愛知工業大学研究報告
第19号B　昭和59年

最上位桁優先式 Radix-Sorting 法の予備的研究

小　田　哲　久

A Preliminary Study of MSD-First Radix-Sorting Methed

Tetsuhisa ODA

Many kinds of sorting algorithms have been developed from the age of Punched Card System. Nowadays, any sorting algorithm can be called either (1) internal sorting methed or (2) external sorting method. Internal sorting method is used only when the number of records to be sorted (N) is not so large for the internal memory of the computer system. Larger memory space has become available with the aid of semiconductor technology. Therefore, it might be desired to develop a new internal sorting methed which works efficiently for very large N on the computer with huge internal memory space.

This study investigated the internal sorting algorithm which works well when N is very large. The sorting algorithm presented in this study is developed for designing and constructing a data base system, such as marketing-research data base. The algorithm could be called a kind of radix-sorting method which starts at the most significant digit (MSD). After presenting the algorithms and associated programs written in BASIC language, suggestions for improving these algorithms are made. It is concluded that MSD-first radix-sorting method could be one of the most powerfull and applicable sorting methods in the near future.

Ⅰ．緒言

ソーティング（Sorting：並べ換え）の技法は，コンピュータ出現以前の，パンチカードシステム時代から，多くの先人により工夫・改良が加えられて来て，ハードウェアの機構や性能，又，そのソーティング技法が組み込まれるソフトウェアの設計上の要請，等に応じて，実に多種・多様なアルゴリズム群が出現している。これらのソーティング・アルゴリズムは，大別して，外部ソートと内部ソートに分けられるが，以下に示すのは，メモリ内部でソーティングを行う，内部ソーティングのアルゴリズムを分類する基準の例である。

1．Sorting by Insertion
2．Sorting by Exchange
3．Sortiog by Selection
4．Sorting by Merging
5．Sorting by Distribution

この基準は，Donald E. Knuth の著書[1]中に採用されているもので，基本操作をどの様な手段で行うか，に着目したものである。

個々のアルゴリズム，例えばQuick-Sort や Bubble-Sort 等，ポピュラーなソーティング技法は，値の交換を基本操作としているので，Sorting by Exchange に分類される。表1は，よく使われる，あるいは，興味深い特徴を持つ，各アルゴリズムが，上記5分類のどこに入るかを，一覧表にしたものである。

これらのアルゴリズムは，それぞれ長所・短所があり，例えば，アルゴリズムが単純でプログラムが書きやすいものは，おおむね低速である。又，高速なアルゴリズムは一般に複雑であり，しかも，余計なメモリ領域を必要とする場合が多い。本報告に於て提示されるアルゴリズムは，著者が，市場調査結果分析用データー・ベース・システムを設計中に，システムの要請に応じて開発したものであるが，計算機アルゴリズムの総集ともいうべき，上記 Knuth の著書に当って詳細に検討した所，これは，Sorting by Distribution のうちの，Radix-sorting の一種であるとわかった。しかも，その，Radix-sorting のうちでも，most significant digt（MSD）—first radix method と呼ばれうるタイプであるという事が判明した。それを敢えてここに報告する理由は以下の通りである。

1．アルゴリズムの原理を可能性として示唆する事と，具体的にプログラムとして提示する事は全く違う行為である。というのは，基本的に同一の原理に基づくア

ルゴリズムであっても，プログラム化の過程で相当のバリエーションが生ずるからである。

2．Knuthが上記著書[1]中で示唆している内容は，筆者が本報で提示する内容と，微妙な点で違いがある様に思われる。その違いは，KnuthがRadix-sorting全般について，メモリ最小化設計を前提として議論しているように思われるのに対して，筆者の場合は利用可能なだけメモリを使い，そのぶん高速化をはかる，という方針で設計している点である。これは，ある意味では年代の違い，すなわち，ハードウェアの発達（メモリの増大）を反映しているともいえよう。

3．実際的な見地から，本プログラム自体，極めて有用であり，公開する事によって，多方面で有効に利用されうるからである。特に，データーベース・システムへ組み込んで，マルチ・キー・ソートをする場合，極めて効果的である。

4．一見複雑にみえるが，実際はさほどでもなく，今後，別の高水準言語系（PL/I又はPASCAL）へ移殖する事により，極めて簡潔かつ高性能なプログラムが書ける可能性を含んでいる。又，逆に低水準なアセンブリー言語で一部分を記述する事により，省メモリ化と高速化が可能となる。つまり，発展の可能性が大きい。

