「アルゴリズムとデータ構造」資料３
線形探索と二分探索

鴨浩靖

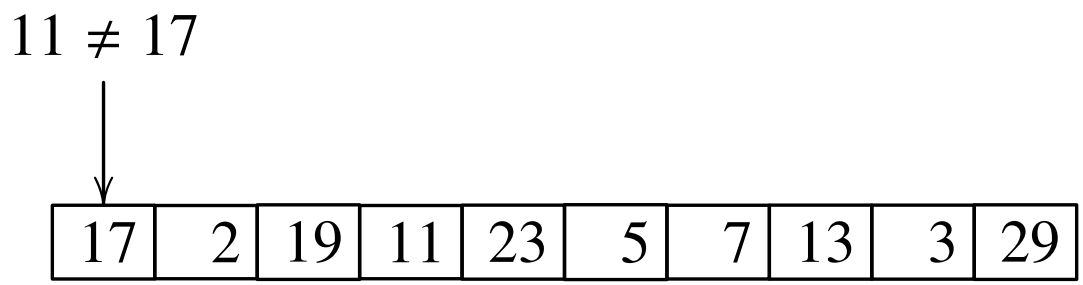
2009 年 10 月 20 日 初版
2011 年 10 月 11 日 第二版
2013 年 10 月 11 日 第三版
2018 年 10 月 29 日 第四版

線形探索

探索範囲の端から順にみつけたいものと比較することを、みつかるか終端に達するまで繰り返すアルゴリズムを、線形探索と呼ぶ。
探索範囲の要素の並び方に制約はない。

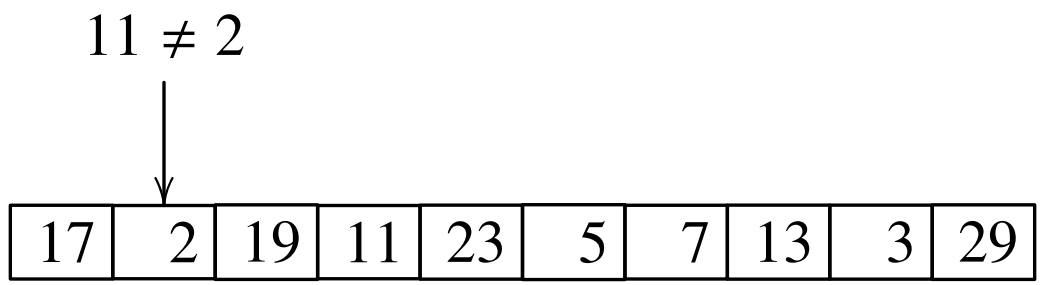
線形探索の動作の例

11 を探す [1/5]



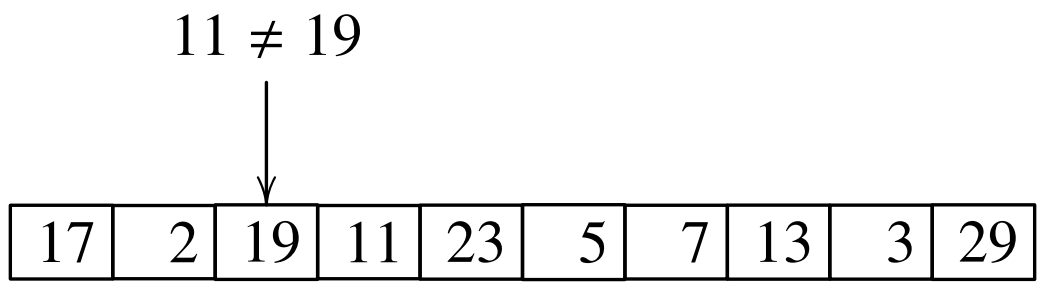
線形探索の動作の例

11 を探す [2/5]



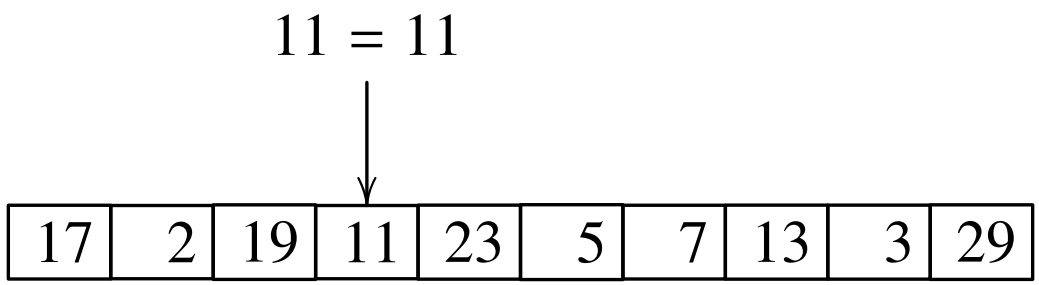
線形探索の動作の例

11 を探す [3/5]



