プログラミング応用

Week12

本日の内容

- 探索アルゴリズム
  - 二分探索
- 演習(先に演習をします)
- ”計算量”の考え方

復習：線形探索の利点・欠点

- 利点
  - もっとも単純な探索アルゴリズム
    = 実装が簡単
- 欠点
  - 先頭から調べていくので条件を満たすデータが末尾の方にあると探索に時間が掛かる

6/1	6/2	6/3	…	6/30
25℃	26℃	25℃	…	30℃

6/30にたどり着くまで真夏日が見つからない。

二分探索(概要)

- 二分探索
  - データがすでに昇順または降順に並べ替えられているときに使える探索アルゴリズム
    (並べ替えアルゴリズムは来週学習します)

  - 昇順

25℃	26℃	30℃	35℃	36℃

  - 降順

36℃	35℃	30℃	26℃	25℃

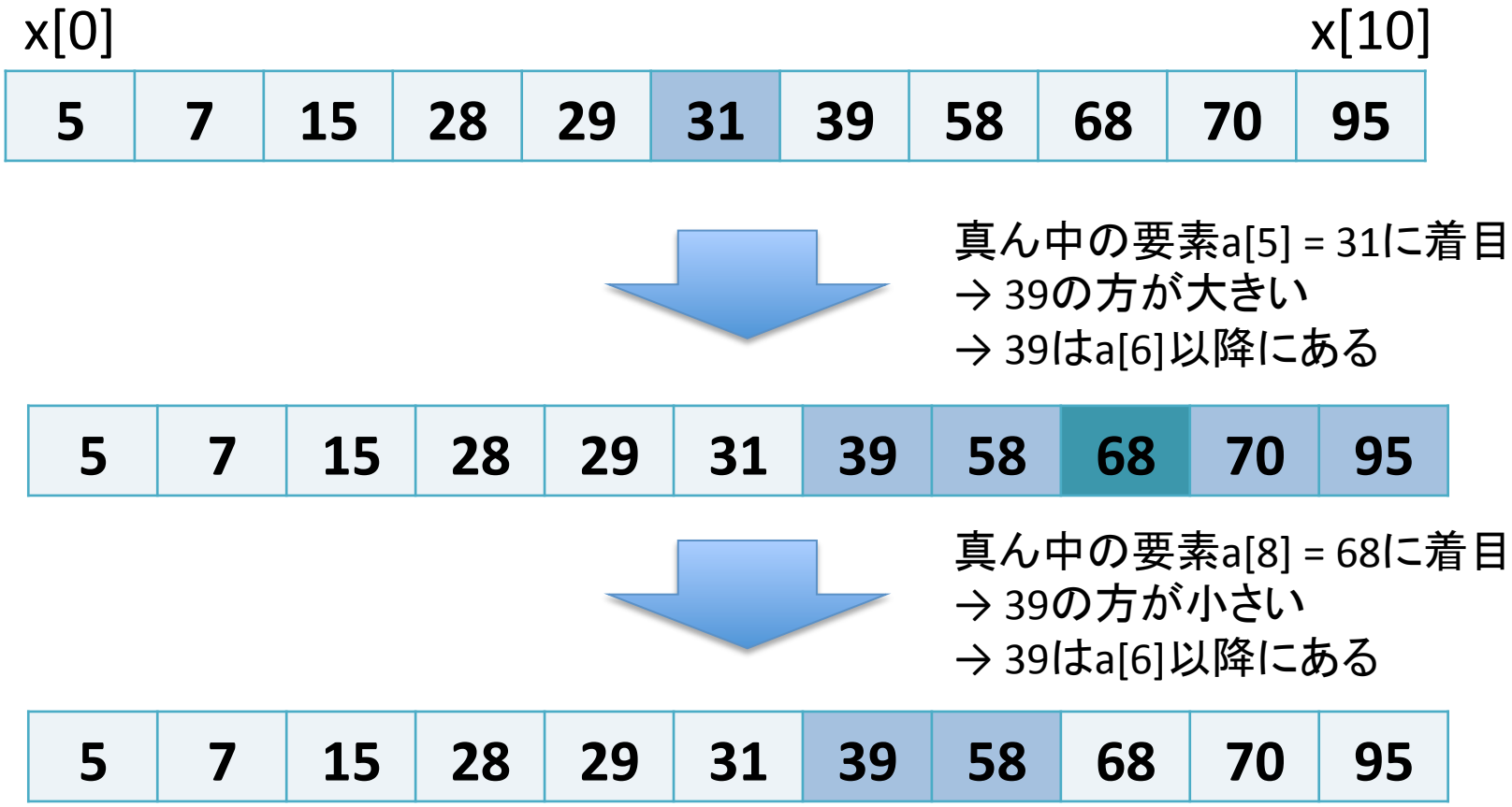
二分探索(イメージ)