II．状況設定

本報で提示するソーティング・アルゴリズムが，どの様な状況を想定して考案されたかを示す事は，応用面を端的に表すと共に，アルゴリズム自体の理解を容易なものとするであろう。本アルゴリズムは，既に簡単に触れてあるとおり，データーベース・システムへの組み込みを想定して開発されたが，詳細は，以下の通りである。

(1) ランダム・アクセス可能な外部メモリー（一般的には，ディスク・メモリー）上に，大量のデーターが，固定長もしくは可変長のレコード型式で格納されており，各レコードは，レコード番号（正の整数）をKeyとして，ランダム・アクセス可能であるとする。又，このランダム・レコードのうちで，Sortingに使用する際にKeyとなるItemは，全部，Index Fileとして，用意されているものとする。

(2) Sortingの手順は，まず，このIndex Fileを，メモリ内に取り込んで，そのItemの値を昇順に並べたときの，「レコードNo.の並び」をメモリ上に実現する。次にこの「レコードNo.の並び」をKeyとして，その順番に，外部メモリ上のランダム・レコードを取り出すと，それは，Index File化されているItemの値が昇順となるように，レコード群をSortingした結果となっている。

(3) さらに，このIndex File群化されたKey-Item群を，複数同時にメモリ内に取り込み，これらのKey-Item群の間に，優先順位を割り当ててSortingする。その優先順位は当然ながら，ユーザーの指示によって決定する。以上がMulti-Key-Sortingの実行手順の設定状況である。

III．アルゴリズム開発の基本方針

以上の状況設定の下で，Sortingアルゴリズムを開発するに当り，次の様な基本方針を立てて，作業を行った。

(1) データーの値を直接比較する事をしない。(non-compare-sorting)

(2) データーの値を直接，メモリ上で移動させる事をせず，代りに，データーのレコードNo.を移動させる。

(3) 利用可能な限り，メモリを充分に使って，その分，高速化を可能とする。

(4) 基本的に，データ数と，ソーティングの処理時間が比例する様にし，超大量データに対する高速性を追求する。

(5) 掛け算，割り算等，時間のかかる演算はなるべく使わず，単純な演算を中心に組み立てる。

(6) データを外部メモリ等から入力する時点で，同時に，データの最大値，最小値，区分ごとの出現頻度等の情報を知り，これを，その後のSorting手続きに於て積極的に利用する。

IV．開発装置ならびにOS，及び，使用言語

以上の方針に従い，具体的なアルゴリズムを，特定言語によるプログラムという表現形式で実現する事は，さほど困難な事ではない。ただし，プログラムの大きさや，メモリの占有効率，ならびに処理速度等の差異を別にすれば，である。これらは，コンピュータの機構・性能や，OS及びプログラミングを含む，ソフトウェアーの機能・性能に，大きく依存するからである。

今回は，著者の研究室にあって利用しやすい，という意味と，手軽に追試・応用されやすいように，という意味の両面から，諸々の欠点には目をつむって，敢えて，8 bitパーソナル・コンピュータを使用した。

具体的機種は，沖電気製if-800/20型であり，これに，HP-IBバスを介して，ISA社製の20MBハードディスクを接続して用いた。OSは，デジタル・リサーチ社のCP/Mを採用し，又，開発言語は，BASICである。BASICには種々あるが，ここではマイクロソフト社のBASIC-80文法に準拠した。

V．結果

1．アルゴリズムA（基本的アルゴリズム）

(1)　まず，データの値は，0から9までの整数値であり，項目x 1と項目x 2の2項目から成り，データ数は両項目ともn個であるとする。これらを各々，配列x 1(i)と，x 2(i)に格納してゆく時に，同時に，x 1の最大値mx 1，x 2の最大値mx 2，x 1の最小値mn 1，x 2の最小値mn 2を算出しておく。又，同時に，f 1(i)の値がjであればf 1(j)の値を1つ増加する。といった手順で，x 1に含まれるデータの値の頻度を配列f 1に格納する。