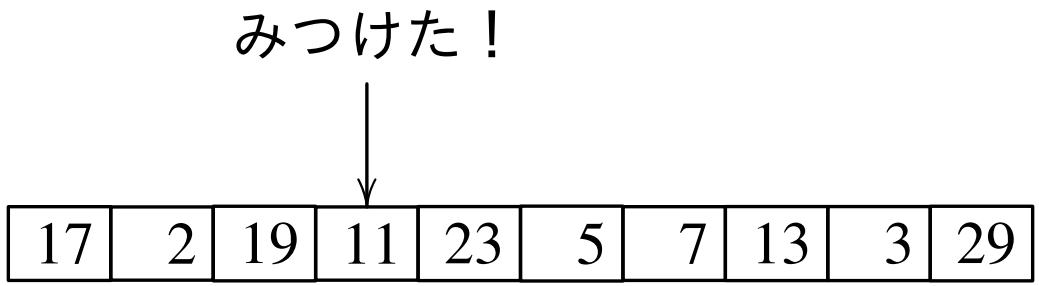
線形探索の動作の例

11 を探す [4/5]



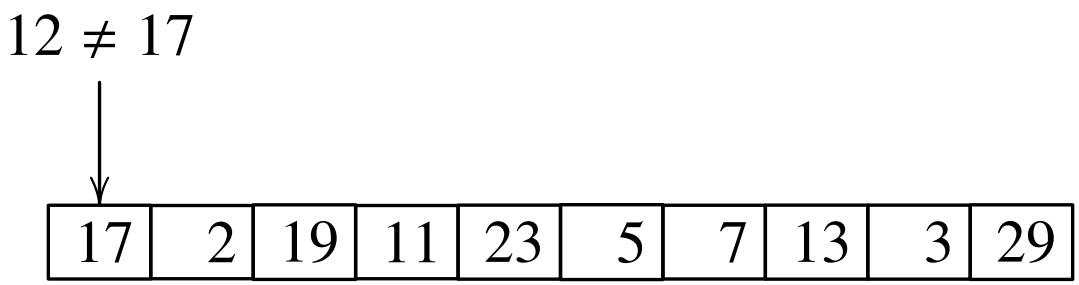
線形探索の動作の例

11 を探す [5/5]



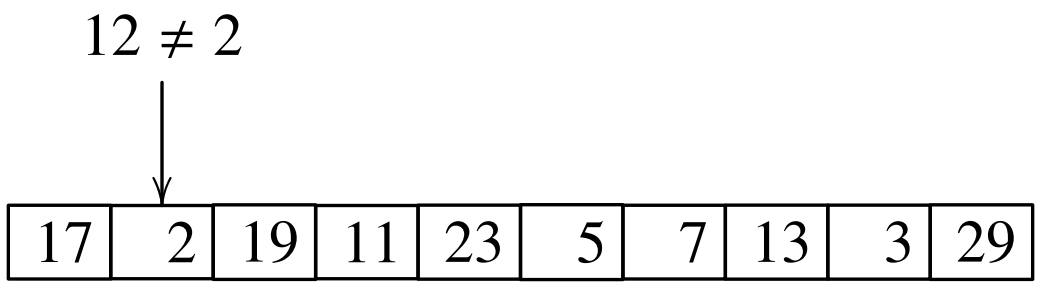
線形探索の動作の例

12 を探す [1/12]



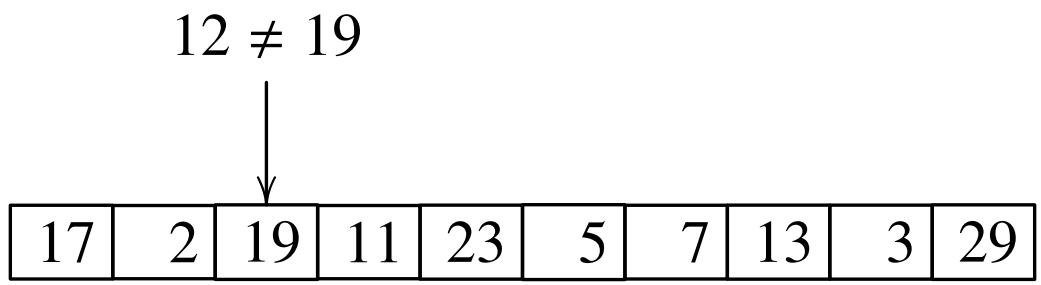
線形探索の動作の例

12 を探す [2/12]



