アルゴリズムとデータ構造

第16週目

担当　情報システム部門　徳光政弘
2025年10月7日

今日の内容

- 分割統治法の考え方
- クイックソート
- 和の計算
- マージソート（今日の主題はこれ）

分割統治法

- 問題をさらに小さい問題に分割して、解を統合して全体の解を求める方法

分割統治法

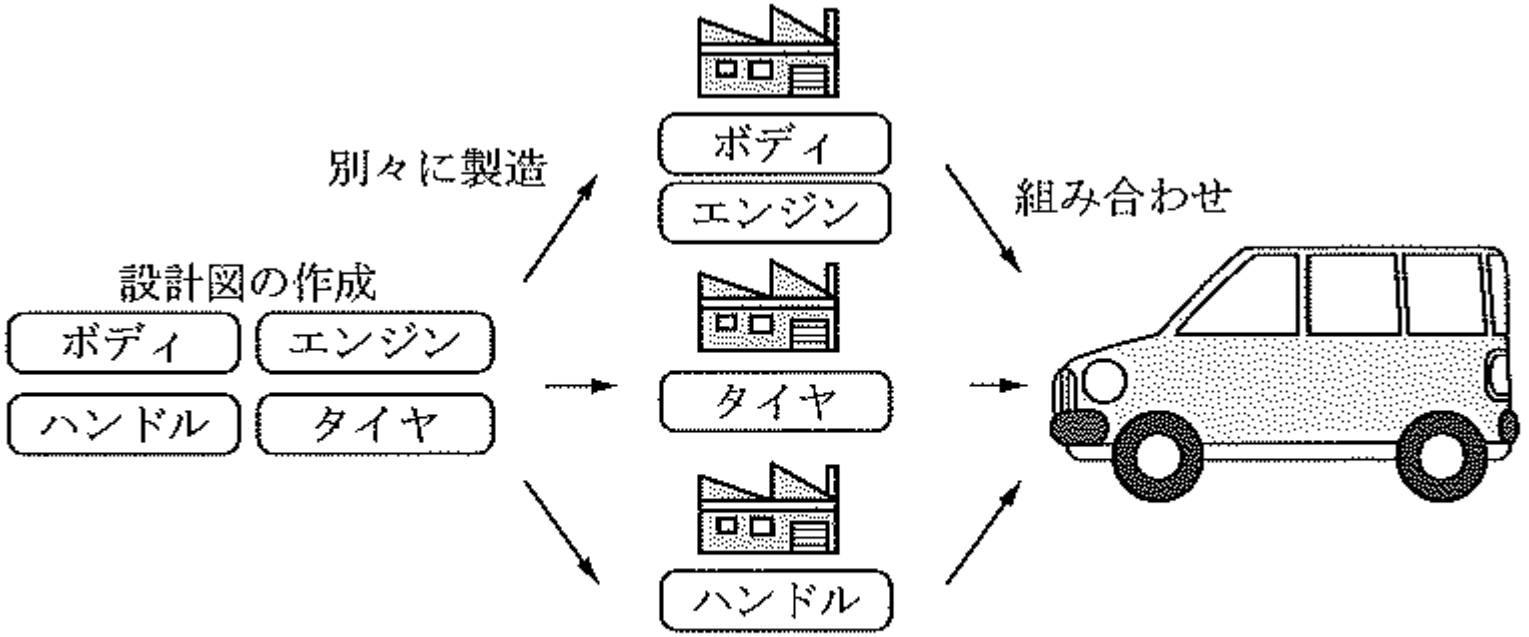


図 7.1　分割統治法を用いた自動車の製造

分割統治法の手順

ステップ１：分割　問題をいくつかの部分問題に分割する．

ステップ２：統治　分割された部分問題を解く．

ステップ３：組合せ　ステップ２で得られた部分問題の解をもとに，いくつかの計算を行い，問題全体の解を得る．

和の計算アルゴリズム　再訪

アルゴリズム **3.4**　和の計算を行う再帰的なアルゴリズム(その **2**)

```
recursive_sum2(A[0], A[1], ..., A[n-1])  {
  if (入力の引数がA[k]という1つの配列要素のみである) { return A[k]; }
  else  {
    配列Aを半分ずつの以下の2つの配列に分割する;          ---(1)
      A1={A[0], A[1], ..., A[(n-1)/2]},
      A2={A[(n-1)/2+1], A[(n-1)/2+2], ..., A[n-1]}
    x=recursive_sum2(A1);                                ---(2)
    y=recursive_sum2(A2);                                ---(2)
    return x+y;                                          ---(3)
  }
}
```