- 例) 以下の配列aから31を探索する

a[0]										a[10]
5	**7**	**15**	**28**	**29**	**31**	**39**	**58**	**68**	**70**	**95**

二分探索(イメージ)

- 例) 以下の配列xから39を探索する



二分探索(イメージ)

- 例) 以下の配列aから39を探索する

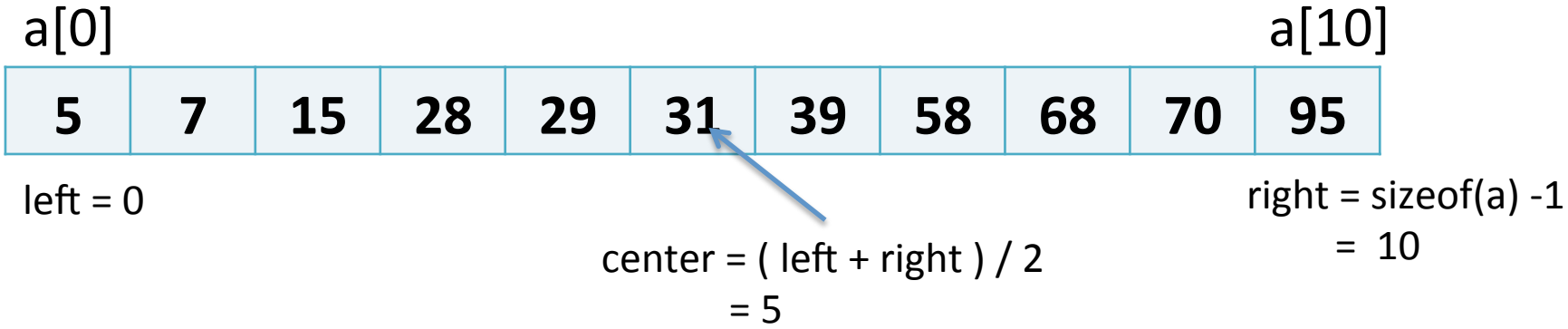
5	7	15	28	29	31	39	58	68	70	95

真ん中の要素はないので便宜的に
前の方の値に着目
a[6] == 39なので探索完了

5	7	15	28	29	31	39	58	68	70	95

二分探索(C言語での表現)

- まずは必要な変数と初期化
  - データを格納する配列a
  - 探索する範囲を指定する変数
    - left: 探索範囲の左端の添字
    - right: 探索範囲の右端の添字
    - center: 探索範囲の中央の添字