(2)　次に，f 1(mn 1)からf 1(j)までの累積値を計算し，s 1(j+1)に代入する。数式で表記すると，

$$s\,1(j+1) \longleftarrow \sum_{i=mn\,1}^{j} f\,1(i)$$

（ただし，jはmn 1からmx1まで）

となる。

(3)　以上までが第1パスの為の準備段階であり，x 1(i)のデータが，昇順に配分されうる場所b 1(i)への道すじ〈i ⇒ j〉を確保する事になる。ただし，実際にb 1(j)に格納されるのは，そのデータのレコード番号iである。この格納作業が第1パスであり，ここでは，p(i)という配列が，格納場所を指示する，一種のポインタとして作用する。データx 1(i)はレコード番号の順に，どこへ配分されるべきか，が決定されてゆく訳だが，この過程は，以下の様に表記される。

p{x 1(i)}← p{x 1(i)}+1

b 1〔s 1{x 1(i)}+p{x 1(i)}〕← i

この過程をmn 1＝0，mx 1＝9の場合について具体的に述べると，まず，(1)でf 1(0)からf 1(9)までに，x 1中に出現する各数字の頻度が格納される事によって，b 1(1)からb 1(n)までの，レコード番号iの格納場所を，各数字にどれだけ割りあてたらよいか，が準備された事になる。又，次にf 1(0)からf 1(i)までの累積値をs 1(j)に格納してゆく事によって，b 1(i)のどこからどこまでを，x 1のどの数字に割り当てたらよいかを決めてゆく事になる。例えば，x 1(i)の値が0であれば，レコード番号iは，b 1(1)から，b 1{f 1(0)}までの範囲に，前から順に後方へ格納されてゆく。これを管理するのがp(0)という変数であり，これは，x 1(i)の値が0であるたびに1つずつ増加してゆく。ただし，p(0)の初期値はゼロである。次にx 1(i)が，任意の値kであったとしよう。(当然ながらkは0から9までの整数値のいずれかの値である。)その場合，レコード番号iは，b 1{s(k)}から，b 1{s(k)+f(k)}までの範囲のうち，b 1{s(k)+p(k)}の場所に格納されるわけであるが，ここでのp(k)の値は，x 1(0)から，x 1(i)までに，値kが何回出現したか，その回数が格納されているわけである。

(4)　以上で，x 1についてのソーティングは完了した事になる。次に，同じx 1の値を待つレコードどおしの間で，x 2について，昇順にソーティングする事になる。その手順は，ほとんど，(1)～(3)の手順に準ずる。

(5)　まず，x 1の値がmn 1であるレコード群の数はf 1(mn 1)に格納されているので，この値が0であれば，x 2の値についてソーティングする事は無用である。その場合には，次の，x 1が(mn 1＋1)なる値をとるレコード群のソーティングに移る事になる。

(6)　f 1(mn 1)が0か否かの判定の結果，x 1の値がmn 1であるようなレコーぽド群の個数が0でない事が判明すると，まず，配列f 2，s 2，pを0に初期化する。これらは，(1)～(3)の手順における，f 1，s 1，pの機能に対応する。すなわち，f 2へは，x 1の値がmn 1であるようなレコードのうち，x 2の値に出現してくる各数字の頻度が格納される。具体的に述べると，x 1の値がmn 2であるレコードのレコードNo.は，b 1(1)から，b 1{f 1(mn 2)}まで，即ち，b 1{s 1(mn 2)+1}から，b 1{s 1(mn 2＋1)}までに格納されている。そこで，これらのレコードを次々に呼び出してx 2の値を調べ，その頻度情報をf 1(0)からf 1(9)に記憶しておくのである。そして，f 2の累績がs 2に格納される。数式表記すると，

$$s\,2(j+1) \longleftarrow \sum_{i=1}^{j} f\,2(i)$$

（ただし，jは，mn 2からmx 2まで）

となる。以上で，第2パスのうち，x 1の値がmn 1であるレコードの範囲内でのソーティングの準備が完了した事になる。

(7)　手順(6)で述べた事から容易にわかるとおり，x 1の値がkであるレコードのレコードNo.はb 1{s 1(k)+1}から，b 1{s 1(k+1)}までに格納されているので，これらのレコードを順番に呼び出してx 2の値を調べ，その値に応じて，配列b 2へレコード番号b 1(i)を格納してゆく。この過程は以下の様に表記される。

c ←— x 2{b1(i)}

p(c)+1

a ←— s 2(c)+p(c)+s 1(k)

b 2(a)←— b 1(i)

この過程の原理はほとんど手順(3)と同じであるが，x 2の値を取り出すのに，一旦，b 1を経由するので，その分，余計な手続を要する。ここでは，表記を簡潔にする

為に，c，aという，中間変数を採用しているが，これが高速化に貢献するか否かは，もっぱら，コンパイラーの仕様，特に，最適化（Optimization）の程度に依存する事になろう。

