サーチ

山本昌志*

2005 年 11 月 14 日

1　前回の復習と本日の学習内容

1.1　前回の復習

前回までは，データを並び替えるソートについて学習した．学習したソートのアルゴリズムは，バブルソート，クイックソート，マージソート，コームソート，単純挿入ソート，2 分挿入ソートである．

1.2　本日の学習内容

本日は，サーチ (search:探索あるいは検索) について学習する．教科書 [1] に書いてあるとおり，サーチとは，

> 文字どおりたくさんのデータの中から目的のデータがどこにあるか (もしくは，あるかないか) を調べる作業です．

である．本日の講義では，整数のデータが配列に格納されている場合について，サーチを学ぶ．どこにあるかは，配列の添え字で示すことになる．

サーチの方法はいろいろがあるが，ここでは，

- リニアサーチ
- バイナリーサーチ

について学習する．

2　リニアサーチ

リニアサーチ (linear search) は，逐次検索あるは順検索 (sequential search) とも呼ばれ，配列などに格納されたデータを一つ一つ順に端から調べるだけの，素朴な方法である．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

2.1 単純な方法

2.1.1 原理

原理は簡単で，配列の端から比較を行い，目的の整数を捜しているだけである．もう少し詳しい説明は，教科書に書かれている．もし，配列が整理されておらずランダムに並んでいる場合，この方法しか無い．

このリニアサーチの計算量のオーダーを考えよう．単純なリニアサーチのプログラムは，教科書の List 2-1(p.37) に示されているとおりで，同じものをプリントのリスト 1 に載せてある．このプログラムで，もっとも多く実行される，すなわちもっとも多くの時間がかかる命令は，リスト 1 の 13 行目

```
n<num&&a[n]!=x
```

である．もし，n 個のデータがあり，その中に目的のデータ (キー) がある場合，この命令の平均実行回数は $n/2$ である．データの中にキーが無い場合，n 回，その命令は実行される．データの中にキーの有無で平均実行回数は変わるが，いずれにしても計算量のオーダーは $O(n)$ である．