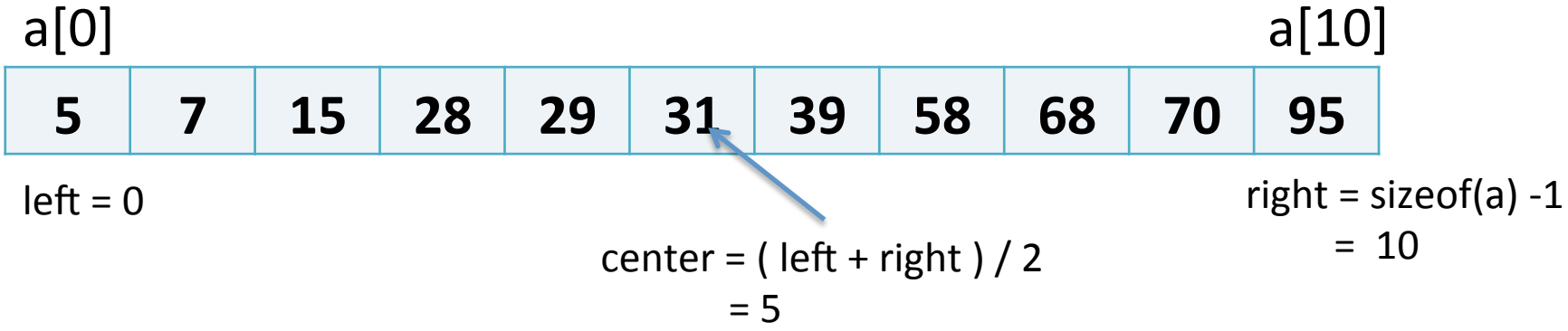


二分探索(C言語での表現)

- 変数の初期化後、以下を繰り返す
  1. 中央の添字を求める
     - center = (left + right) / 2;
  2. 中央の数字を探索対象の数を比較
     - 中央の数が探索対象と等しければ探索終了
     - 中央の数が対策対象よりも大きければ探索範囲の右端の変数を更新
       right = center -1;
     - 中央の数が探索対象よりも小さければ探索範囲の左端の変数を更新
       left = center + 1;

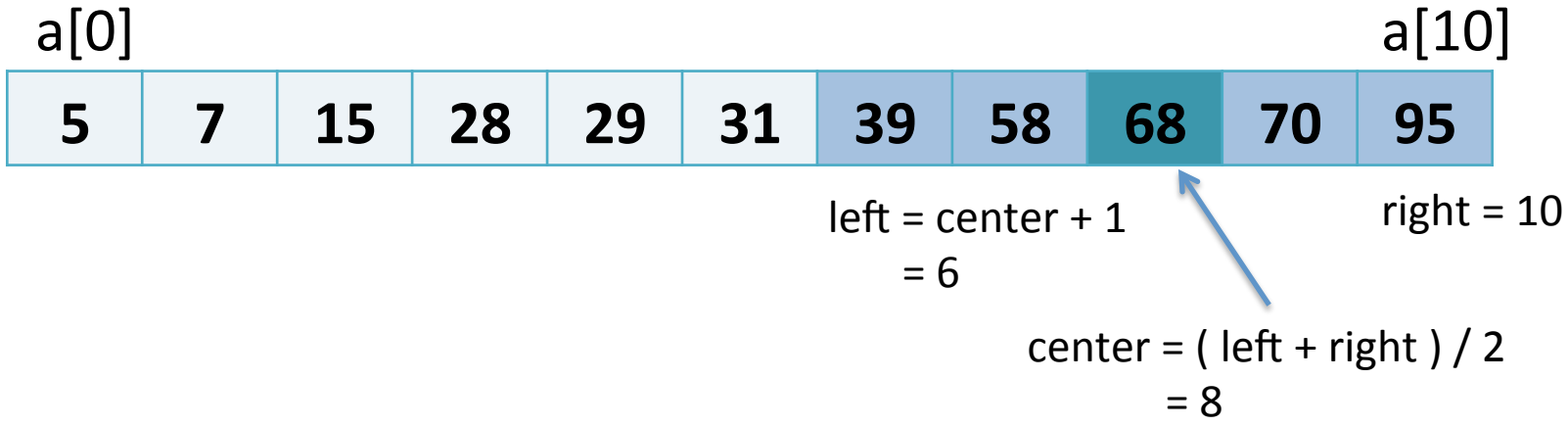
C言語での動作例(1/3)

