

9.3 例 : マージソート (Merge sort)

この章で前に示した挿入ソートは、書くのは簡単ですがあまり効率的ではありません。その平均的な計算量は入力リストの長さの2乗に比例します。さて、挿入ソートより効率的にリスト要素をソートするプログラムをデザインしましょう。そのためのよいアルゴリズムは **マージソート** です。次のように動きます。

はじめに、もしリストが0個あるいは1個の要素を持つなら、それはすでにソートされているので、そのままリストを返します。長いリストは2つのサブリストに分割され、それぞれが元のリストの半分の要素を含むようにします。各サブリストはソート関数への再帰呼び出しでソートされ、得られる2つのソート済みリストは、マージ操作にて結合されます。

マージソートを汎用的な実装とするために、ソートされるリストの要素型だけでなく、要素の比較に使う関数も指定できるようにすべきです。それら2つの要素をパラメータとして渡すことで、可能な限り汎用性のある関数が得られます。以上から次の実装を得ます。

```
def msort[A](less: (A, A) => Boolean)(xs: List[A]): List[A] = {
  def merge(xs1: List[A], xs2: List[A]): List[A] =
    if (xs1.isEmpty) xs2
    else if (xs2.isEmpty) xs1
    else if (less(xs1.head, xs2.head)) xs1.head :: merge(xs1.tail, xs2)
    else xs2.head :: merge(xs1, xs2.tail)
  val n = xs.length/2
  if (n == 0) xs
  else merge(msort(less)(xs take n), msort(less)(xs drop n))
}
```

msort の計算量は $O(N \log(N))$ 、ただし N は入力リストの長さです。理由を見てみます。リストを二つに分割し、ソートされた2つのリストをマージするには、それぞれに引数のリストの長さに比例した時間を要します。msort の再帰呼び出しの度に入力要素数は半分になり、リスト長が1に達するまでに $O(\log(N))$ 回の再帰呼び出しが行われます。しかし長い方のリストについては、各呼び出しで更に2回の呼び出しが生じます。すべてを足し合わせると、 $O(\log(N))$ 回の呼び出しレベルのそれぞれに対して、元のリストの全要素が1回の分割操作と1回のマージ操作に関わります。すなわち各呼び出しレベルは、全部で $O(N)$ に比例するコストがかかります。 $O(\log(N))$ の呼び出しレベルがあるため、全体で $O(N \log(N))$ のコストとなります。このコストはリストの要素の初期分布には依存せず、最悪ケースのコストは平均ケースと同じです。このためマージソートは、リストのソートとして魅力的なアルゴリズムです。

次は msort を使う例です。

```
msort((x: Int, y: Int) => x < y)(List(5, 7, 1, 3))
```

msort の定義はカーリー化されていて、特定の比較関数を使って簡単に特化できます。たとえば

```
val intSort = msort((x: Int, y: Int) => x < y)
val reverseSort = msort((x: Int, y: Int) => x > y)
```

[前ページ](#) [9章](#) [目次](#) [次ページ](#)

名前:	<input type="text"/>
コメント:	<input type="text"/>

投稿