2.1.2 フローチャート

教科書の List 2-1(p.37) あるいはこのプリントのリスト 1 のプログラムのフローチャートを図 1 に示す．

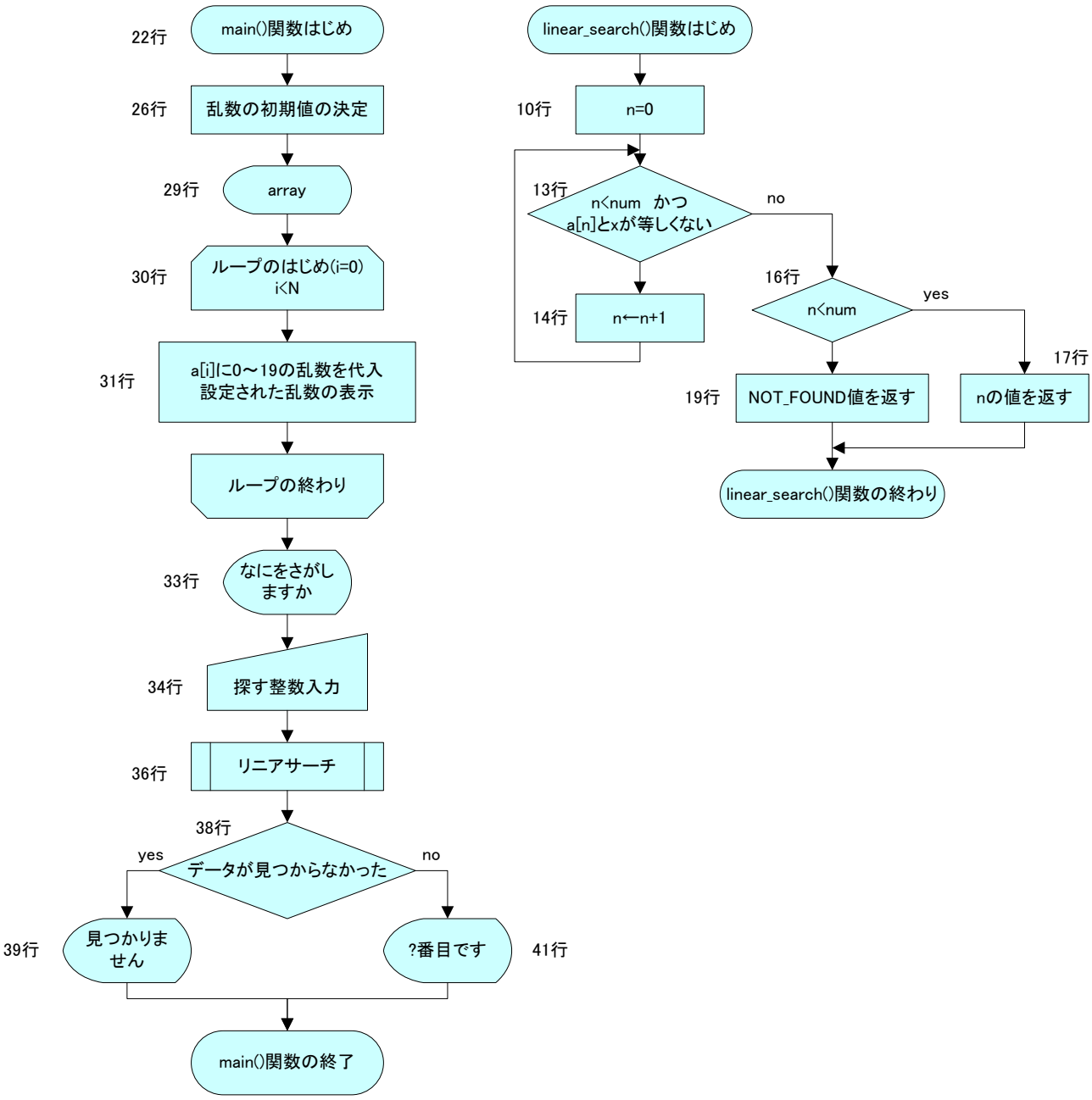

図 1: 単純なリニアサーチのフローチャート.