(8) 以上で，x 1の値がmn 1であるレコードの，x 2についてのソーティングが完了した事になる。そこで，今度は，x 1の値がmn 1＋1であるレコードの，x 2についてのソーティングに移る。その手順は，(5)，(6)，(7)を追う事になる。ただし，x 1の値がmn 1＋1であるとして行うのである。以下同様に，x 1の値を順次1ずつ増加させては(5)，(6)，(7)を実行してゆき，x 1の値がmx 1になって，(7)が完了した時点で，ようやく，x 1，x 2の両方をKeyとして，x 1を優先させたソーティングが完了した事になる。

(9) 最終的なソート結果は，レコード番号の並びの形で，b 1(1)からb 1(n)までに格納されている。これをどう使うかは，ユーザーの目的によるわけだが，II.（状況設定）で述べたとおりの状況であれば，この，レコード番号の並びをKeyとして，外部メモリのランダムファイルを順次参照する事になろう。その際に，一旦，配列b 1の内容を外部メモリにファイルして置けば，再利用が可能であるが，これらの処理は，全く，ユーザーの判断に任されるわけである。

2．アルゴリズムB（応用アルゴリズム）

アルゴリズムAは，データの値を0から9までに限定していたが，これを0から255までの整数に拡張する。そして，x 1とx 2を別々の項目として把えず，256進法の上位桁の数値と下位桁の数値として認識する。つまり，2Byte整数の，〈上位Byte〉と〈下位Byte〉として把握する訳である。これによって，0000Hから，FFFFHまでの整数が取り扱える事になり，通常のソーティング技法とほぼ同様の使い方が可能となる。この様な解釈上の相違と，10進と256進法の差はあるものの，他は，アルゴリズムAと違いはない。

表1 ソーティング法の分類

基本操作	具体的アルゴリズム
1. Insert	Address calculation-sort Schell-Metzner's sort
2. Exchange	Quick-sort Bucher's parallel-sort Bubble-sort
3. Selection	Straight selection-sort Heap-sort
4. Merging	Natural merge-sort Straight merge-sort Chain merge-sort
5. Distribution	Radix-sort

3．プログラム1

附録プログラム1に示したのは，アルゴリズムAのデモンストレーション用のBASICプログラムである。x 1，x 2の入力がキーボードから直接入れる形式になっているのは，動作を簡単に確認してもらう為である。入力は，x 1(i)とx 2(j)の値（これらはいずれも0から9までの整数値）を，コンマで区切って入力してゆくわけだが，出力は，書式上の工夫で，x 1(i)とx 2(i)を，みかけ上結合し，あたかも2桁の10進数であるかの如くに印字している。出力されるもののうち，見出しb 1(i)は，配列B 1に，x 1についてのソーティング結果が，「レコードNo.の並び」の形で格納されているので，これをそのまま出力したものであり，見出しPass 1は，配列B 1の示すx 1とx 2の内容を結合して表示したものである。又，見出しb 2(i)は，配列B 2に最終的な，x 1及びx 2についてのソーティング結果が，「レコードNo.の並び」の形で格納されており，これを出力したものである。そして，見出しPass 2は，配列B 2の示すx 1とx 2の内容を結合して表示したもので，本当にソーティングが完了している事を視覚的に確認する為の情報である。

4．プログラム2

附録プログラム2に示したのは，アルゴリズムBのデモンストレーション用のBASICプログラムである。本プログラムは，他のソーティング技法，例えばクイックソート等との処理速度比較を行う為に作成されたものである。ただし，入出力時間，特に入力時間については，全く問題視しない事にした。何故なら，実際的利用状況下では，データをx 1とx 2に分解する作業は，入力バッファーにデータを取り込んだ時点で，極めて効率的に行われるはずだからである。又，x 1の値の頻度分布や，最大・最小値の情報を取り出す作業は，データ入力に要する時間に比べ，充分に小さいからである。

5．処理速度測定結果

参考の為に，処理時間を測定した結果を図1に示す。右側の直線状のグラフは，比較の為の，Quick-Sort Programによる結果であり，左方の曲線が，アルゴリズムBによるもので，プログラム2をそのまま使用している。尚，Quick-Sort Progmamは岡村迪夫氏が，BASICで書かれたもの[2]を，入出力形式のみ変更して利用した。

VI. 考察

1．設計方針が確定した時点で，アルゴリズムの概要も決ってしまい，コーディング作業はさほどの困難を伴わなかった。しかし，コーディング中に気づいたのは，

BASIC で書こうとしたのは失敗であり，PL/I あるいは PASCAL で書くべきだったという点である。プログラムを一見すればわかるとおり，レベル 1 とレベル 2 は，ほとんど同一の手続であり，さらにレベルを増やす場合には，プログラムが大変に冗長なものになって行く点である。ここはもっとすっきりと，再帰的に書くべきである。

