アルゴリズムとデータ構造

第2回　線形探索法と計算量

第2回のキーワード

アルゴリズム関係

- 配列のアルゴリズム
- 計算量
- 時間計算量
- 空間計算量
- 最大(最悪)計算量
- 平均計算量
- O記法(オーダー記法)
- 線形探索(linear search)
- $O(n)$

Java関係(プロIIの復習)

- 配列
- クラスの配列
- ポリモーフィズム(多態性)
- equals, compareTo
- ラッパークラス

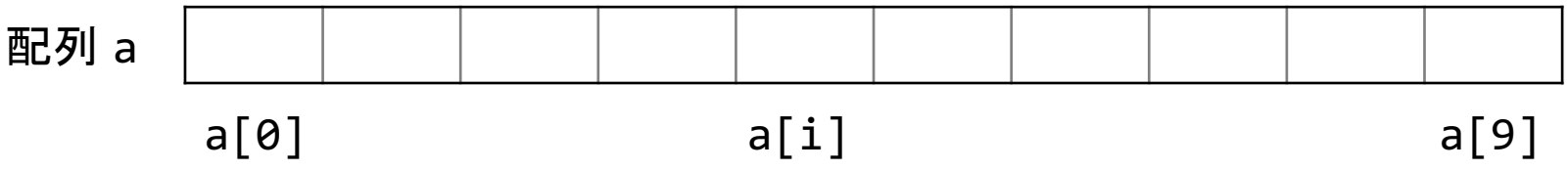
配列要素に関する確率

- 単純な確率は対称性で考える
  - 何と何が「同様に確からしい」(対称的な)関係か考察する

0 1 2 3 4 5 6 7 8 9 の 10通り

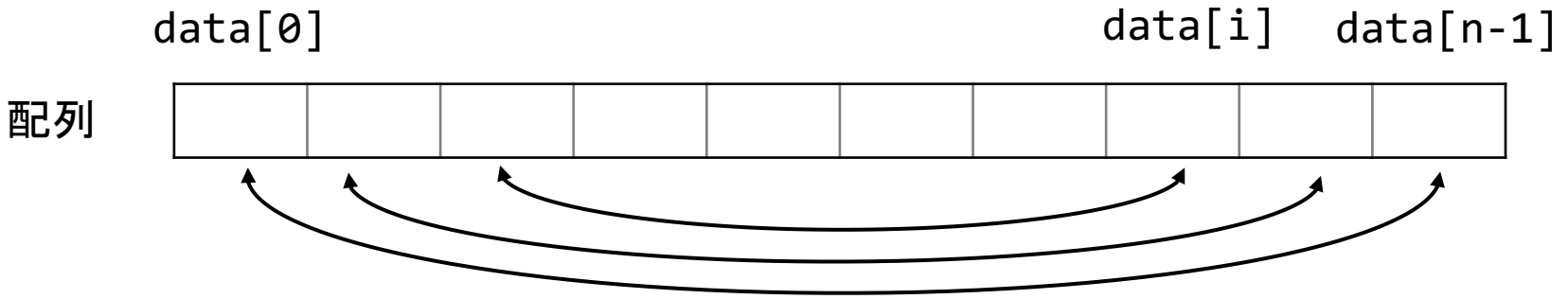
完全にランダムなら
どの要素にどの数字が
入るのも立場が同じ

つまり，どれも確率が等しい
（かつ，確率の合計は1）



どのa[i]でも，0〜9の特定の数字が入る確率は等しい ⇒ 1/10

配列の反転のアルゴリズム



データ数が n 個の場合，交換回数は $\lfloor n/2 \rfloor$ 回

```
n = data.length;
for (int i = 0; i < n / 2; i++) {
    double t = data[i];
    data[i] = data[n - 1 - i];
    data[n - 1 - i] = t;
}
```

ソースコードを分析すると
データ数に依存するのは
ここのループ回数だけ

配列の長さが2倍になると
交換回数も2倍

筆算のアルゴリズム

- 筆算と時間の計算は共通の方法
  - 下の位(桁)から順に計算し，繰り上がり(繰り下がり)があったら，1つ上の位に加算(減算)していく
  - 非常に桁数が多い整数を扱うプログラムにも応用される