2.1.3 プログラム

単純なリニアサーチのプログラムをリスト 1 に示す．これは，教科書の List 2-1(p.37-38) と同じである．

リスト 1: リニアサーチ

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define NOT_FOUND    (-1)
6  #define N            (10)
7
8  int linear_search(int x,int *a,int num)
9  {
10     int n=0;
11
12     /* 配列の範囲内で目的の値を探す */
13     while(n<num&&a[n]!=x)
14         n++;
15
16     if(n<num)
17         return n;
18
19     return NOT_FOUND;
20 }
21
22 int main(void)
23 {
24     int i,r,array[N];
25
26     srand((unsigned int)time(NULL));
27
28     /* 適当な配列を作る */
29     printf("array ");
30     for(i=0;i<N;i++)
31         printf("[%d]:%d ",i,array[i]=rand()%20);
32
33     printf("\n何を探しますか:");
34     scanf("%d",&i);
35
36     r=linear_search(i,array,N);
37
38     if(r==NOT_FOUND)
39         printf("%dは見つかりません\n",i);
40     else
41         printf("%dは%d番目です\n",i,r);
42
43     return EXIT_SUCCESS;
44 }
```

2.2 番兵を使う方法

2.2.1 原理

最初に示したリスト 1 のプログラムは，ちょっとした工夫で実行速度を向上させることができる．リスト 1 のもっとも計算時間がかかる 13 行目は，2 つの比較 (`n<num` と `a[n]!=x`) を行っている．キーを配列の最後に入れる (番兵) ことにより，`n<num` の比較の計算を行わないで済むようにできる．こうすると配列の最後では，`a[n]!=x` が成立しなくなるので，ループから抜けることになる．ループから抜けた後，キーと元々あった配列の最後の値と比較すればよいのである．

このようにすると，比較の数が半分になる．しかし，計算量のオーダーは $O(n)$ と変わらないことに注意しよう．

2.2.2 フローチャート

番兵を使うリニアサーチのプログラムである教科書の List 2-2(p.38-39) あるいはこのプリントのリスト 2 のプログラムのフローチャートを図 2 に示す．番兵をどのように処理しているか，理解せよ．

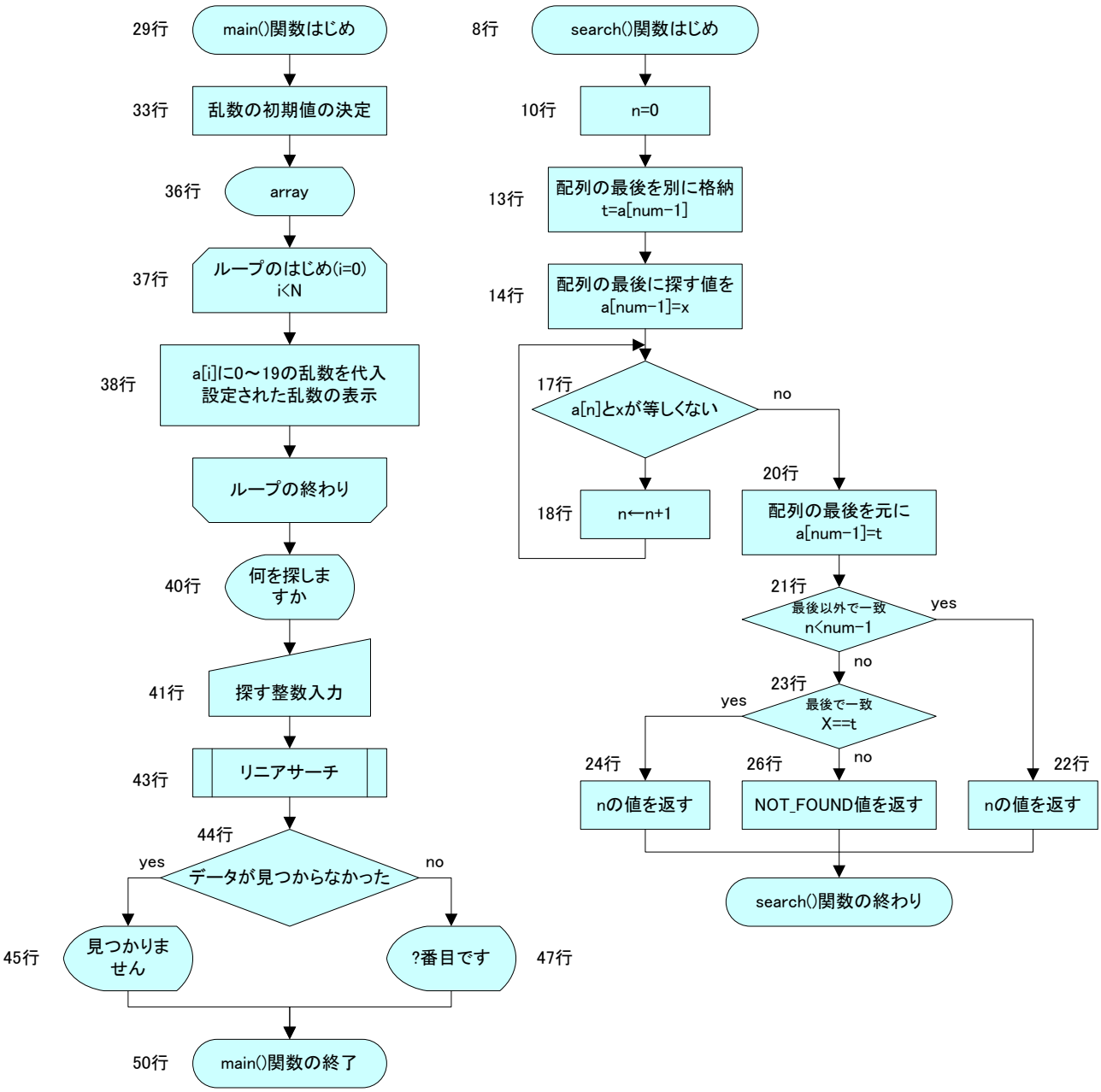

図 2: 番兵を使うリニアサーチのフローチャート.

2.2.3 プログラム

番兵を使うリニアサーチのプログラムをリスト 2 に示す．これは，教科書の List 2-2(p.38-39) と同じである．

リスト 2: リニアサーチ．番兵を使う方法．

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NOT_FOUND    (-1)
#define N            (10)

int search(int x,int *a,int num)
{
    int      n=0,t;

    /* 最後の値をxに入れ替える（番兵） */
    t=a[num-1];
    a[num-1]=x;

    /*目的の値を探す*/
    while(a[n]!=x)
        n++;

    a[num-1]=t;      /* 配列最後の値を元に戻す */
    if(n<num-1)
        return n;    /* いちばん最後以外で一致 */
    if(x==t)
        return n;    /* いちばん最後が一致 */

    return NOT_FOUND;
}

int main(void)
{
    int      i,r,array[N];

    srand((unsigned int)time(NULL));

    /* 適当な配列を作る */
    printf("array ");
    for(i=0;i<N;i++)
        printf("[%d]:%d ",i,array[i]=rand()%20);

    printf("\n何を探しますか:");
    scanf("%d",&i);

    r=search(i,array,N);
    if(r==NOT_FOUND)
        printf("%dは見つかりません\n",i);
    else
        printf("%dは%d番目です\n",i,r);

    return EXIT_SUCCESS;
}
```