クイックソート　再訪

アルゴリズム 6.1　クイックソート

```
quicksort(D,left,right) {
  if (left<right) {
    pivot_index=partition(D,left,right);  ---(1)
    quicksort(D,left,pivot_index-1);      ---(2)
    quicksort(D,pivot_index+1,right);     ---(2)
  }
}
```

ソート対象が分割されて、それぞれのデータ集合で再度分割とソートが実行される

掛け算と分割統治法

- C = A × B
- 通常扱う掛け算は32ビットや64ビット
- 64ビットの場合は19桁の整数を扱える
- 暗号や科学計算では100桁以上の計算が必要で、単純な整数型では計算ができない（つまり、工夫が必要）

教科書の例　説明文

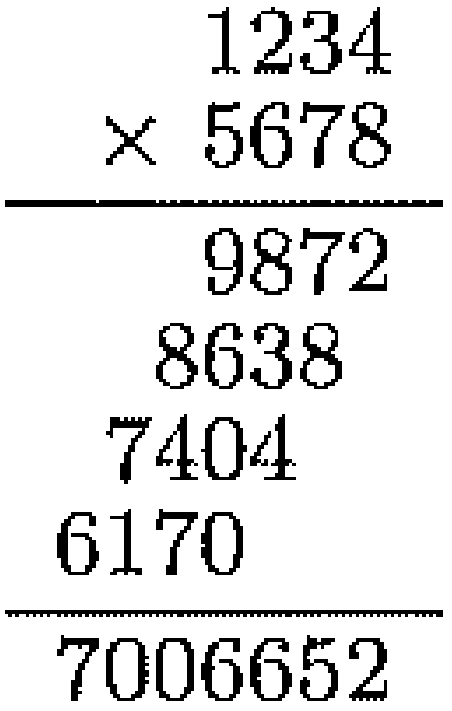
−9223372036854775808 ～ 9223372036854775807 の間の 19 桁以下の整数しか扱う

掛け算と分割統治法

X = 1234、Y = 5678として、X×Yを考える

X[0]=4,　X[1]=3,　X[2]=2,　X[3]=1,

Y[0]=8,　Y[1]=7,　Y[2]=6,　Y[3]=5



掛け算と分割統治法

アルゴリズム 7.1　基本的な整数の掛け算

入力：サイズ n の２つの配列 X[0], X[1], ..., X[n-1], Y[0], Y[1], ..., Y[n-1]

```
sum=0; power=1;
for (i=0; i<n; i++)  {          //xの掛け算を行う桁を変数iで指定する
  s=0; p=power;
  for (j=0; j<n; j++)  {        //このfor文でxのi桁目とyの掛け算を計算
    s=s+p*X[i]*Y[j]; p=p*10;
  }
  sum=sum+s; power=power*10;
}
sumを出力;
```

「power」の役割は各自で考えてみる

掛け算の分割統治法による計算

分割統治法の考え方で任意の桁数が計算できるアルゴリズムを考える。
整数x、yをn/2桁ずつに分解して、計算する。

$$x = x_1 \times 10^{\frac{n}{2}} + x_2, \quad y = y_1 \times 10^{\frac{n}{2}} + y_2$$

たとえば，$x = 1234, y = 5678$ の場合，$x_1 = 12, x_2 = 34, y_1 = 56, y_2 = 78$ である．つまり，x_1, y_1 はそれぞれ x, y の上位 $\frac{n}{2}$ 桁を表し，x_2, y_2 はそれぞれ x, y の下位 $\frac{n}{2}$ 桁を表している．

