アルゴリズム 43−11
（1995. 1. 23）

第1回SIGALプログラミングコンテストの結果報告

浅野哲夫[1]
[1]大阪電気通信大学（寝屋川市初町）

昨年7月のアルゴリズム研究会において第1回SIGALプログラミングコンテストの実施要領を説明し、約2カ月の期限でプログラムを募集したが、今回はその結果を報告する。問題は、ランダムに生成されたn個の数の中でk番目に大きい要素を選ぶプログラムを作成することであった。結果的に画期的なアルゴリズムの提案にまでは至らなかったが、ランダムサンプリングの考え方が有効であることが数字的に示された。

Results of the First SIGAL Programming Contest

Tetsuo ASANO[1]
[1]Osaka Electro-Communication University

This article describes the results of the first SIGAL programming contest which was proposed at the SIGAL meeting, July 1994. During the two months before the deadline, we had a number of submissions. The problem was to choose the $k-$th largest element among n randomly generated numbers. Proposed algorithms were not based on surprising new ideas, but we had numerical proof that the idea of random sampling is effective.

1 コンテストの目的

近年のハードウェア技術の進歩により計算機の能力は飛躍的な向上を遂げたが、解こうとする問題の規模も従来とは比較にならないほど大規模になってきている。従来は、スーパーコンピュータやＲＩＳＣ等のハードウェア技術の進歩で問題の大規模化に対処することが多かったが、これは短眼視的な解決方法に過ぎない。

アルゴリズムの分野では、言うまでもなく、様々な計算問題に対する効率のよいアルゴリズムを開発したり、問題の難しさを解析したりするのが主要な研究テーマであり、従来から多様な問題を扱ってきているが、産業界にその成果が充分に取り入れられているとは言い難い面がある。その原因としては色々と考えられるが、単に試すだけでも難解なアルゴリズムを理解してプログラム化するという多大な労力を払わなければならないので面倒だというのが実際であろう。そこで、アルゴリズムを実行可能なプログラムの形で実現したアルゴリズム・ライブラリの構築と、各アルゴリズムの実行時間、必要メモリーなどの統計データの整備が重要であると思われる。

さて、アルゴリズム研究者は数学的に記述されたアルゴリズムの（係数を無視した）漸近的な計算量を解析するだけで、実際にどのような状況で提案しているアルゴリズムが他より有効であるのかを示そうとしない、という批判をしばしば耳にする。この批判に答え、アルゴリズム研究の重要性を社会にアピールするためには、今回は最初の試みとしてプログラムの初心者にも容易に理解でき、しかも実用的にも重要な問題についてコンテストを行うことになった。アルゴリズム研究会の立場としては、単にアルゴリズムまたはプログラムを作ることよりも、アルゴリズムを解析することの方が重要であろうが、今回はできるだけ多数の参加者を得るために面倒な解析は抜きにして、単純にプログラムの実行時間を競うことにした。

7月のアルゴリズム研究会で実施要綱を公表し、約2カ月という短い期間に応募を募ったところ、全部で14件の応募があった。当初は計算機に依存しない相対値で優劣を競うことを予定していたが、実際には計算機の環境、能力などに値がかなり依存することが判明したので、送られたプログラムをすべて同じ計算機で実行し、比較を行なった。

2 問題

コンテストの問題はセレクション問題として知られているもので、形式的に次のように記述されるものである。セレクション問題では、k番目の要素だけでなく、k番目に大きい要素以上の要素をすべて列挙することが要求されるのが普通であるが、ここでは出力の時間を少なくするために、k番目の要素だけを出力することとした。

[問題] n個の実数データの中でk番目に大きいものを出力するプログラムを作れ。

データは乱数で生成するが、データの中には同じ値が複数個存在しても構わない。したがって、上の問題をより正確に記述すると次のようになる：「与えられたn個のデータを降順に並べたとき、k番目の位置にあるデータの値を出力すること」。もちろん、データを実際に降順に並べる必要はない。正確な定義のための記述であることに注意されたい。データとすべき実数データは次節に示すCプログラムによって生成すること。また、以下の制約をすべて満たすことが条件である。[1]

制約1 言語による差をなくすため、すべてC言語にてプログラムを書くこと。

制約2 データを蓄えるためのdouble型の配列をa[ ]とする。これ以外には整数型であっても大きな（すなわち、nのオーダーの）配列は用いないこと。ただし、$o(n)$でさえあれば、たとえば、$n^{0.99}$のサイズの配列も許すものとする。

