが，高速大容量の記憶装置は現状では非現実的．また，作業領域からデータを探す回数（比較作業に相当）が大きいので，比較が代入より遅い現状では非現実的．しかし，前者の制限は減少の傾向にあるし，並列計算機にむいているので後者の制限も将来的には本質的とは思えない．