2．ここに提示したプログラムは，両方とも，デモンストレーション用であり，データーの取込みは実際的な形ではない。既に簡単に述べたように，データーをバッファーに入力した時点で，Digit ごとに，別の配列に格納するわけであるが，この作業は，PL/I を採用した場合，極めて書きやすい事が予想できる。しかし，Digit ごとに別配列を使う必要は必ずしもない。例えば，入力データを Digit で分解する事なく，配列に格納したとしよう。この場合，ある配列要素がメモリのどの番地に，どの様な形式で格納されているかがわかれば，直接に，特定アドレスを指定する事により，今回提示したアルゴリズムを採用する事ができる。ただし，その場合，Radix の数値は，ハードウェアーによって，ほとんど一義的に決定される事になる。

3．今回提示したアルゴリズムには，未だ大いに改良の余地がある。例えば，配列ｂ１は，最後まで保存され続ける必要はなく，パス 2 に於て読出した分は，もはや 2 度と読出される事はない。従って，レベル数をもっと増やしても，所要メモリがさほど増加するわけではない。又，配列ｆ２は配列ｆ１と共通にしてもよい。

4．Radix-Sorting には，Least Significant Degit（LSD)からスタートするソーティング技法と，Most Significant Degit（MSD）からスタートする技法の 2 つが考えられている。Knuth はこれらを，各々，LSD-first radix method と，MSD-first radix method と名づけて各々の得失を論じ，「MSD-first radix method が格別にすばらしい動作をするわけではない理由は，pile（積み重ねる山）の数が多くなり過ぎて煩雑だからである。」[3]と評している。確かに，MSD-first radix method は一般的な LSD-first radix method の場合ほどにアルゴリズムが簡潔ではない。しかし，MSD-first 法には，LSD-first 法と比べ，演算を省略できる可能性がある。今回提示したプログラムの場合はそうなってはいないが，これを可変長データに適用できるように，再帰的構造に書き直した時，この特質が浮び上がってこよう。

尚，Knuth の書以外でも Radix-Sorting が扱われているものはあるが，これらはことごとく，LSD-first 法である。例えば，Alfred V. Aho, John E. Hopcroft 及び Jeffrey D. Ullman による，「データ構造とアルゴリズム」[4]の中では，Radix Sorting のアルゴリズムが極めて明解に示されているが，これもやはり LSD-first 法である。

又，岩波講座情報科学シリーズ中の，渋谷政昭・山本毅雄共著，「データ管理算法」中では，「一般に整順列化の算法で，同順位のものの前後関係を，元のまま保つものは安定な算法と呼ぶ。」[5]として，安定な算法を用いるソーティング技法の例に，基数法（radix sort）を採り上げており，又，同書の「他の整順列化法」の節中で，再度，基数法が採りあげられ，原理と，長所・短所の説明が加えられている[6]が，ここでの説明も LSD-first 法を前提としている。

これらの書中で言及されている，Radix 法の欠点は，次の様にまとめる事ができる。

(1) Key の値が，固定長の整数を典型とする有限精度の数で，M進Ｐ桁の数として扱えるものに限定される事。（M，Ｐは自然数）

(2) 処理するパス数は，Radix をＭとすればＭ×Ｐ回必要であり，Radix をＬ×Ｍとすすれば，Ｍ×Ｐ/Ｌ回ですむが，一般には，メモリ容量の制約があって，Ｌを余り大きくする事ができない。又，配列の初期化等，準備作業の手数もＰ回必要である。従って，Ｍ×Ｐが大きいと，極めて効率の悪いものとなる。（Ｌは自然数）

これらは，LSD-first 法の場合には避けられないが，MSD-first 法ならば，多少の緩和が可能である。何回か述べて来た様に，可変長データーを扱えるような，MSD-first 法を開発すればよい。これには，各データを，リスト型としてもよいし，又，単にデータの長さの情報を，Key に添えてもよい。あるいは，各データの末端が終了した事を知らせるような，マーカーを置く方法でもよい。いずれにせよ，再帰を終らせる為の環境の 1 部として使えればよい。