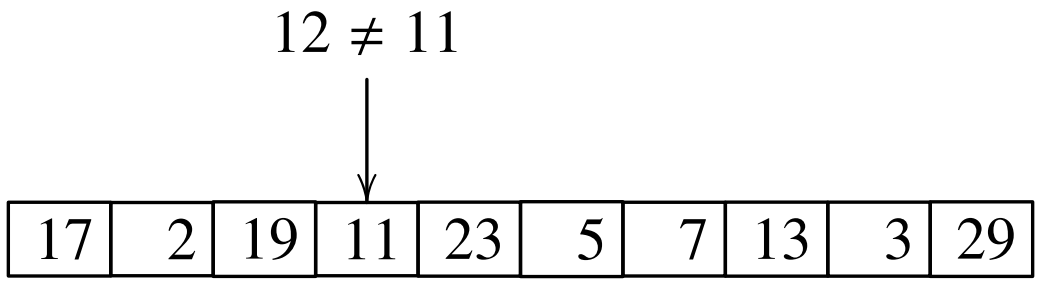
線形探索の動作の例

12 を探す [3/12]



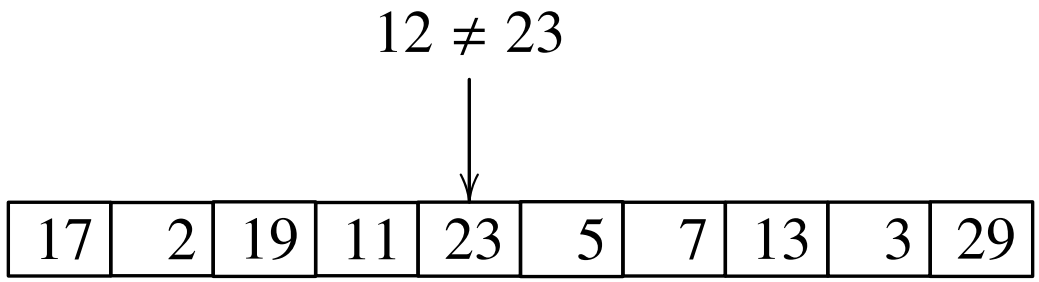
線形探索の動作の例

12 を探す [4/12]



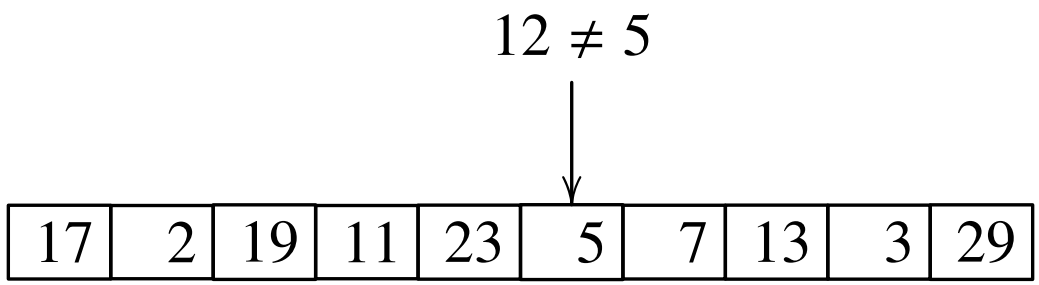
線形探索の動作の例

12 を探す [5/12]



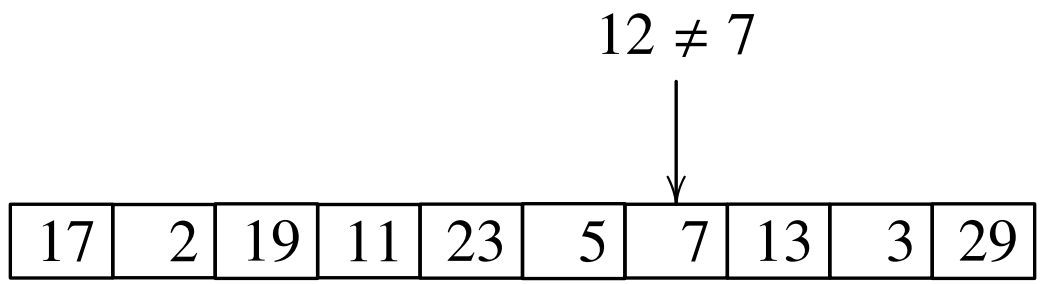
線形探索の動作の例

12 を探す [6/12]