制約3 途中でデータを加工することは許されるが、最終的な答は正確に元の値に等しいこと。つまり、加工したデータを用いてアルゴリズムをガイドしてもよいが、最初に乱数で生成した値は配列に蓄えておいて、代入文以外の加減乗除などの算術演算は施さないこと。

制約4 再帰呼出は可能。その場合のスタック領域のサイズは不問とする。実行できればよい。スタックのサイズが結果的に$O(n)$であっても構わない。

制約5 特殊な関数を用いないこと。使える関数はデータ生成用のプログラムにあるもの—main(), printf(), scanf()—に限る。switch(),for(),log(),exp()などの関数はもちろん使ってもよい。禁止しているのは非常に特殊な関数である。つまり、特殊な関数を用いて高速化しはいけないということ。[2]

制約6 出力に要する時間を最小にするため、k番目に大きい値は最後に出力するが、これ以外には一切出力しないこと。

---

[1] ここに掲げた制約はアルゴリズム研究会で公表したものと若干異なるが、本質的な差はない。

[2] 実際には特殊な関数を用いたプログラムはなかったように思われるので、この制約の曖昧さは特に問題は引き起こさなかったように思われる。

3 データ生成法

データは乱数を用いて生成するが、条件を同一にするために、コンパイラに標準的に付属している乱数関数ではなく、以下に与える関数を用いること。最初の関数 rnd1() は、単純な線形合同法に基づいて一様乱数を生成するものである。2番目の関数 rnd2() は、この一様乱数を三角分布の乱数に変換したものであり、さらに3番目の rnd3() は指数分布の乱数に変換したものである。

これらの乱数生成のプログラムに関しては技術評論社から出版されている奥村晴彦著「C言語による最新アルゴリズム事典」から編集部の了解を得て、プログラムを引用させて頂いた（69069という定数の表現が同書では 69069UL となっている）。ここに記して感謝する。

```
#include <stdio.h>
#include <limits.h>
static unsigned long seed = 1;
#define rnd1() (seed*=69069,seed/(ULONG_MAX+1.0))
#define init_rnd1(x) (seed = (unsigned long)(x) | 1)
double rnd2(void)
{
   double a1,a2;
   a1 = rnd1(); a2 = rnd1();
   return a1 - a2;
{
double rnd3(void)
{
   return -log(rnd1());
}
```

(注) 2番目の乱数 rnd2() は上のように変更した。前回のプログラムではうまく行きません。これは日立の中野さんのご指摘による。また、プログラムのコンパイルには cc ではなく、gcc を用いた。

4 実行時間の計測方法

プログラムの実行時間は使用する計算機の能力とコンパイラに依存するため一概に比較するのは困難であるが、ここでは次のプログラムの実行時間を $T_1(n)$ する。乱数として rnd2(), rnd3() を用いた場合は $T_2(n), T_3(n)$ とする。

このプログラムの関数 Select() を記述するのが問題である。関数 Select() 完成後のプログラム実行時間を $T_i'(n)$ とするとき、相対値 $T_i'(n)/T_i(n)$ を用いて比較することにする。なお、m の値は時間の計測がしやすいように適当に定めるものとする。具体的には UNIX の t i m e コマンドを用いて、そのユーザー時間で測定するものとする。多少の誤差は今回は目をつぶることにした。

```
#include <stdio.h>
#include <limits.h>
#define rnd1() (seed*=69069,seed/(ULONG_MAX+1.0))
#define init_rnd1(x) (seed = (unsigned long)(x) | 1)
#define MAXSIZE 1000001
static unsigned long seed = 1;
double a[MAXSIZE];
double Select(int, int);
int n, m = 10;

void main(void)
{
   int i, j; double x;
```

```
    printf("Enter value for n ");
    scanf("%d", &n);
    for(i=0; i<m; i++){
        init_rnd1(1);
        for(j=1; j<10; j++){
            Data_generate();
            x = Select(n, n*j/10);
            printf("\n k-th = %lf", x);
        }
    }
}
double Select(int k, int n)
{
    return 1.0;
}
```

5 コンテスト実施要項

上記の制約をすべて満たすプログラムを作成し、プログラムとともに、以下に指定した n の値に対する実行時間の相対値を添えて送ってもらった。なお、期限は平成6年10月1日で、期限内に13件の応募があった。