5．処理速度のグラフを見ると，データ数Ｎが2500個程度では，まだまだ「Ｎに比例する処理時間」にはほど

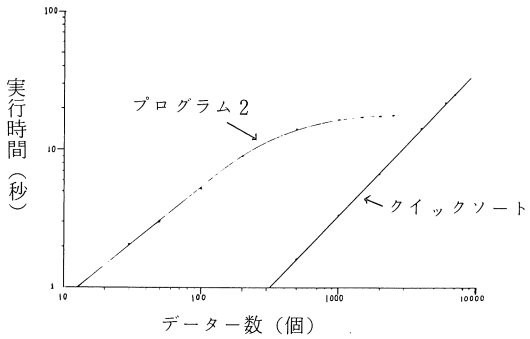

図１　処理時間測定結果

遠く，Nが大きくなるにつれ，データ1個当りの処理時間が減少している様子がうかがわれる。このアルゴリズムの実力はまだ表面に出ていないのであろうが，それでも，Quick-Sortを抜くのは，N ≒ 5000個～6000個のあたりであろうと推定できる。

Ⅶ. 結論

Radix Sorting法は歴史的には，パンチカード・システムの時代から使われて来て，「誰が開発したかも不明」[7]であるといわれるほど，古典的な技法である。そして，機械がパンチカード用Sorterから，コンピュータへと変っても，最下位桁から順にSortしてゆく，いわゆるLSD法が使われ続けて来た。一方，同じRadix Sortingでも，最上位桁から順にSortしてゆく，いわゆるMSD法は，下位桁でのSortingを管理する為の手続の煩雑さと，その為に必要なメモリ量の大きさによって，実用的でないと考えられて来たようである。又，LSD法の欠点が，あたかもRadix法自体の欠点の様に誤解されて来た可能性もある。

本報で呈示したアルゴリズムは，まだ，これらの欠点を完全に克服できたわけではないが，少くとも，解決の為の糸口を示す事が出来たと思われる。次の課題は，これを実現して，〈処理時間が，基本的にデータ個数に比例する〉という長所を持つRadix-Sortingを，多方面で応用できる様にする事である。

LSI技術の進歩により，我々が利用できるメモリ空間は極めて広大なものとなりつつある。拡大されたメモリ空間は，従来考えられもしなかった様なアルゴリズムを可能にするかもしれないが，本報で示唆した，可変長データへ適用可能なタイプのMSD-first Radix-Sorting法は，近未来のコンピュータ・システムに於て，極めて有効なソーティング手段となろう。

参考文献

1）D. E. Knuth : The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley Publishing Company, Inc., 1973.

2）岡村迪夫：super BASIC, CQ出版社, 1982, p.96

3）D. E. Knuth：上掲書, p.177

4）Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman: Data Structures and Algorithms, Addison-Wesley Publishing Company, Inc., 1983. p.280～p.281

5）渋谷政昭・山本毅雄：データ管理算法，岩波講座情報科学—11, 岩波書店, 1983, p.102～p.104

6）渋谷政昭・山本毅雄：上掲書, p.117～p.121

7）D. E. Knuth；上掲書, p.170～p.171

8）森脇幸生：プログラム言語PASCALとその応用，工学図書, 1981.