	1	7	4
+		5	9
	2	3	3

下の位から上の位へ計算

	18時間	34分	48秒
+)	9時間	53分	35秒
1日	4時間	28分	23秒

下の位から上の位へ計算

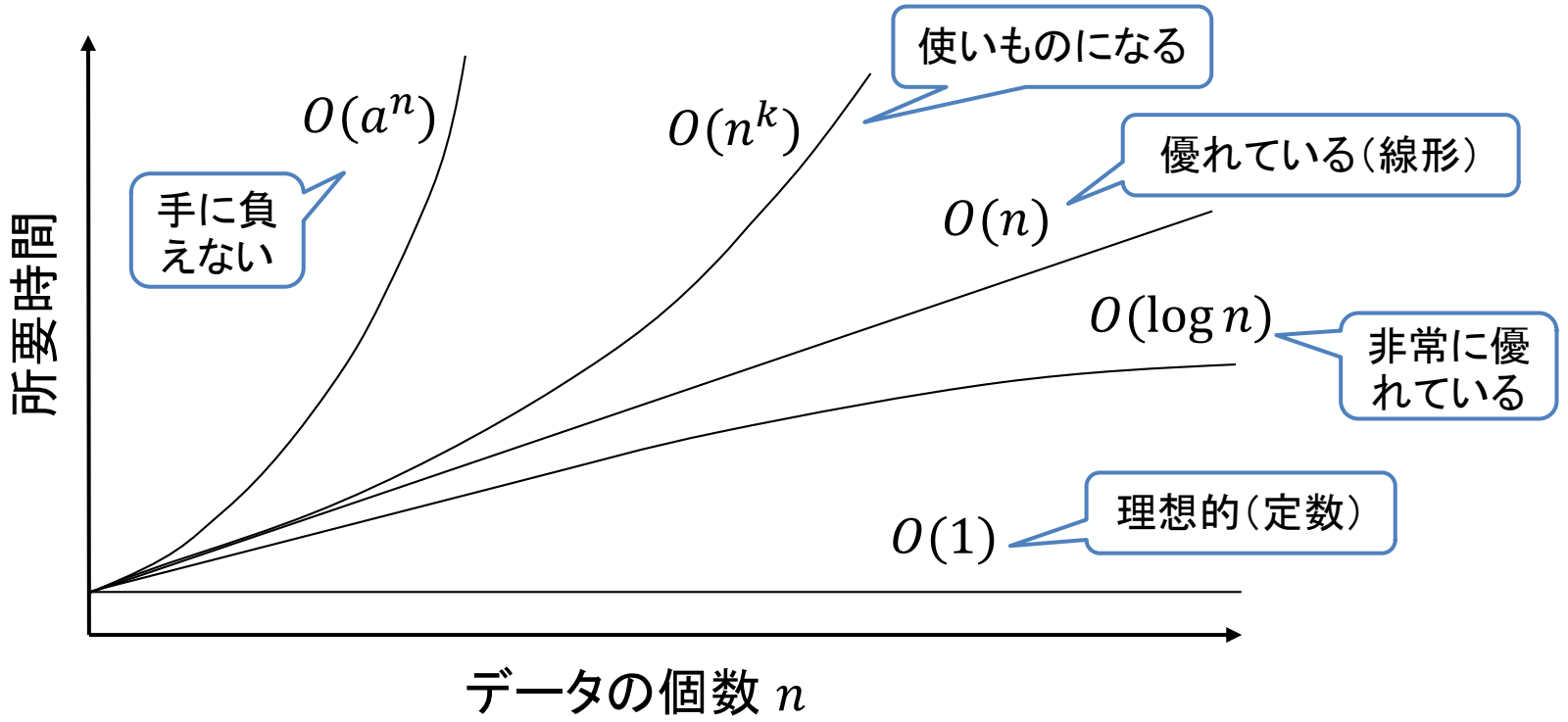
アルゴリズムの性能評価

- よいアルゴリズムとは？
  - 速い（所要時間）　⇒「時間計算量」が少ない
  - 小さい（使用メモリ）　⇒「空間計算量」が少ない
  - これらは，両立が難しいことが多い