1. $T_i'(1000)/T_i(1000)$ の値, $i = 1, 2, 3$。
2. $T_i'(10000)/T_i(10000)$ の値, $i = 1, 2, 3$。
3. $T_i'(100000)/T_i(100000)$ の値, $i = 1, 2, 3$。
4. $T_i'(1000000)/T_i(1000000)$ の値, $i = 1, 2, 3$。

6 基本的な考え方

6.1 方法1：ソーティングに基づく方法

k番目に大きいデータを求める最も直接的な方法は、当てられたデータをソートすることである。ただし、$O(n \log n)$ の時間がかかるのが難点である。

6.2 方法2：QuickPartition に基づく方法

数値データのソーティング技法としてはクイックソートが有名であるが、この方法では、列の任意の要素 x を用いて列の要素を x 以上のブロックと x 以下のブロックに分割し、それぞれを再帰的にソートする。しかし、ここでは k番目に大きい値を求めるだけでよいので、両方のブロックを処理する必要はなく、k番目に大きい値を含むブロックのみを再帰的に分割すればよい。最悪の実行時間は $O(n^2)$ であるが、入力がランダムであれば、平均的には $O(n)$ の時間で十分である。実際にコンテストの結果をみても、この方法はかなり有効である。

6.3 方法3：中央値発見の線形時間算法に基づく方法

Blum ら [1] は、n 個のデータの中央値を最悪の場合でも $O(n)$ 時間で求めるアルゴリズムを提案している。この方法では、n 個のデータを最初5個ずつのグループに分け、適当な方法でそれぞれのグループの中央値を求める。次に、それら $\lceil n/5 \rceil$ 個の中央値の中央値 Mを再帰的にもとめる。したがって、約 $n/10$ 個のグループについては、その中央値が M以下であり、それ以外の約 $n/10$ 個のグループについてはその中央値が M以上である。各グループには中央値以下の要素が3個、中央値以上の要素が3個あるから、Mに関して n 個のデータを QuickPartition の要領で

分割すると、M以下およびM以上のデータ数はそれぞれ$3n/10$以上ということになる。これは、M以上およびM以下のデータ数が高々$7n/10$であることを意味している。

M以上のデータ数とM以下のデータ数がわかれば、中央値がどちらのグループに属するかが分かるから、以後はそのグループだけを考えればよい。このように、毎回探索の対象になるのは全体の高々7割となるから、上の操作を続ければ、枝刈り探索の原理により総計算時間$O(n)$以内で処理を終えることができる。ただし、ここでは中央値ではなくk番目に大きい要素を求めるのであるから、k番目の要素を含むブロックだけを考慮すればよい。この場合も計算時間は$O(n)$である。

6.4 方法4：ランダムサンプリングに基づく方法

この方法は、一般的には次のように記述できる。

(1)n個のデータの中からランダムにm個のサンプルを取り、集合Rとする。
(2)Rの中で元のk番目に対応する要素（正確にはkm/n番目の要素）を含むある程度の幅の区間$[L, H]$を求める。
(3) 元のデータの中で、L以上の要素数n_LとH以上の要素数n_Hを求め、$n_H \leq k < n_L$でなければ、区間$[L, H]$にk番目の要素が含まれないので、(1)に戻ってやりなおす。
(4) 区間$[L, H]$に含まれるデータを列挙し、この中で$k - n_H$番目に大きい要素を求める。

このような方法をレイジーセレクトと呼ぶが、問題は集合Rのサイズと区間$[L, H]$の幅の決め方にある。文献[2]では、$|R| = n^{3/4}$と定め、区間$[L, H]$については、Rにおける$km/n - \sqrt{n}$番目と$kn/m + \sqrt{n}$番目に大きい要素によって決めている。つまり、区間の幅は、Rでは$2\sqrt{n}$、元の列では$2n^{3/4}$ということになる。文献[3]では、$|R| = n^{2/3}$とし、Rにおける区間の幅を$2n^{1/3}$と決めているが、この幅は元の列では$2n^{2/3}$となる。このように、一般的にはランダムサンプリングのサイズがステップ(3)で求める区間の幅と同じになるようにしておくと、それぞれの部分での計算時間の釣合が取れるので、計算時間の面では都合がよい。

7 応募プログラムの分類

先にも述べたように全部で１３件の応募があった。内訳は以下の通りである。

1. 阿久津さん（群馬大学）：ランダムサンプリングに基づく方法で、本文で述べた方法と異なるのは、$|R| = n^{1/2}$にしたこと、サンプルのソートは行なわないこと、区間の幅を$2n^{1/4}$にしたこと、さらにサンプリングが失敗だったときはQUickSelectionを用いることである。