（受理　昭和59年1月17日）

付録プログラム1　ならびに実行例

```
100 ' **********  radix sort  program 1 **********
110 option base 0
120 defint a-z
130 dim x1(1000),b1(1000),f1(9),s1(10),p(10)
140 dim x2(1000),b2(1000),f2(9),s2(10)
150 ' .......... initialize mx1,mx2,mn1,mn2
160 mx1=0 :mx2=0
170 mn1=9 :mn2=9
180 ' .......... clear f1,s1,p
190 for i=0 to 9
200     f1(i)=0 :s1(i)=0 :p(i)=0
210 next i
220 ' .......... data input
225 input "n=";n
230 for i=1 to n
240     input x1(i),x2(i)
250     f1(x1(i))=f1(x1(i))+1   :' count distribution
255     ' .......... maximum & minimum value of x1,x2
260     if x1(i)>mx1 then mx1=x1(i)
265     if x1(i)<mn1 then mn1=x1(i)
270     if x2(i)>mx2 then mx2=x2(i)
275     if x2(i)<mn2 then mn2=x2(i)
280 next i
290 ' .......... count sigma(f(i))
300 for i=mn1 to mx1
310     s1(i+1)=f1(i)+s1(i)
320 next i
330 ' .......... 1st pass
340 for i=1 to n
350     p(x1(i))=p(x1(i))+1
360     b1(   s1( x1(i) ) + p( x1(i) )   )=i
```

```
370 next i
380 '
390 ' .......... lebel 2
400 for j=mn1 to mx1
410   if f1(j)=0 then goto 700
420   ' .......... clear f2,s2,p
430   for i=0 to 9
440       f2(i)=0 :s2(i)=0 :p(i)=0
450   next i
460   ' .......... count f2
470   for i=s1(j)+1 to s1(j+1)
480       f2(  x2( b1(i) )  ) = f2(  x2( b1(i) )  )+1
490   next i
500   ' .......... count sigma(f2)
510   for i=mn2 to mx2
520       s2(i+1)=f2(i)+s2(i)
530   next i
540   ' .......... 2nd pass
550   for i=s1(j)+1 to s1(j+1)
560       c=x2( b1(i) )
570       p(c) = p(c)+1
580       a=s2(c)+p(c)+s1(j)
590       b2(a)=b1(i)
600   next i
610   ' ********** space for the next lebel **********
620   ' ********************************************
700   ' .......... loop1    -------------case f1(i)=0
710 next j
800 ' .......... output procedure
810 print spc(5);"i";"   data";spc(11);"b1(i)";spc(7);
815 print "pass 1";spc(11);"b2(i)";spc(6);"pass 2"
820 for i=1 to n
830     print using "######    ";i;
840     print using "#";x1(i);x2(i);
850     print spc(10);
860     print using "  ######";b1(i);
870     print spc(10);
880     print using "#";x1(b1(i));x2(b1(i));
890     print spc(10);
900     print using "  ######";b2(i);
910     print spc(10);
920     print using "#";x1(b2(i));x2(b2(i))
930 next i
940 print:print:print
950 '
1000 end
```

```
Prog1
n=? 20
? 5,2
? 3,4
? 3,6
? 5,5
? 1,9
? 4,3
? 7,8
? 1,4
? 0,6
? 4,8
? 5,9
? 4,6
? 5,5
? 7,2
? 4,4
? 6,5
? 5,2
? 5,5
? 4,1
? 0,5
     i    data          b1(i)       pass 1          b2(i)       pass 2
     1    52               9           06             20           05
     2    34              20           05              9           06
     3    36               5           19              8           14
     4    55               8           14              5           19
     5    19               2           34              2           34
     6    43               3           36              3           36
     7    78               6           43             19           41
```

8	14	10	48	6	43
9	06	12	46	15	44
10	48	15	44	12	46
11	59	19	41	10	48
12	46	1	52	1	52
13	55	4	55	17	52
14	72	11	59	4	55
15	44	13	55	13	55
16	65	17	52	18	55
17	52	18	55	11	59
18	55	16	65	16	65
19	41	7	78	14	72
20	05	14	72	7	78

