アルゴリズム B 第 7 回　　2019 年 11 月 25 日

7　整列アルゴリズム (2)

前回と同様，配列 $a[1], a[2], \ldots, a[n]$ を昇順に並び替えるものとして説明する．

マージソート

マージソート (merge sort) は，配列を 2 つに等分し，それぞれを再帰的に整列する．整列された 2 つの部分列を結合して整列された 1 つの列を作る．

整列された 2 つの列から整列された 1 つの列を作る操作を**マージ** (merge) という．長さ m の配列 a と長さ n の配列 b をマージして配列 c を作るアルゴリズムは次のように書ける[*1]．

```
1: MERGE(a[ ], m, b[ ], n, c[ ])
2:     i = 1, j = 1
3:     for k = 1 to m + n
4:         if j = n + 1 or (i ≤ m and a[i] ≤ b[j]) then
5:             c[k] = a[i], i = i + 1
6:         else
7:             c[k] = b[j], j = j + 1
```

このアルゴリズムによるマージの最悪計算量は $O(m+n)$ である．

マージソート全体のアルゴリズムは次のように書ける．

```
1: MERGESORT(a[ ], n)
2:     if n > 1 then
3:         h = ⌊n/2⌋
4:         長さ h の配列 l を作り，a[1], a[2], ..., a[h] を代入する
5:         長さ n − h の配列 r を作り，a[h + 1], a[h + 2], ..., a[n] を代入する
6:         MERGESORT(l[ ], h)
7:         MERGESORT(r[ ], n − h)
8:         MERGE(l[ ], h, r[ ], n − h, a[ ])
```

長さ n の配列に対するマージソートの最悪計算量を $T(n)$ とする．簡単のため，$n = 2^k$ とすると，次の漸化式が成り立つ．

$$T(1) = c_1, \quad T(n) = 2T\left(\frac{n}{2}\right) + c_2 n \quad (n \geq 2).$$

ただし，c_1, c_2 は n に依存しない正の定数である．この漸化式から $T(n) = O(n \log n)$ が得られる．$n \neq 2^k$ の場合も同じ漸近評価が成り立ち，マージソートの最悪計算量は $O(n \log n)$ である．なお，マージソートでは配列 l, r を作るために追加の作業領域を必要とする．

マージソートや次に説明するクイックソートのように，問題をより小さいサイズの部分問題に分割し，その解を組み合わせて元の問題を解く方法を**分割統治法** (divide and conquer) という．

[*1] 配列 a, b のどの要素よりも大きい値を持つ要素 ∞ を a, b の末尾に追加することで，第 4 行の条件を $a[i] \leq b[j]$ のみにすることもできる．この ∞ のような要素を**番兵** (sentinel) と呼ぶ．

クイックソート

クイックソート (quicksort) は，配列から**ピボット** (pivot) という要素 x を選び，配列を x 以上の要素からなる部分列，x 自身，x 以下の要素からなる部分列に分割する[*2]．2 つの部分列に対して再帰的に整列を行う．

部分列を作る素朴な方法として，新たな配列 b, c を作り，x 未満の要素を b に，x 以上の要素を c にそれぞれ追加する方法がある．この方法は作業領域を必要とする．

次の関数は，作業領域を使わずに，部分列 $a[l], \ldots, a[r]$ から 2 つの部分列を作る．この関数はピボットとして左端の $a[l]$ を選び，列を分割した位置の添字を返す．また，**repeat**（文）**until**（条件）は，まず（文）を実行し，（条件）が偽であれば **repeat** の直後に戻って繰り返す．（条件）が真であればループを終了する[*3]．

```
1:  PARTITION(a[ ], l, r)
2:      i = l, j = r + 1, x = a[l]
3:      while i ≤ j
4:          repeat
5:              i = i + 1
6:          until i > r or a[i] ≥ x
7:          repeat
8:              j = j − 1
9:          until a[j] ≤ x
10:         if i < j then swap(a[i], a[j])
11:     a[l] = a[j], a[j] = x
12:     return(j)
```

クイックソートは次のように書ける．ただし，配列 a 全体を整列するときは QUICKSORT$(a[\,], 1, n)$ とする．

```
1:  QUICKSORT(a[ ], l, r)
2:      if l < r then
3:          j = PARTITION(a[ ], l, r)
4:          QUICKSORT(a[ ], l, j − 1)
5:          QUICKSORT(a[ ], j + 1, r)
```

計算量の議論を簡単にするため，配列の要素はすべて異なるとして，PARTITION ではなく素朴な方法で部分列を作ると仮定する．計算量は配列要素の比較回数で評価できるから，比較回数を評価すればよい．

ピボットが部分列の最大値や最小値であった場合，得られる部分列の一方は長さ 0 になる．これが繰り返された場合，比較回数は $O(n^2)$ となる．したがって，クイックソートの最悪計算量は $O(n^2)$ である．

一方，入力としてすべての順列が一様に現れると仮定する．平均比較回数を $T(n)$ とすると，

$$T(0) = T(1) = 0, \quad T(n) = \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) + n - 1 \quad (n \geq 2)$$

が成り立つ．この漸化式から $T(n) = O(n \log n)$ を得る．したがって，平均計算量は $O(n \log n)$ である．

[*2] x 以外の x と値が等しい要素はどちらの部分列に入れてもよく，実装に依存して決まる．PARTITION では両方に入る．

[*3] PARTITION を呼び出したとき $a[r+1]$ が存在して $a[l] \leq a[r+1]$ ならば，第 6 行の条件を $a[i] \geq x$ のみにすることができる．QUICKSORT から呼ぶときは番兵として $a[n+1] = \infty$ を追加すればよい．