1. 中央の添字(=5)を求める
    1. 中央の値(=31)はkey(=39)よりも小さい



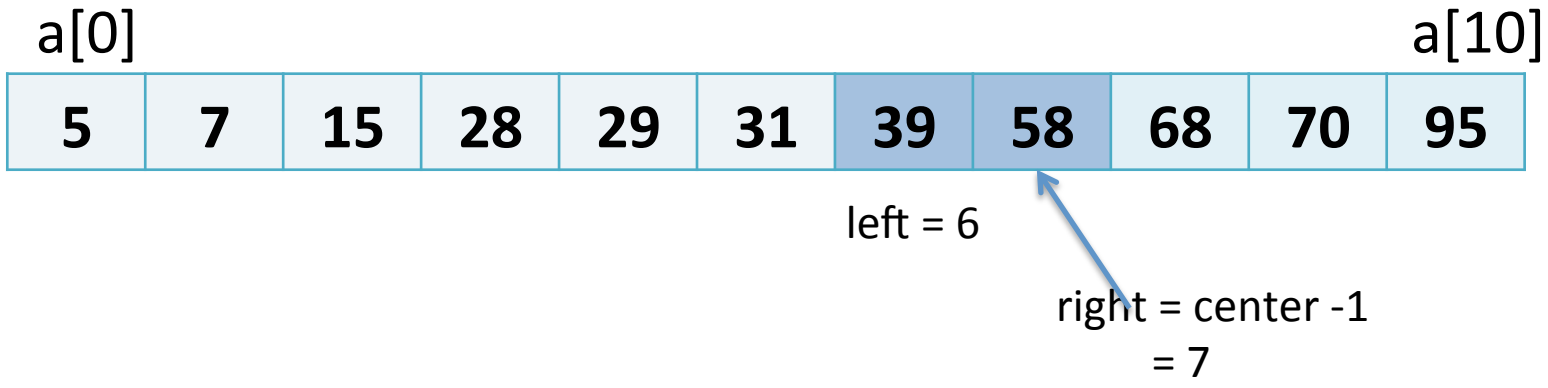
C言語での動作例(2/3)

2. left = center +1する
3. 再度centerを計算
4. 中央の値(=68)はkey(=39)よりも大きい



C言語での動作例(3/3)

5. right = center -1する
6. centerを再度計算する
   → centerは6となりa[6] == keyなので探索終了！



演習0(空のファイル作成)

- 本日の演習では二分探索を実装していきます。
- 演習1から演習3で用いる、
  空のC言語プログラムbinary_search.cを作成

演習1(データ定義)

1. binary_search.cに以下の配列aを定義しなさい

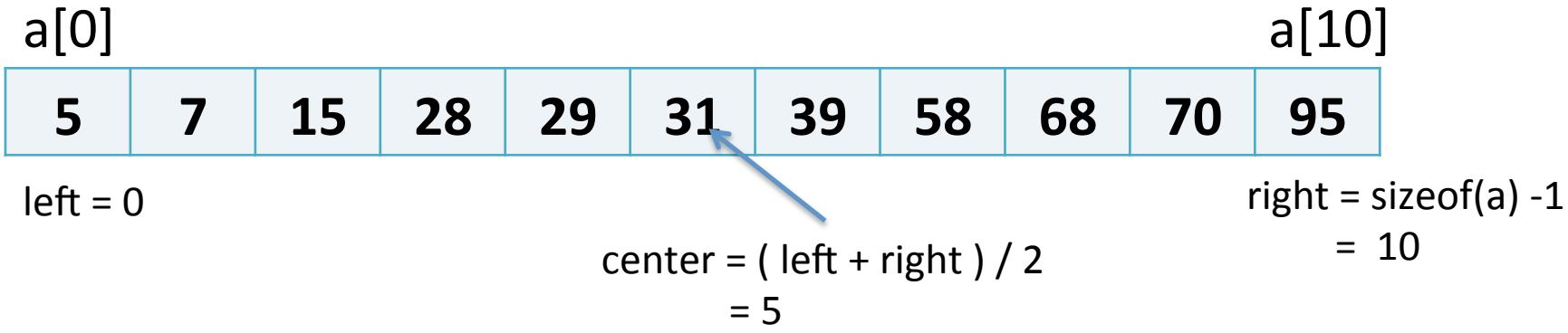
a[0]										a[10]
5	**7**	**15**	**28**	**29**	**31**	**39**	**58**	**68**	**70**	**95**