3 バイナリーサーチ

リニアサーチの欠点は，計算時間がかかることである．データの並びに規則性が無い場合，リニアサーチのように端から順に捜すしかない．しかし，データの並びに規則性がある場合，その性質を利用できる．

たとえば，データが昇順に並んでいた場合，端から比較するのではなく，最初に真ん中に位置するデータ

と比較する．すると，サーチするデータの個数がいっぺんに半分になる．同じことを繰り返すと，比較すべき対象が 1/2 ずつ減少する．データの個数が多い場合，この方法は劇的にサーチが早くなる．この方法をバイナリーサーチと言う．

リニアサーチだと，1 回の比較で 1 個しかデータの候補が減らないのに，バイナリーサーチだと $n/2$ 個減少させることができるのである．ただし，バイナリーサーチを使うためには，データを予めソートしておく必要がある．サーチの回数が 1 回であれば，ソートの手間を考えるとリニアサーチの方が良い．サーチの回数が増加すると，バイナリーサーチの方が有利になる．

3.1 通常の方法

3.1.1 原理

原理は単純で，データが昇順に並んでいる場合を考えれば理解できる．キーのある範囲を $[left, right]$ とする．これは，配列 `a[left]` と `a[right]` の間が候補で，この中にキーがある可能性があるということである．この範囲外には，キーと一致するデータは絶対に無いのである．データが n 個ある場合，`a[0]` から `a[n-1]` の範囲が候補なので，$[0, n-1]$ の間にあると表現するのである．

つぎに，$mid = (left + right)/2$ とキー `x` を比較する．この比較の結果，

- `a[mid]` とキー `x` が等しければ，キー `x` のある位置は mid である．
- `a[mic]` がキー `x` よりも小さければ，$[mid + 1, right]$ にキー `x` と一致するデータがある可能性がある．
- `a[mic]` がキー `x` よりも大きければ，$[left, mid - 1]$ にキー `x` と一致するデータがある可能性がある．

が言える．これを繰り返すことにより，キーと一致するデータのある場所を捜す．

1 回の計算，リスト 3 の 12〜22 行のループで，検索する範囲は 1/2 になる．したがって，α 回のループでその範囲が 1 になれば計算が終了する．データの個数を n として，式で表すと，

$$n\left(\frac{1}{2}\right)^{\alpha} = 1 \tag{1}$$

となる．これから，計算回数 α は，

$$\alpha = \log_2 n \tag{2}$$

と導き出せる．バイナリーサーチの場合の計算量のオーダーは，$O(\log_2 n)$ である．

3.1.2 フローチャート

単純なバイナリーサーチのプログラムである教科書の List 2-3(p.41-42) あるいはこのプリントのリスト 3 のプログラムのフローチャートを図 3 に示す．