2. 川戸さん（三菱電機）：最初に20個ほどのデータを見て、３種類の乱数のどれであるかを推定して、QuickPartitionにおけるpivotの選び方を変えている。

3. 匿名（三菱電機）：本質的にはQuickPartitionであるが、kの値によって分割の方法を少し変えている。

4. 森下さん（東京工大）：本質的にはQuickPartition。

5. 宇野さん（東京工大）：文献[3]の方法と同様で、$n^{2/3}$個のランダムサンプリングに基づく方法。

6. 篭谷さん（東京工大）：初期データが乱数であることを利用し、先頭の数パーセントのデータによって「kプラス（マイナス）アルファ番目」の値を推測し、これをpivotとして分割する。この「アルファ」となるマージンの決定には、いくつかのパラメータを変化させて最適なものを実験的に選んでいる。プログラムが非常に簡潔でしかも速いのが特徴。

7. 吉田さん（東芝）：クイックソートとバブルソートの併用。

8. 藤原さん（千葉大学）：データの最大値、最小値、平均を計算して、これらから k 番目の値を予測する。この値を中心とするある区間に入るデータを求めて、k 番目のデータが含まれているなら、この操作を再帰的に繰り返す。

9. 戸川さん（早稲田大学）：本質的には QuickPartition。配列の最初、中間、最後の３要素の内の２番目に大きいものを pivot として選んでいる。

10. 田内さん（岐阜大学）：本質的には QuickPartition。

11. 菱田さん（岐阜大学）：本質的には QuickPartition。

12. 中野さん（日立基礎研）：$\sqrt{n}$ 個程度のサンプルを選んで、この中で k 番目の要素に対応するものを求め、分割を行なうが、分割後にサイズの小さい方が再帰的に呼び出されるように、1/16 だけ真ん中に寄せたところの値を候補として選ぶという工夫により、再帰の深さを抑えている。

13. 喜多さん（大阪電気通信大学）：ランダムサンプリングに基づく方法であるが、$|R| = n^{1/2}$ と定め、幅は $2n^{1/4}$ に定めている。また、L と H 以外に k 番目に対応する値 M も用いて分割を行なう点が異なる。プログラムがかなり複雑なのが欠点か？

14. 浅野（大阪電気通信大学）：ランダムサンプリングに基づく方法を一般化したもの。ランダムサンプリングの失敗確率はチェビシェフの不等式を用いて推定できるが、逆に失敗確率を入力で指定するようにして、ランダムサンプリングのサイズと区間幅を最適になるように理論的に定めたもの。

8 評価結果

先にも述べたように、当初はマシンに依存しない評価基準を用いて比較を行なおうとしたが、実際にはマシンによって先に定めた相対値そのものに差が出ることが判明した。そのために、応募プログラムを再度同じ条件でコンパイルし直し、同じワークステーション上で実行した。用いたワークステーションは FUJITSU S-4/10 モデル 512（SUN の Sparc Station 10 に対応、SpecInt は 2950）である。また、コンパイラには public domain の gcc コンパイラを用いた。

応募者	$n = 1000000$	$n = 100000$	$n = 10000$	$n = 1000$
喜多	1.28	1.24	1.26	1.50
浅野	1.28	1.30	1.33	1.67
篭谷	1.29	1.26	1.26	1.50
宇野	1.33	1.31	1.31	1.50
中野	1.34	1.33	1.34	1.33
藤原	1.36	1.33	1.34	1.33
阿久津	1.39	1.41	1.43	1.67
菱田	1.58	1.54	1.18	1.50
戸川	1.62	1.56	1.49	1.50
匿名	1.64	1.59	1.52	1.67
吉田	1.64	1.56	1.51	1.50
田内	1.87	1.87	1.80	2.00
森下	2.02	1.96	2.05	2.00
川戸	2.04	1.94	1.87	1.83