2. 探索対象の整数31をint型の変数keyに格納しなさい

演習2(必要な変数と初期化)

- 探索範囲を指定するための変数を以下のように定義しなさい

```
int left = 0; /* 探索範囲の先頭の添字 */
int right = sizeof(a) – 1; /* 探索範囲の末尾の添字 */
int center; /* 探索範囲の中央の添字 */
```



演習3(探索処理)

- 以下のdo while文を埋めて探索処理部分を作成

```
// 二分探索アルゴリズム
do {
  /* ここに中央の添字を求める処理を書く*/
  if (a[center] == key) {
    printf("a[%d]が%dでした。", center, key); return 0;
  } else if (/* 「中央の値が探索対象よりも小さい」条件式を書く*/) {
    /* leftの値を中央の添字+1する処理を書く */
  } else {
    /* rightの値を中央の添字-1する処理を書く */
  }
} while (left <= right);
printf("見つかりませんでした。");
return 0;
```

演習4

- 降順に並べ替えた以下の配列の探索をする二分探索プログラムをbinary_acs.cというファイル名で作成しなさい

a[0]										a[10]
95	**70**	**68**	**58**	**39**	**31**	**29**	**28**	**15**	**7**	**5**

演習5(データファイルの利用)

- これまではプログラム中でデータ配列を定義していたが、実際にはファイルからデータを読み込み配列に格納することが多い
- 以下のコマンドでサンプルデータを入手しなさい$以下のコマンドでサンプルデータを自身のマシンにダウンロードしなさい
  $ cp ~ishigaki/sample4binary.txt .

- 次ページのプログラムを用いて、ファイルから配列aを作成しなさい
- 演習3の二分探索プログラムをファイル読み込みした場合でも正しく動作するように書き換えなさい

演習5(ファイルからの読み出し例)

```
int i, data_size = 0;
int a[50000];    /* 大きめのサイズの配列を用意しておく．  */
FILE *fp;

fp = fopen("sample4binary.txt", "r");
if(fp == NULL) {
  printf("ファイルを開くことが出来ませんでした．¥n");
  return 0;
}

/*  ファイルが終わりでない　「かつ」
  配列を飛び出さないうちは，読み込みを続ける  */
while ( ! feof(fp) && data_size < 50000) {
  fscanf(fp, "%d", &(a[data_size]));
  data_size++;
}
fclose(fp);
data_size = data_size-1;
/* なお、上のwhileループでは，EOFの行を余分に
読み込んでいるので，実際のデータ数は一つ少ない．  */
```

演習6

- 先週の演習の線形探索プログラムを、ファイルからデータを読み込むように書き換えなさい。

演習6(時間計測)