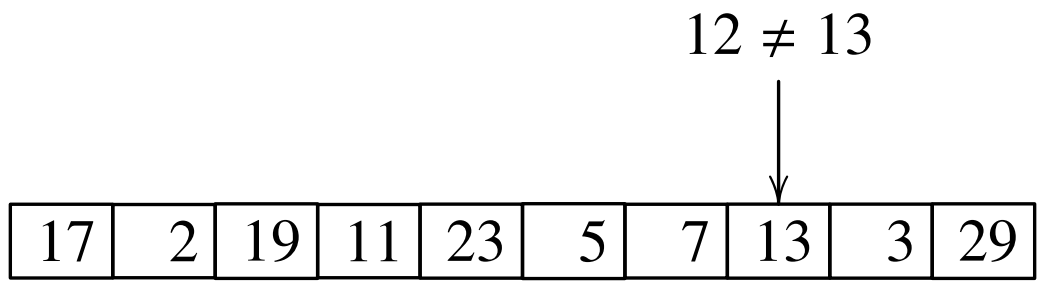
線形探索の動作の例

12 を探す [7/12]



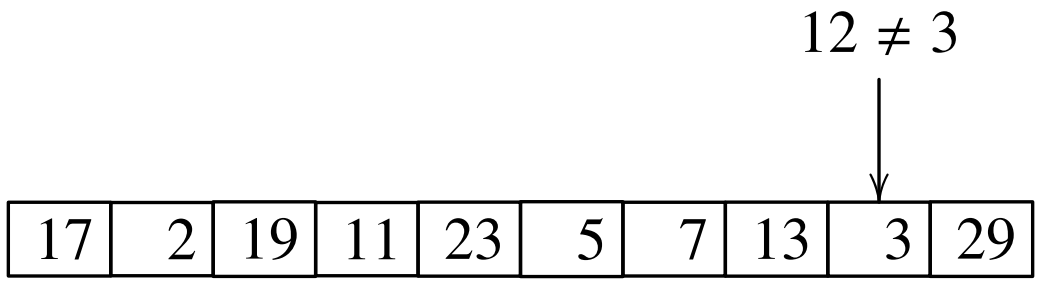
線形探索の動作の例

12 を探す [8/12]



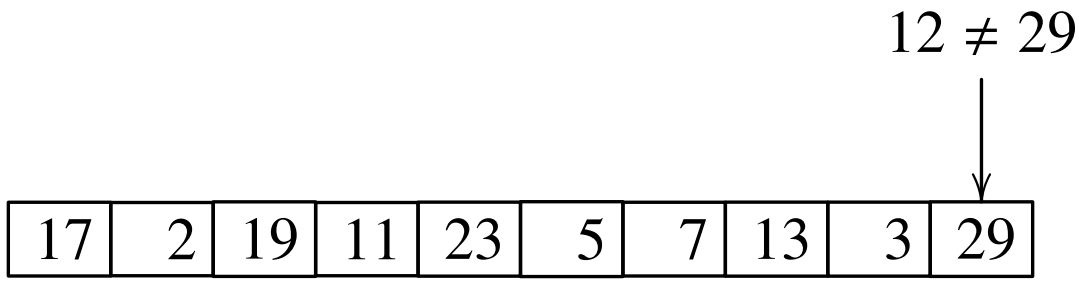
線形探索の動作の例

12 を探す [9/12]



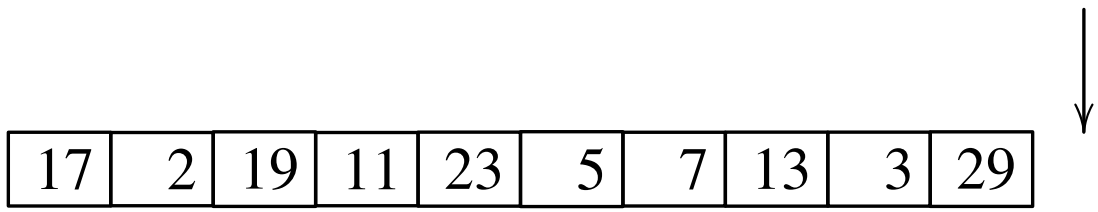
線形探索の動作の例

12 を探す [10/12]



線形探索の動作の例

12 を探す [11/12]



線形探索の動作の例

12 を探す [12/12]