掛け算の分割統治法による計算

一般化する。単純に上位・下位の桁ごとに表現しているだけ。

このとき，$x \times y$ を x_1, x_2, y_1, y_2 を用いて表すと以下のように変形できる．

$$
\begin{aligned}
x \times y &= (x_1 \times 10^{\frac{n}{2}} + x_2) \times (y_1 \times 10^{\frac{n}{2}} + y_2) \\
&= x_1 y_1 \times 10^n + (x_1 y_2 + x_2 y_1) \times 10^{\frac{n}{2}} + x_2 y_2
\end{aligned}
$$

掛け算の分割統治法による計算

さらに整理を進める。再び同じ表現ができる構造が出てくる。

$$
\begin{aligned}
x \times y &= x_1 y_1 \times 10^n + (x_1 y_2 + x_2 y_1) \times 10^{\frac{n}{2}} + x_2 y_2 \\
&= \underbrace{x_1 y_1}_{a} \times 10^n + (\underbrace{(x_1 + x_2)(y_1 + y_2)}_{b} - (\underbrace{x_1 y_1}_{a} + \underbrace{x_2 y_2}_{c})) \times 10^{\frac{n}{2}} + \underbrace{x_2 y_2}_{c}
\end{aligned}
$$

掛け算の分割統治法による計算

教科書の具体例

この事実について例を用いて再帰的に考えてみよう．$x = 1234$, $y = 5678$ の場合，1234×5678 という 4 桁の整数の掛け算は，$a = 12 \times 56$, $b = (12 + 34)(56 + 78)$, $c = 34 \times 78$ という 3 つの 2 桁の掛け算を用いて以下の式で表すことができる．

$$1234 \times 5678$$
$$= \underbrace{12 \times 56}_{a} \times 10^4 + (\underbrace{(12 + 34)(56 + 78)}_{b} - (\underbrace{12 \times 56}_{a} + \underbrace{34 \times 78}_{c})) \times 10^2 + \underbrace{34 \times 78}_{c}$$

12×56に再び同じ構造になるように考え方を適用する。

$$12 \times 56 = \underbrace{1 \times 5}_{a} \times 10^2 + (\underbrace{(1 + 2)(5 + 6)}_{b} - (\underbrace{1 \times 5}_{a} + \underbrace{2 \times 6}_{c})) \times 10^1 + \underbrace{2 \times 6}_{c}$$

掛け算の分割統治法による計算

アルゴリズム **7.2**　分割統治法を用いた大きな整数の掛け算

```
入力：サイズnの２つの配列 X[0], X[1], ..., X[n-1], Y[0], Y[1], ..., Y[n-1]
product(X[0],X[1],...,X[n-1],Y[0],Y[1],...,Y[n-1]) {
  if (入力配列X,Yのサイズが1である) { return XとYの値の積; }   ---(1)
  else {
    配列XとYをそれぞれ半分ずつの以下の2つの配列に分割する.      ---(1)
    X1={X[0],X[1],...,X[n/2-1]},X2={X[n/2],X[n/2+2],...,X[n-1]}
    Y1={Y[0],Y[1],...,Y[n/2-1]},Y2={Y[n/2],Y[n/2+2],...,Y[n-1]}
    a=product(X1,Y1);b=product(X1+X2,Y1+Y2);c=product(X2,Y2);  ---(2)
    //X1+X2, Y1+Y2は配列の対応する各要素を足した配列を表す
    return a*10^n+(b-(a+c))*10^(n/2)+c; //10^nは10のn乗を表す  ---(3)
  }
}
```

掛け算の分割統治法による計算

計算量の考え方は木の考え方で求める。
T(n/2)が3個あるため、3倍する。和の部分は定数とする。
全ページのアルゴリズムとの対応を意識する。

$$T(n) = \begin{cases} \underbrace{3T\left(\frac{n}{2}\right)}_{(2)\ \text{再帰呼び出し}} + \underbrace{cn}_{(1),\ (3)} & (n \geq 2 \text{ の場合}) \\ c & (n = 1 \text{ の場合}) \end{cases}$$