- 計算量の指標
  - アルゴリズムの種類によって，データの並び方や構造に対する得意・不得意がある
  - 最大計算量：そのアルゴリズムが最も不得意なデータセットに対する計算量
    - 例：小さい順に並んでいるデータセットから最大値を線形探索
  - 平均計算量：あらゆるデータセットに対する平均の計算量

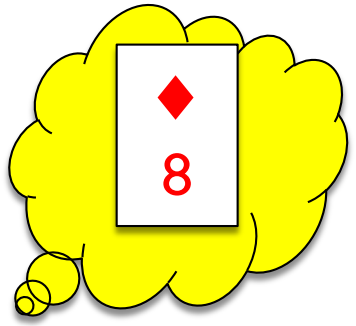
オーダー記法

- 計算量の数学的表現
  - ほとんどアルゴリズムは，データの個数（n）が増えれば増えるほど，計算に時間がかかるようになり，メモリも必要になる
  - nに対する計算量の増加を“関数の種類”（最も影響が大きい関数の定数係数を除いた形式）で分類し，O記法で表す



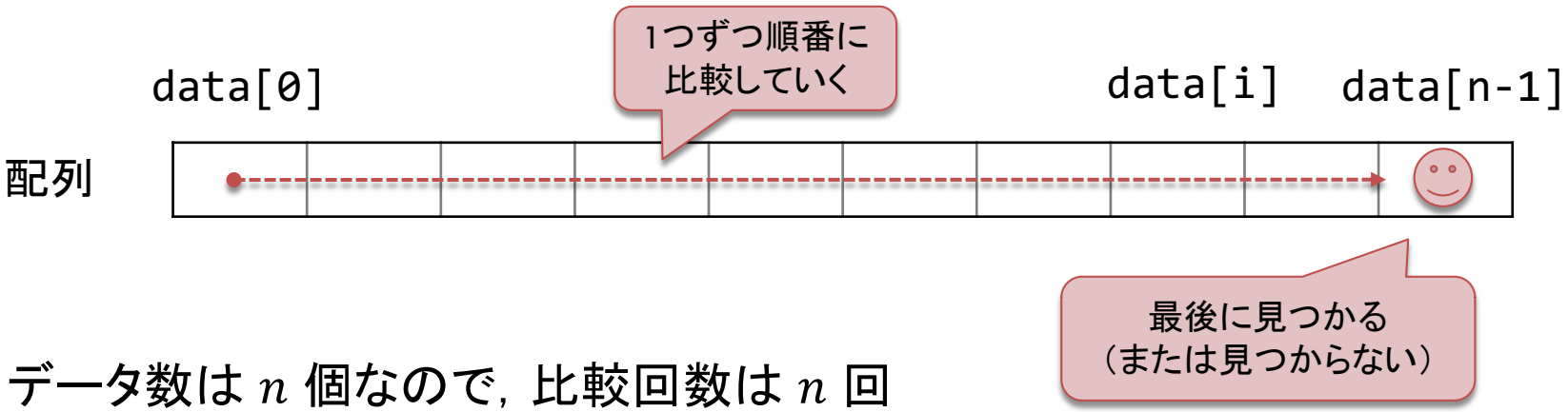
線形探索の考え方

- トランプにたとえると…
  - カードが裏返しに並んでいる（特に順番はない）
  - この中で探したいカードはどこにあるか？



- どうやって探すか？
  - とりあえず，左から（または右から）順にめくってみよう

線形探索の最大計算量



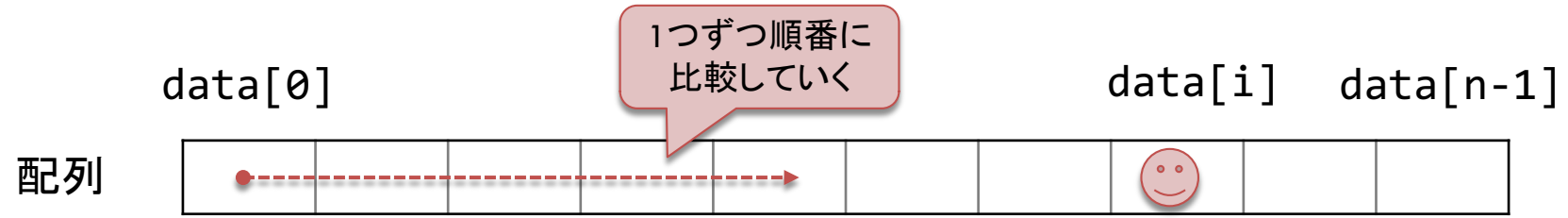
データ数は n 個なので，比較回数は n 回

比較以外にかかる時間は，n に関わらずほぼ一定（定数）

n が十分大きい場合（$n \to \infty$）には比較部分の影響だけ考えればよい

時間計算量は $O(n)$

線形探索の平均計算量



そこにkeyがある確率

$$\frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \frac{1}{n} \quad \cdots \quad \frac{1}{n} \quad \cdots \quad \frac{1}{n}$$

$$\times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \cdots \quad \times \quad \cdots \quad \times$$

発見までの比較回数

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad \cdots \quad i+1 \quad \cdots \quad n$$

$$\| \quad \| \quad \| \quad \| \quad \| \quad \| \quad \cdots \quad \| \quad \cdots \quad \|$$

比較回数の期待値

$$\frac{1}{n} + \frac{2}{n} + \frac{3}{n} + \frac{4}{n} + \frac{5}{n} + \frac{6}{n} + \cdots + \frac{i+1}{n} + \cdots + \frac{n}{n}$$