なかった！

17	2	19	11	23	5	7	13	3	29

線形探索の素朴な実装例

```
#include <stdlib.h>

int *
search_key(int array[], size_t size, int key)
{
        size_t  i;
        for (i = 0; i < size; i ++) {
                if (array[i] == key)
                        return &array[i];
        }
        return NULL;
}
```

線形探索の計算量

見つからなかった場合

引数 `size` の値を n として、この関数の実行時間を見積る。
みつからなかった場合の計算時間は、

$$a(n+1) + bn + c_1 n + c_2$$

と見積もることができる。ただし、

- a 添字の比較 1 回あたりの計算時間
- b 要素の比較 1 回あたりの計算時間
- c_1 ループの中でのその他のもろもろの処理 1 回あたりの計算時間の計
- c_2 その他のもろもろの処理に必要な計算時間の計

たいていの場合、a, b, c_1, c_2 とも、データによらず、ほぼ一定とみなして良い。

線形探索の計算量

見つかった場合

引数 `size` の値を n として、この関数の実行時間を見積る。
第 i 要素でみつかった場合の計算時間は、

$$a(i+1) + b(i+1) + c_1 i + c_2$$

と見積もることができる。ただし、

- a 添字の比較 1 回あたりの計算時間
- b 要素の比較 1 回あたりの計算時間
- c_1 ループの中でのその他のもろもろの処理 1 回あたりの計算時間の計
- c_2 その他のもろもろの処理に必要な計算時間の計

たいていの場合、a, b, c_1, c_2 とも、データによらず、ほぼ一定とみなして良い。

線形探索の計算量

最悪計算量

最悪の場合は、見つからなかったとき。
最悪の計算時間は

$$a(n+1) + bn + c_1 n + c_2$$

と見積もることができる。

線形探索の計算量

平均計算量

計算を簡単にするために次のように仮定する。

- みつかる確率は α である。
- みつかる場合、配列のどの場所もみつかる確率は等しい。

すると、平均の計算時間は、

$$\frac{\alpha}{n}\sum_{i=0}^{n-1}(a(i+1)+b(i+1)+c_1 i+c_2)+(1-\alpha)(a(n+1)+bn+c_1 n+c_2)$$

と見積もることができる。

線形探索の計算量

オーダー

計算すると

$$\text{最悪計算時間} = (a + b + c_1)n + a + c_2$$

$$\text{平均計算時間} = \left(1 - \frac{\alpha}{2}\right)(a + b + c_1)n + \frac{\alpha}{2}(a + b - c_1) + (1 - \alpha)(a + c_2)$$

細かい部分には興味ないので、

最悪計算時間 $O(n)$

平均計算時間 $O(n)$

で良い。

番兵を使った線形探索の高速化

配列に書き込みが可能な場合は、探索前に探したい値を配列の末尾に書き込むことで、範囲チェックを省いて高速化できる。
この技法で、末尾に書き込むデータを番兵と呼ぶ。

番兵を使った線形探索の実装例

```
#include <stdlib.h>

int *
search_key(int array[], size_t size, int key)
{
        size_t  i;
        array[size] = key;
        i = 0;
        while (array[i] != key) {
                i ++;
        }
        return i < size ? &array[i] : NULL;
}
```

番兵を使った場合の線形探索の計算量

最悪

引数 size の値を n として、この関数の実行時間を見積る。
最悪の場合は、みつからなかった場合。その時の計算時間は、

$$a + b(n+1) + c_1' n + c_2'$$

と見積もることができる。ただし、

- a 添字の比較 1 回あたりの計算時間
- b 要素の比較 1 回あたりの計算時間
- c_1' ループの中でのその他のもろもろの処理 1 回あたりの計算時間の計
- c_2' その他のもろもろの処理に必要な計算時間の計

$c_1' < c_1$ であるので、番兵を使わない場合よりも比例定数が小さくなって、計算時間は短くなる。
でも、オーダーは変わらない。

番兵を使った場合の線形探索の計算量（つづき）

平均

平均計算時間は、同じ仮定の下で、

$$\frac{\alpha}{n}\sum_{i=0}^{n-1}(a+b(i+1)+c_1'i+c_2')+(1-\alpha)(a+b(n+1)+c_1'n+c_2')$$

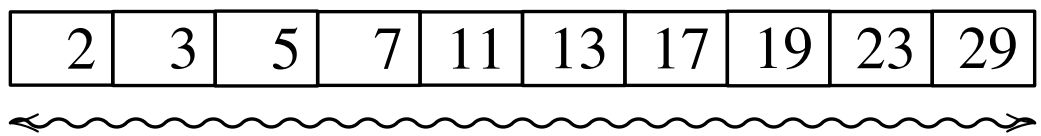
と見積もることができる。
$c_1' < c_1$ であるので、番兵を使わない場合とオーダーは変わらないが、比例定数が小さくなって平均計算時間も短くなる。