掛け算の分割統治法による計算

3個の項があるため2分木ではないことに注意する。
木の高さは $h = 1 + \log_2 n$ となる。

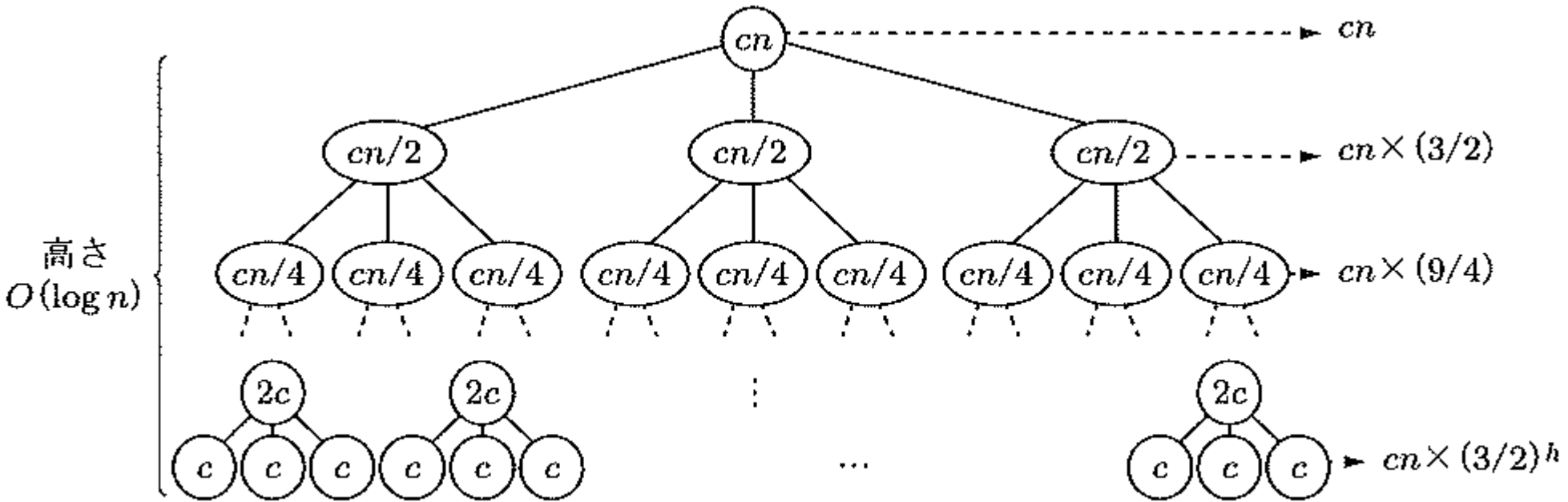


図 7.2　アルゴリズム 7.2 の再帰木

掛け算の分割統治法による計算

計算量の総和は、木の各レベルにおける計算量と、各レベルの計算量の総和となる。

初項 a, 公比 r の等比数列の和の公式 $\sum_{i=0}^{n-1} a \cdot r^i = a \cdot \frac{1-r^n}{1-r}$

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_2 n} cn \cdot \left(\frac{3}{2}\right)^i \\
&= cn \cdot \frac{1-(\frac{3}{2})^{1+\log_2 n}}{1-\frac{3}{2}} = 2cn\left(\left(\frac{3}{2}\right)^{1+\log_2 n} - 1\right) \\
&= 2cn\left(\frac{3}{2} \cdot \left(\frac{3^{\log_2 n}}{2^{\log_2 n}}\right) - 1\right) = 2cn\left(\frac{3}{2n} \cdot 3^{\log_2 n} - 1\right) \\
&= O(3^{\log_2 n}) = O(n^{\log_2 3}) \quad (\because \log_2 3^{\log_2 n} = \log_2 n \log_2 3 = \log_2 n^{\log_2 3})
\end{aligned}
$$