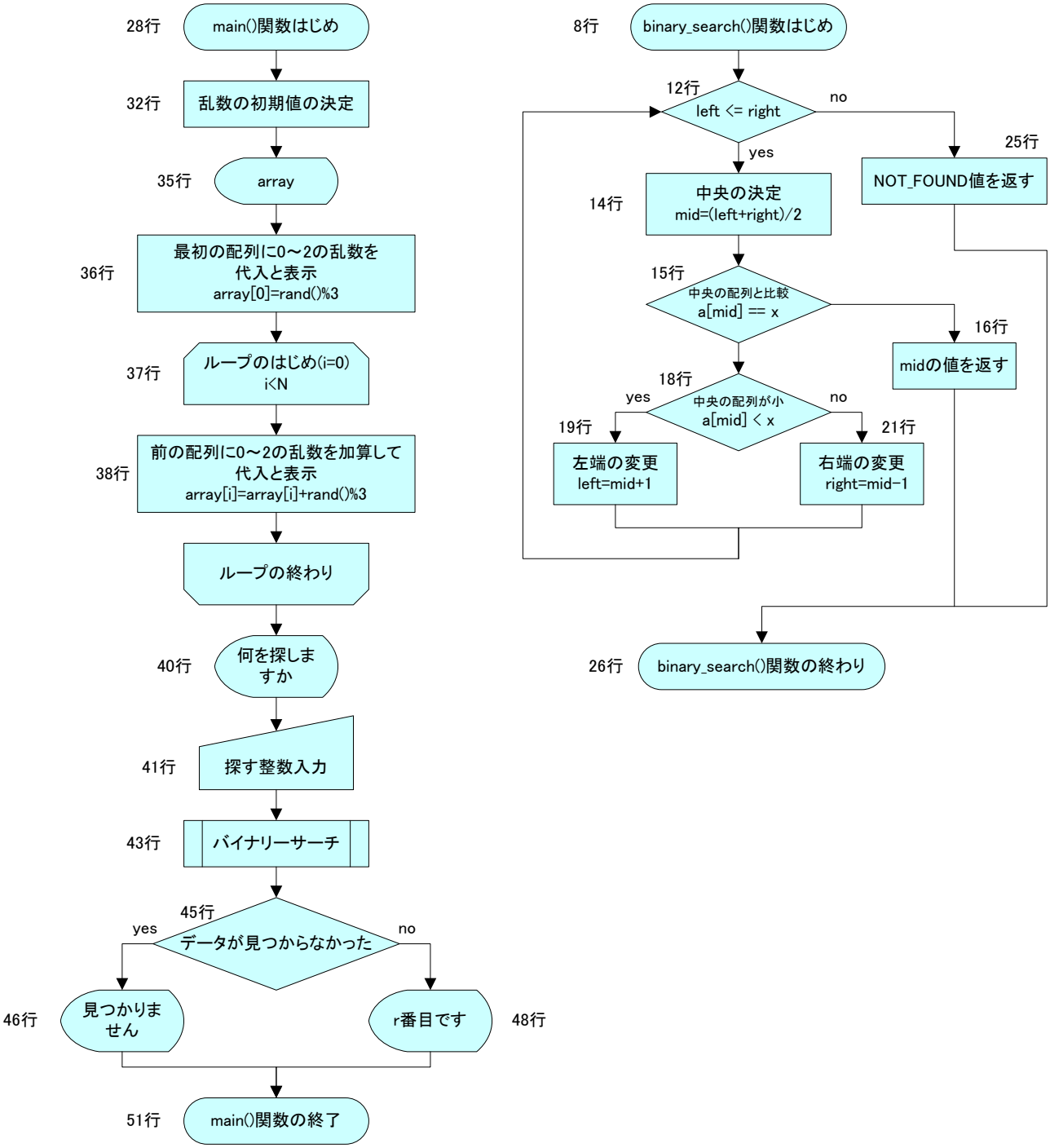

図 3: 単純なバイナリーサーチ.