二分探索

探索範囲の真ん中あたりとみつけたいものとを比較し、その結果によって探索範囲を約半分に縮小することを、みつかるか探索範囲がなくなるまで繰り返すアルゴリズムを、二分探索と呼ぶ。
探索範囲の要素があらかじめ大きさの順に並んでいることが前提。

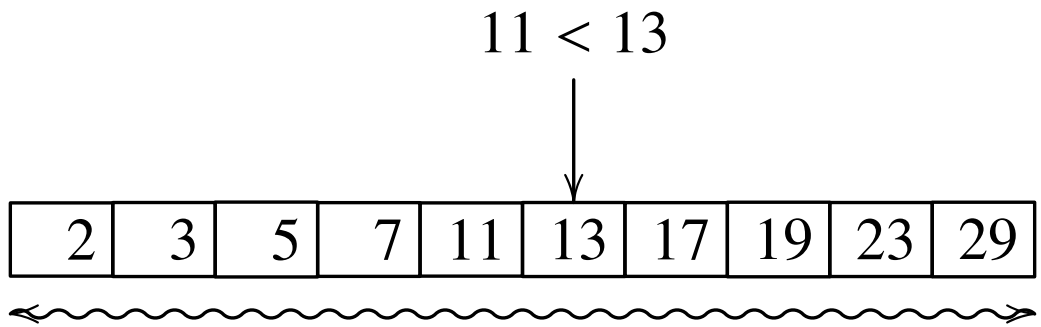
二分探索の動作の例

11 を探す [1/7]



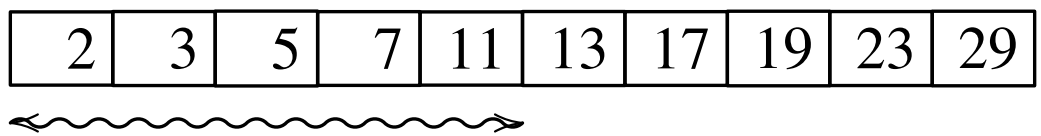
二分探索の動作の例

11 を探す [2/7]



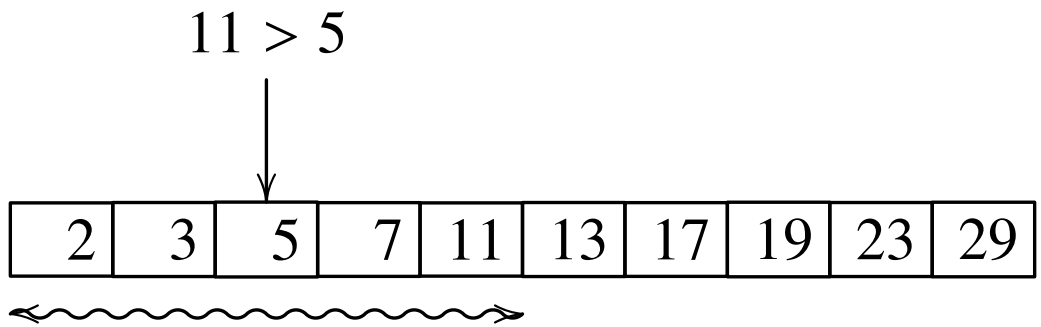
二分探索の動作の例

11 を探す [3/7]



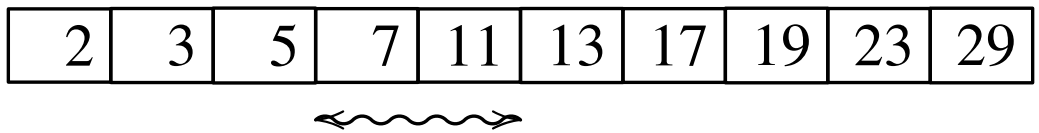
二分探索の動作の例

11 を探す [4/7]