掛け算の分割統治法による計算

計算量の比較

単純な方法(教科書のアルゴリズム)

$$O(n^2)$$

掛け算を分解する方法(教科書のアルゴリズム)

$$O(n^{\log_2 3}) = O(n^{1.59})$$

マージソート

- 分割統治の考え方を使うソート
- 計算量は速い、安定性(入力の順番)がある。

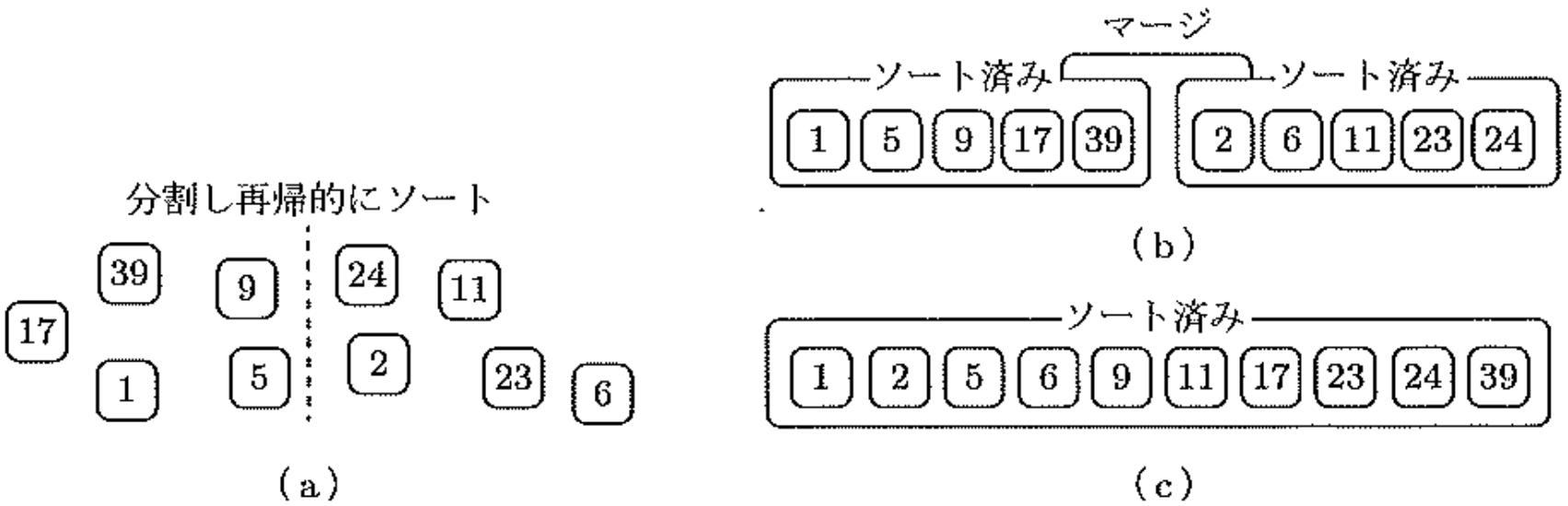


図 7.3　マージソートのアイデア

マージソートの手順

$D = \{d_0, d_1, \ldots, d_{n-1}\}$ とする).

① 集合 D に含まれる要素が 1 つならば，そのまま何もせずにアルゴリズムを終了する.