平均計算量

$$= \frac{1}{n}(1 + 2 + \cdots + n) = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2} = \frac{1}{2}n + \frac{1}{2}$$

$n \to \infty$ を考え $O(n)$

数列の和の復習

- 等差数列の和
  - 考え方：順番をひっくり返したものを足す

$$\begin{aligned} S &= 1 + 2 + \cdots + (n-1) + n \\ +)\ S &= n + (n-1) + \cdots + 2 + 1 \\ \hline 2S &= (n+1) \times n \end{aligned}$$

$$\therefore S = \frac{n(n+1)}{2}$$

- 等比数列の和
  - 考え方：公比をかけたものと差をとる

$$\begin{aligned} S &= 2 + 4 + 8 + \cdots + 2^{n-1} + 2^n \\ -)\ 2S &= \quad 4 + 8 + 16 + \cdots + 2^n + 2^{n+1} \\ \hline (1-2)S &= 2 - 2^{n+1} \end{aligned}$$

$$\therefore S = \frac{2 - 2^{n+1}}{1 - 2}$$

クラス型の比較

- Javaにおけるクラス型の比較
  - 値の比較に，比較演算子（==，<，>等）は使えない
  - 代わりに，以下のメソッドを使う

- x.equals(y)
  - 意味的に x = y ならば true，そうでなければ false を返す
  - Javaの全てのクラスがObjectクラスから継承するメソッド

- x.compareTo(y)
  - x < y ならば負，x = y ならば 0，x > y ならば正を返す
  - StringやIntegerなど，Comparableインタフェースを実装するクラスが持つメソッド

ラッパークラス

- Javaの基本型（intやdouble）はクラスではない
  - 昔，Javaの開発時にパフォーマンスを重視したためだが…
  - Objectクラス継承していないなど，不便な場合がある
- 各基本型に対応する「ラッパークラス」が使える
  - int → Integer，double → Double，char → Character
  - 対応する基本型をフィールドに持つ（包む）特別なクラス
  - 例）Double x = new Double(1.4142); // doubleの値を包む
- 基本型と自動的に相互変換（autoboxing, unboxing）
  - 文法上，ほぼ基本型と同じように使え，四則演算等も可能
  - 例）Integer i = 10; i += 20; System.out.println(i);