表１．相対値による比較（T_i'/T_i の値）．

応募者	$n = 1000000$	$n = 100000$	$n = 10000$	$n = 1000$
喜多	12 分 52.4 秒	1 分 15.0 秒	7.7 秒	0.9 秒
浅野	12 分 52.8 秒	1 分 18.7 秒	8.1 秒	1.0 秒
笵谷	12 分 55.8 秒	1 分 16.4 秒	7.7 秒	0.9 秒
宇野	13 分 22.5 秒	1 分 19.2 秒	8.0 秒	0.9 秒
中野	13 分 28.0 秒	1 分 20.5 秒	8.2 秒	0.8 秒
藤原	13 分 41.4 秒	1 分 20.7 秒	8.2 秒	0.8 秒
阿久津	13 分 56.9 秒	1 分 25.4 秒	8.7 秒	1.0 秒
菱田	16 分 04.5 秒	1 分 33.1 秒	9.0 秒	0.9 秒
戸川	16 分 15.9 秒	1 分 34.4 秒	9.1 秒	0.9 秒
匿名	16 分 28.0 秒	1 分 36.1 秒	9.3 秒	1.0 秒
吉田	16 分 32.6 秒	1 分 34.5 秒	9.2 秒	0.9 秒
田内	18 分 48.5 秒	1 分 53.3 秒	11.0 秒	1.2 秒
森下	20 分 18.0 秒	1 分 58.6 秒	12.5 秒	1.2 秒
川戸	20 分 28.5 秒	1 分 57.5 秒	11.4 秒	1.1 秒

表2．セレクションに要した CPU 時間による比較

結果を眺めると、データ数 100 万の場合の処理時間において 12〜13 分のグループとそれ以上の時間を要するグループに分かれた。この区別は、ランダムサンプリングを用いたレージーセレクト法を基本とするグループと、QuickPartition に基づく方法との区別に相当している。なお、表には載せなかったが、(a)QuickSort による方法、(b) 中央値を求める線形アルゴリズムに基づく方法、(c) データの分布予測とバケット法に基づく方法についても実験を行なった。(a) の QuickSort による方法は、レイジーセレクトに比べて、データ数 10 万のときに既に 3〜4 倍の時間を要する。(b) の方法はソーティングよりは速いが、約 3 倍の時間を要している。ただし、各グループのソートを要素数を固定したアドホックな方法を用いると約 30%の時間を節約できることが分かった。(c) では、どのようにデータ分布を予測するかが問題であるが、比較的単純な方法でもレイジーセレクトに遜色のない結果を得ることができた。

予想されたことではあるが、ランダムサンプリングを用いたレイジーセレクト法の有効性が確認された結果に終ってしまった。セレクション問題を解くためには $2n$ 回の比較が必要であることが分かっているが、今回のアルゴリズムは 2 回の比較と 1 回のデータ交換操作に要する時間程度しか要していないことから、抜本的な改良は不可能に近いものと思われる。

9 問題点

今回のコンテストで様々な問題点が指摘された。

1. まず、筆者は浅はかにも本文で述べた相対値はマシンにあまり依存しないであろうと楽観視していたが、実際には、パソコンとワークステーションではかなりの差が出た。また、ワークステーションでも RISC 系と CISC 系では差があり、結局、同じ計算機の上で実行して比較を行なった。

2. 乱数発生の時間の方がセレクションのための時間よりずっと長かったが、これは long int の乗算が double float の比較より圧倒的に時間を要するためではないかと思われる（日立の中野氏の指摘）。

3. 結局、アルゴリズム的に斬新な考え方は出てこず、プログラミング技術のコンテストの様相を呈してしまったのは残念である。実際、配列に対するアクセスにポインタを用いると、結構高速化が可能であった。

4. データに単純な乱数を用いているが、この知識を利用すると高速化ができてしまうので問題だという声が多かった。セレクションの難しさをコントロールするパラメータを取り入

れたデータの生成方法もあるが、時間がかかるので今回は採用しなかったが、重要な点ではあると思われる。

10 結論

セレクション問題に関するプログラミングコンテストの結果を報告した。結果的に全く斬新なアルゴリズムが提案されたわけではないが、13 件の応募があり、アルゴリズムの分野にも計算のオーダーだけでなく実際的な計算時間にも興味をもつ研究者が多いことがわかったことは成果の一つであると言えよう。もっと計算複雑度の高い問題についてコンテストを行なう方が差が出て面白いのではないかという意見もあり、次回の問題設定において考慮したい。

謝辞

本報告をまとめるにあたり、計算機実験を実行してくれた卒業研究生の池田智彦君に感謝します。また、プログラミングに際して助言をもらった修士課程学生の喜多富男君に感謝します。

References

[1] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan: "Time Bounds for Selection,", *J. Computer and System Science*, vol.7, pp.448-461, 1972.

[2] P. Raghavan: "Lecture Notes on Randomization Algorithms," Research Report, RC 15340 (#68237), IBM T. J. Watson Research Center, 1990.

[3] M. A. Weiss: "Data Structures and Algorithm Analysis," *The Benjamin/Cummings Publ. Co. Inc.*, 1992.