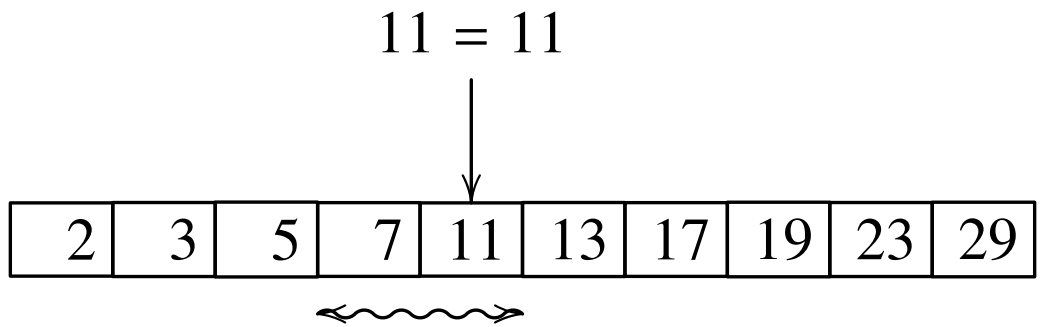
二分探索の動作の例

11 を探す [5/7]



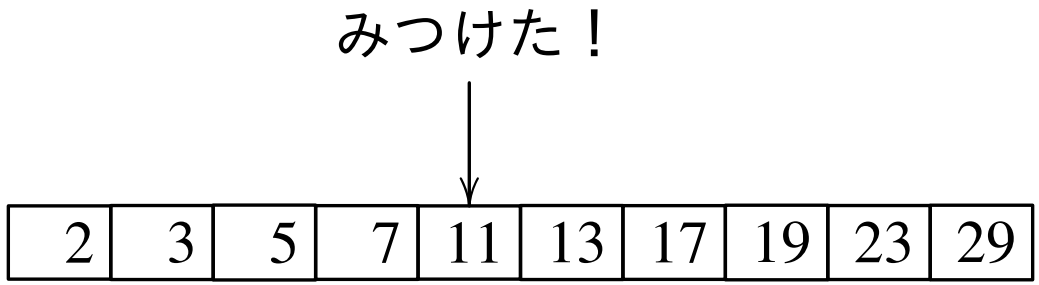
二分探索の動作の例

11 を探す [6/7]



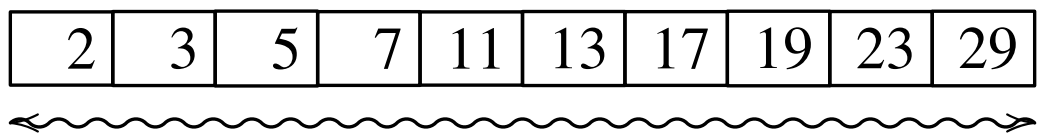
二分探索の動作の例

11 を探す [7/7]



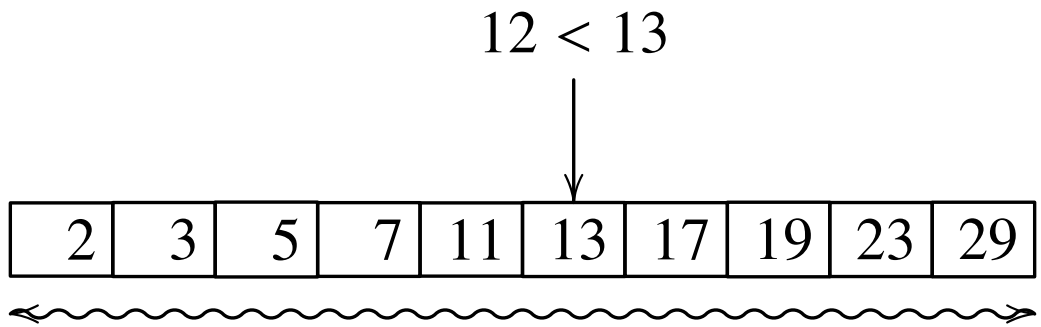
二分探索の動作の例

12 を探す [1/8]



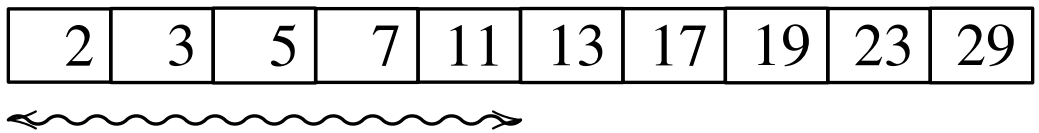
二分探索の動作の例

12 を探す [2/8]