付録プログラム 2　ならびに実行例

```
100 ' ********** radix sort  program 2 **********
105 ' extended for every INTEGER constants ( -32768 to +32767 )
110 option base 0
120 defint a-z
130 dim x1(2000),b1(2000),f1(255),s1(256),p(256)
140 dim x2(2000),b2(2000),f2(255),s2(256)
150 ' ..........  initialize mx1,mx2,mn1,mn2
160 mx1=0 :mx2=0
170 mn1=256:mn2=256
180 ' ..........  clear f1,s1,p
190 for i=0 to 255
200     f1(i)=0 :s1(i)=0 :p(i)=0
210 next i
220 ' ..........  data input
221 print "This program can treat every INTEGER constants. ( -32768 to +32767 )"
222 input "Please input filename of the data";fln$
223 open "I",#1,fln$
225 input "n=";n
230 for i=1 to n
240     input #1,dt
244     x1(i)=int(dt/256)               :'x1(i) := upper byte of dt
246     x2(i)=dt-x1(i)*256              :'x2(i) := lower byte of dt
248     x1(i)=x1(i) + 128               :'shift x1(i)
250     f1(x1(i))=f1(x1(i))+1           :'count distribution
255     ' ..........  maximum & maximum value of x1,x2
260     if x1(i)>mx1 then mx1=x1(i)
265     if x1(i)<mn1 then mn1=x1(i)
270     if x2(i)>mx2 then mx2=x2(i)
275     if x2(i)<mn2 then mn2=x2(i)
280 next i
285 close #1
290 ' ..........  count sigma(f(i))
295 print "** start **"
300 for i=mn1 to mx1
310     s1(i+1)=f1(i)+s1(i)
320 next i
330 ' ..........  1st pass
340 for i=1 to n
350     p(x1(i))=p(x1(i))+1
360     b1(   s1( x1(i) ) + p( x1(i) )   )=i
370 next i
380 ' ..........
390 ' ..........  lebel 2
400 for j=mn1 to mx1
410   if f1(j)=0 then goto 700          :' ..... goto label 1
420   ' ..........  clear f2,s2,p
430   for i=0 to 255
440       f2(i)=0 :s2(i)=0 :p(i)=0
450   next i
460   ' ..........  count f2
470   for i=s1(j)+1 to s1(j+1)
480       f2(  x2( b1(i) )  ) = f2(  x2( b1(i) )  )+1
490   next i
500   ' ..........  count sigma(f2)
510   for i=mn2 to mx2
520       s2(i+1)=f2(i)+s2(i)
530   next i
```

```
540   ' ..........  2nd pass
550   for i=s1(j)+1 to s1(j+1)
560       c=x2( b1(i) )
570       p(c) = p(c)+1
580       a=s2(c)+p(c)+s1(j)
590       b2(a)=b1(i)
600   next i
610   ' ********** space for the next lebel *********
620   ' ********************************************
630  ' ..........  label 1  -------------case f1(i)=0
710 next j
715 print "** completed **"
730 input "Do you want to print the result ? ----- Y/N";ky$
740 if (ky$="N") or (ky$="n") then end
800 ' ..........  output procedure
810 print:print spc(5);"i"tab(12)"data"tab(26)"b1(i)";
815 print tab(40)"pass 1"tab(56)"b2(i)"tab(70)"pass 2":print
820 for i=1 to n
830     print using "######";i;
835     dt=x1(i)-128:dt=dt*256:dt=dt+x2(i)
840     print tab(10);:print using "######"; dt;
860     print tab(25);:print using "######"; b1(i);
870     rs1=x1(b1(i))-128:rs1=rs1*256:rs1=rs1+x2(b1(i))
880     print tab(40);:print using "######"; rs1;
900     print tab(55);:print using "######"; b2(i);
910     rs2=x1(b2(i))-128:rs2=rs2*256:rs2=rs2+x2(b2(i))
920     print tab(70);:print using "######"; rs2
930 next i
940 print:print:print
950 '
1000 end
```

```
Prog2
This Program can treat every INTEGER constants. ( -32768 to +32767 )
Please input filename of the data? DT01
n=? 50
** start **
** completed **
Do you want to Print the result ? ----- Y/N? Y
```

i	data	b1(i)	Pass 1	b2(i)	Pass 2
1	-16704	14	-32653	14	-32653
2	-12779	12	-32320	12	-32320
3	-12330	28	-30598	28	-30598
4	994	40	-30581	40	-30581
5	-28946	16	-30096	16	-30096
6	18933	5	-28946	5	-28946
7	-190	33	-23782	33	-23782
8	-8929	43	-19571	43	-19571
9	31755	35	-18660	35	-18660
10	26319	34	-17921	46	-17984
11	14897	46	-17984	34	-17921
12	-32320	1	-16704	1	-16704
13	30765	48	-15918	48	-15918
14	-32653	31	-13989	31	-13989
15	29899	2	-12779	2	-12779
16	-30096	3	-12330	3	-12330
17	25992	8	-8929	8	-8929
18	10500	39	-8869	39	-8869
19	3571	45	-8280	45	-8280
20	20885	26	-3397	26	-3397
21	26689	50	-3064	50	-3064
22	23463	7	-190	7	-190
23	24172	24	450	24	450
24	450	4	994	4	994
25	5500	19	3571	19	3571
26	-3397	38	4454	38	4454
27	24100	25	5500	25	5500
28	-30598	29	6795	29	6795
29	6795	44	7116	44	7116
30	18278	18	10500	18	10500
31	-13989	11	14897	11	14897
32	18653	47	16092	47	16092
33	-23782	42	17255	42	17255
34	-17921	30	18278	30	18278
35	-18660	32	18653	32	18653
36	24692	6	18933	6	18933

37	23423	20	20885	20	20885
38	4454	22	23463	37	23423
39	-8869	37	23423	22	23463
40	-30581	23	24172	27	24100
41	24673	27	24100	23	24172
42	17255	36	24692	41	24673
43	-19571	41	24673	36	24692
44	7116	17	25992	17	25992
45	-8280	10	26319	10	26319
46	-17984	21	26689	21	26689
47	16092	49	28186	49	28186
48	-15918	15	29899	15	29899
49	28186	13	30765	13	30765
50	-3064	9	31755	9	31755