3.1.3 プログラム

単純なバイナリーサーチのプログラムをリスト 3 に示す．これは，教科書の List 2-3(p.41-42) と同じである．

リスト 3: バイナリーサーチ.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define NOT_FOUND    (-1)
6  #define N            (10)
7
8  int binary_search(int x,int *a,int left,int right)
9  {
10     int      mid;
11
12     while(left<=right)
13     {
14         mid=(left+right)/2;
15         if(a[mid]==x)
16             return mid;
17
18         if(a[mid]<x)
19             left=mid+1;      /* midより左側にxは存在しない */
20         else
21             right=mid-1;     /* midより右側にxは存在しない */
22     }
23
24     /* サーチ範囲がなくなっても一致するものはなかった */
25     return NOT_FOUND;
26 }
27
28 int main(void)
29 {
30     int      i,r,array[N];
31
32     srand((unsigned int)time(NULL));
33
34     /* 適当なソートされた配列を作る */
35     printf("array ");
36     printf("[0]:%d ",array[0]=rand()%3);
37     for(i=1;i<N;i++)
38         printf("[%d]:%d ",i,array[i]=array[i-1]+rand()%3);
39
40     printf("\n何を探しますか:");
41     scanf("%d",&i);
42
43     r=binary_search(i,array,0,N-1);
44
45     if(r==NOT_FOUND)
46         printf("%dは見つかりません\n",i);
47     else
48         printf("%dは%d番目です\n",i,r);
49
50     return EXIT_SUCCESS;
51 }
```

3.2 先頭を返す方法

3.2.1 原理

キーと一致するデータが複数ある場合，ソートされているので，それらは配列で同じ値が続くことになる．この場合，リスト 3 だとそれらのうち最初に一致した位置を返すことになる．どれに一致するかは，データに依存する．このように複数と一致する場合は，その先頭の位置を知りたいことがある．

このように，先頭を知りたい場合は必ずとなり通しと比較するようにすればよい．そのようにするためには，3 を改良して，4 のようにすれば良い．

3.2.2 フローチャート

同じ値が続く場合にその先頭の位置を返すバイナリーサーチのプログラムである教科書の List 2-4(p.44-45) あるいはこのプリントのリスト 4 のフローチャートを図 4 に示す．

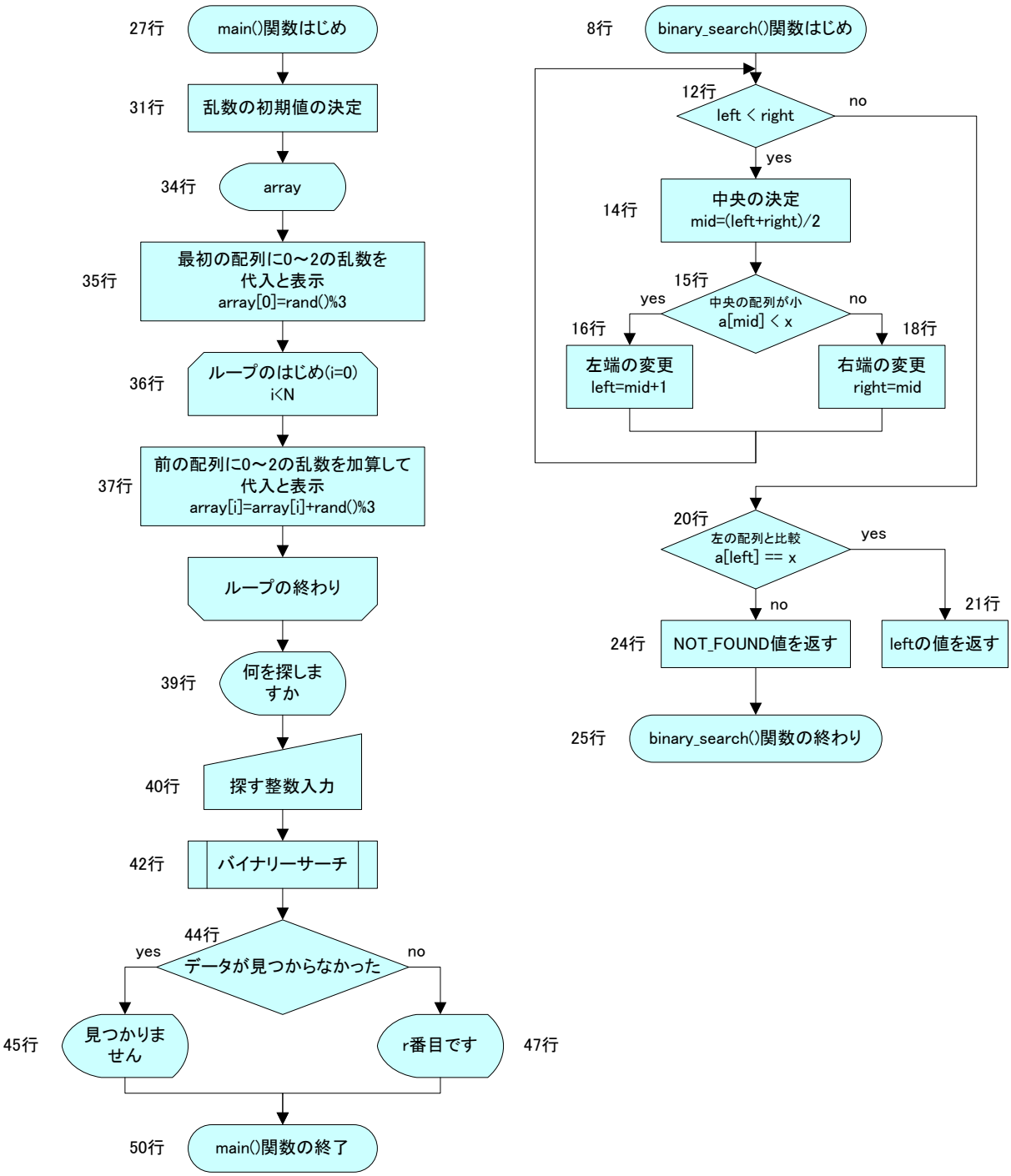

図 4: 一致する配列の先頭を返すバイナリーサーチのフローチャート.

3.2.3 プログラム

一致するデータが複数の場合，もっとも先頭に近い位置を返すバイナリーサーチのプログラムをリスト 4 に示す．これは，教科書の List 2-4(p.44-45) と同じである．

リスト 4: バイナリーサーチ．同じ値が続く場合にその先頭の位置を返す．

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define NOT_FOUND    (-1)
6  #define N            (10)
7
8  int binary_search(int x,int *a,int left,int right)
9  {
10     int       mid;
11
12     while(left<right)
13     {
14         mid=(left+right)/2;
15         if(a[mid]<x)
16             left=mid+1;
17         else
18             right=mid;
19     }
20     if(a[left]==x)
21         return left;
22
23     /* サーチ範囲がなくなっても一致するものはなかった */
24     return NOT_FOUND;
25 }
26
27 int main(void)
28 {
29     int       i,r,array[N];
30
31     srand((unsigned int)time(NULL));
32
33     /* 適当なソートされた配列を作る */
34     printf("array ");
35     printf("[0]:%d ",array[0]=rand()%3);
36     for(i=1;i<N;i++)
37         printf("[%d]:%d ",i,array[i]=array[i-1]+rand()%3);
38
39     printf("\n何を探しますか:");
40     scanf("%d",&i);
41
42     r=binary_search(i,array,0,N-1);
43
44     if(r==NOT_FOUND)
45         printf("%dは見つかりません\n",i);
46     else
47         printf("%dは%d番目です\n",i,r);
48
49     return EXIT_SUCCESS;
50 }
```