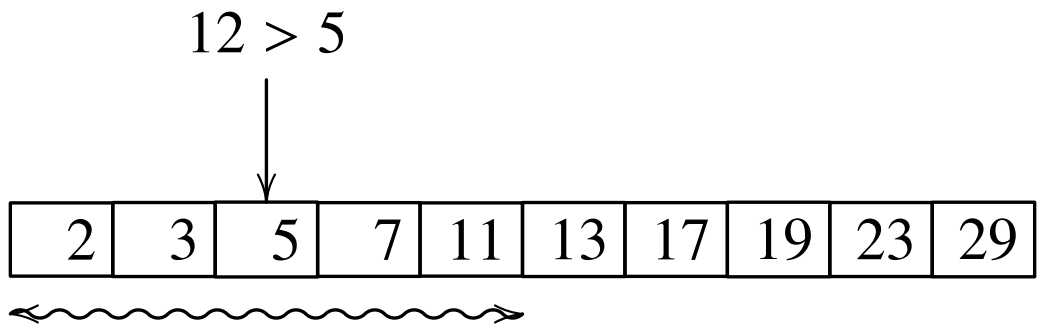
二分探索の動作の例

12 を探す [3/8]



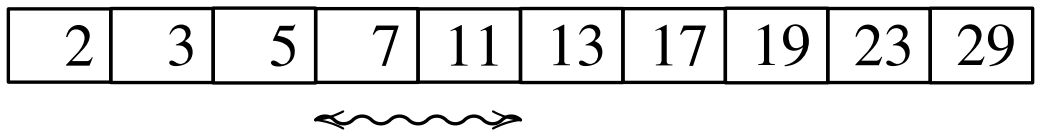
二分探索の動作の例

12 を探す [4/8]



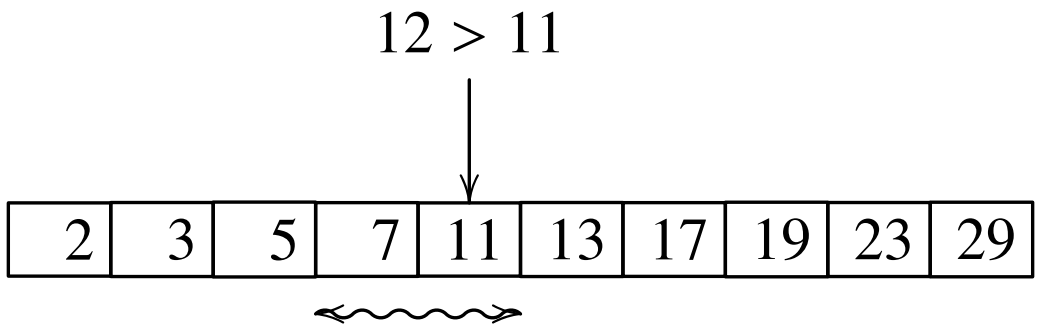
二分探索の動作の例

12 を探す [5/8]



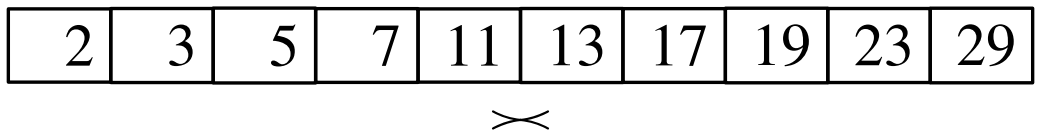
二分探索の動作の例

12 を探す [6/8]



二分探索の動作の例

12 を探す [7/8]



二分探索の動作の例

12 を探す [8/8]

なかった！

2	3	5	7	11	13	17	19	23	29

二分探索の実装例

```
int *
search_key(int array[], size_t size, int key)
{
        size_t  bottom, middle;
        for (bottom = 0; size > 0; size /= 2) {
                middle = bottom + size / 2;
                if (array[middle] == key)
                        return &array[middle];
                if (array[middle] < key) {
                        bottom = middle + 1; size --;
                }
        }
        return NULL;
}
```

二分探索の計算量

引数 `size` の値を n として、この関数の実行時間を見積る。
1 回ループを回るたびに探索範囲が約半分になることに注目すると、探索範囲の幅が 1 になるまで、およそ $\log_2 n$ 回、ループを回る。
したがって、厳密な式を求めることは困難だが、最悪の計算時間が $O(\log n)$ であることは間違いない。
計算の過程は省略するが、前節までと同じ仮定の下で、平均の計算時間も $O(\log n)$ となる。

まとめ

線形探索

- 時間計算量：最悪 $O(n)$，平均 $O(n)$.
- 探索範囲の要素の並び方に制限はない。
- 逐次アクセスのみで可。

二分探索

- 時間計算量：最悪 $O(\log n)$，平均 $O(\log n)$.
- 探索範囲の要素が大きさの順に並んでいる必要あり。
- ランダムアクセスが必要。