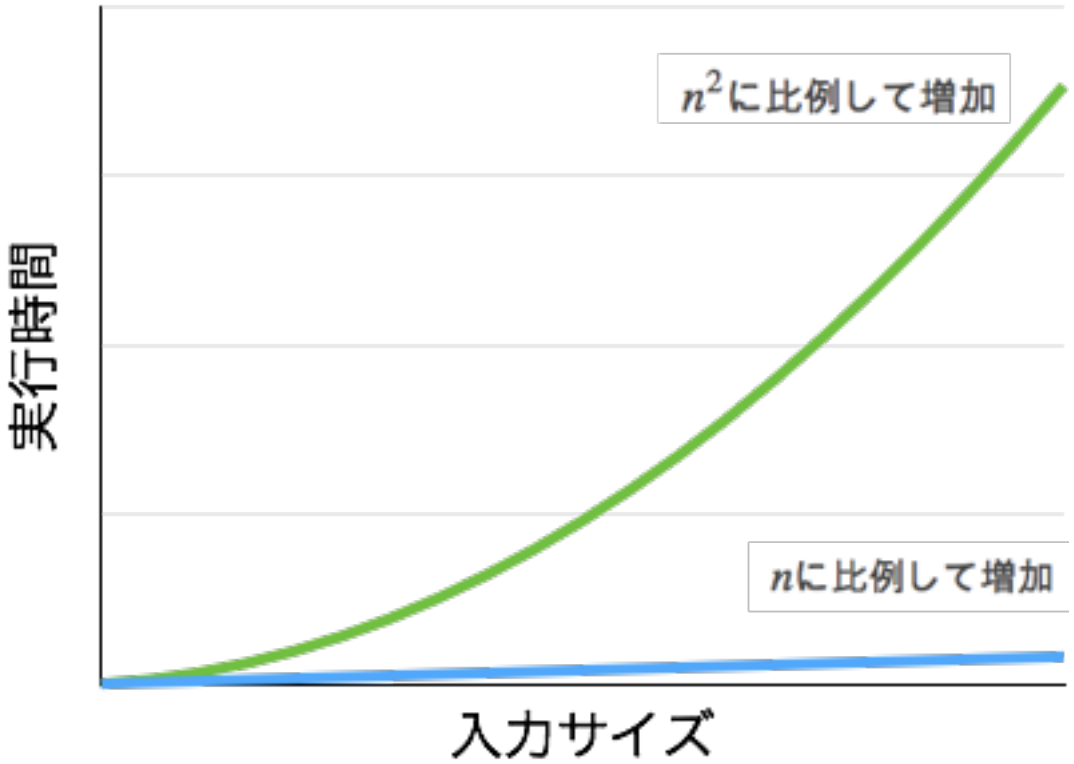
- timeコマンドを使って線形探索、二分探索の実行時間を確認しなさい
  - 1を探索する場合、95を探索する場合の時間を計測する

    $ time ./a.out

  ※ なぜ1では線形探索が速く、95では二分探索の方が速いのか考察してみよう

計算量

- 計算量
  - アルゴリズムがどの程度速いかを表す指標
  - データ数が増加するとどの程度計算時間が増えるかを定量的に表現



計算量の求め方(二分探索)

1. 実行回数を求める

```
 1 #include <stdio.h>
 2
 3 int main(void) {
 4   // 必要な変数定義
 5   int a[] = {5, 7, 15, 28, 29, 31, 39, 58, 68, 70, 95};
 6   int key = 39;
 7   int left = 0;
 8   int right = sizeof(a)/sizeof(int) - 1;
 9   printf("%d", right);
10   int center;
11   // 二分探索アルゴリズム
12   do {
13     int center = (left + right) / 2;
14     if (a[center] == key) {
15       printf("a[%d]が%dでした。", center, key); return 0;
16     } else if (a[center] < key) {
17       left = center + 1;
18     } else {
19       right = center -1;
20     }
21   } while (left <= right);
22   printf("見つかりませんでした。");
23   return 0;
24 }
```

5-8: 変数定義は1回ずつ

13-14: この部分は$\log_2 n$回

15: ここは1回だけ

16-21: この部分は$\log_2 n$回

22-23: この部分は1回

計算量の求め方(二分探索)

2. 一番増加率の高い部分を求める

```
 1 #include <stdio.h>
 2
 3 int main(void) {
 4   // 必要な変数定義
 5   int a[] = {5, 7, 15, 28, 29, 31, 39, 58, 68, 70, 95};
 6   int key = 39;
 7   int left = 0;
 8   int right = sizeof(a)/sizeof(int) - 1;
 9   printf("%d", right);
10   int center;
11   // 二分探索アルゴリズム
12   do {
13     int center = (left + right) / 2;
14     if (a[center] == key) {
15       printf("a[%d]が%dでした。", center, key); return 0;
16     } else if (a[center] < key) {
17       left = center + 1;
18     } else {
19       right = center -1;
20     }
21   } while (left <= right);
22   printf("見つかりませんでした。");
23   return 0;
24 }
```

5-8: 変数定義は1回ずつ

13-14: この部分は$\log_2 n$回

15: ここは1回だけ

16-21: この部分は$\log_2 n$回

22-23: この部分は1回

3. 係数を除去して最終的なオーダーを求める

– $\log_2 n$回実行される行が7行
→ $7 \times \log_2 n$

– 係数を除去したものをオーダーという
$O(\log_2 n)$
→ これが二分探索の計算量