4　課題

4.1　課題内容

4.1.1　リニアサーチ

以下の数列のリニアサーチに関する問題である.

```
752 778 608 239 956 244 535 840 629 353
```

[**問 1**]　リスト 1 で，535 を捜す場合の比較の対象を順に示せ.

[**問 2**]　リスト 1 で，643 を捜す場合の比較の対象を順に示せ.

[**問 3**]　リスト 2 で，535 を捜す場合の比較の対象を順に示せ.

[**問 4**]　リスト 2 で，643 を捜す場合の比較の対象を順に示せ.

4.1.2　バイナリーサーチ

以下の数列のバイナリーサーチに関する問題である.

```
239 244 353 535 608 629 752 752 752 956
```

[**問 1**]　リスト 3 で，650 を捜す場合の比較の対象を順に示せ.

[**問 2**]　リスト 3 で，752 を捜す場合の比較の対象を順に示せ.

[**問 3**]　リスト 4 で，650 を捜す場合の比較の対象を順に示せ.

[**問 4**]　リスト 4 で，752 を捜す場合の比較の対象を順に示せ.

4.2　レポート 提出要領

提出方法は、次の通りとする。

期限	11 月 21 日 (月) AM 10:40
用紙	A4
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて、以下の項目を分かりやすく記述すること。 授業科目名「情報工学」 課題名「課題　サーチ」 2E　学籍番号　氏名 提出日
内容	2 ページ以降に問いに対する答えを分かりやすく記述すること.

参考文献

[1] 春日伸弥紀平拓男. プログラミングの宝箱　アルゴリズムとデータ構造. ソフトバンクパブリッシング(株), 2004 年.