② 集合 D に含まれるすべてのデータを，

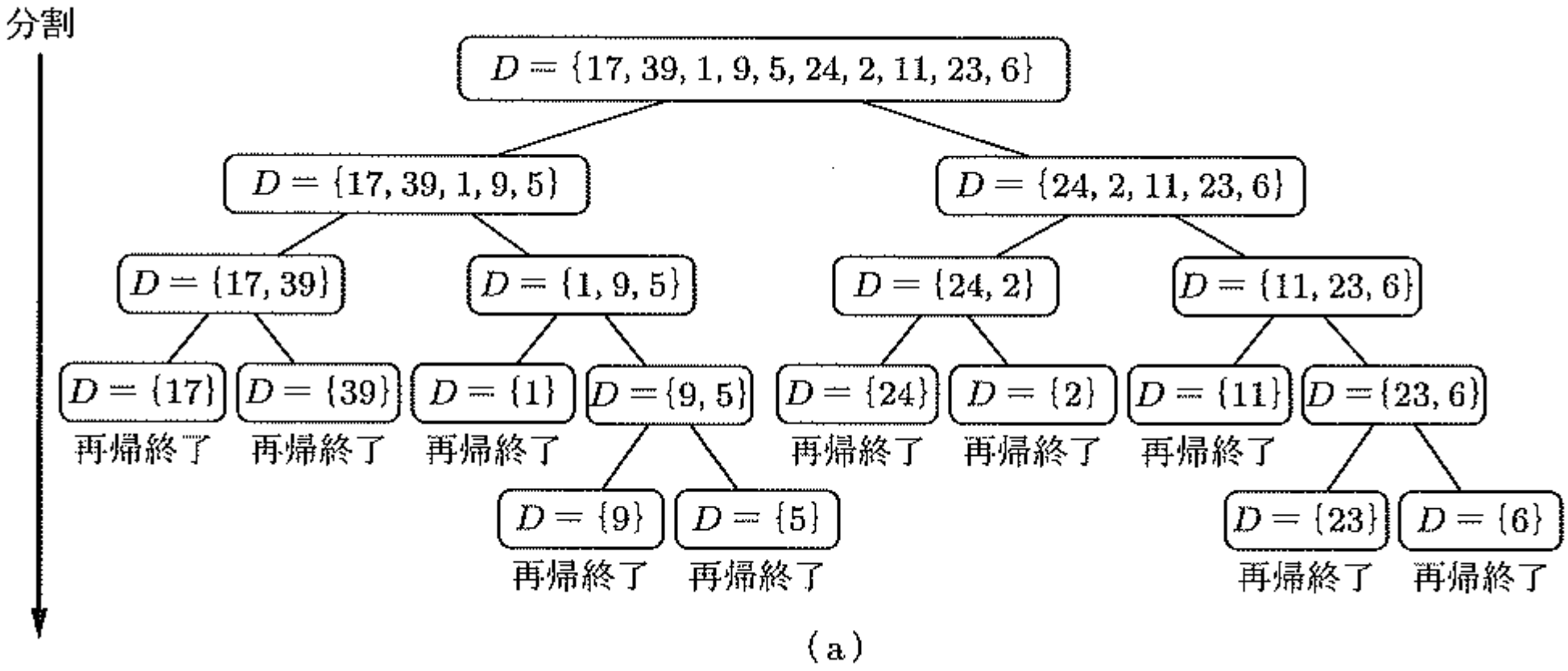
$$D_1 = \{d_0, d_1, \ldots, d_{\frac{n}{2}-1}\}, \quad D_2 = \{d_{\frac{n}{2}}, d_{\frac{n}{2}+1}, \ldots, d_{n-1}\}$$

という 2 つの集合に分割する.

③ 集合 D_1 と集合 D_2 をそれぞれ再帰的にソートする(再帰的なソート終了時には，集合 D_1 と集合 D_2 はソート済みの列である).

④ マージ操作により，D_1 と D_2 から 1 つのソート列を求める.

マージソート


分割
$D = \{17, 39, 1, 9, 5, 24, 2, 11, 23, 6\}$
$D = \{17, 39, 1, 9, 5\}$
$D = \{24, 2, 11, 23, 6\}$
$D = \{17, 39\}$
$D = \{1, 9, 5\}$
$D = \{24, 2\}$
$D = \{11, 23, 6\}$
$D = \{17\}$
$D = \{39\}$
$D = \{1\}$
$D = \{9, 5\}$
$D = \{24\}$
$D = \{2\}$
$D = \{11\}$
$D = \{23, 6\}$
$D = \{9\}$
$D = \{5\}$
$D = \{23\}$
$D = \{6\}$
再帰終了

(a)

マージソート

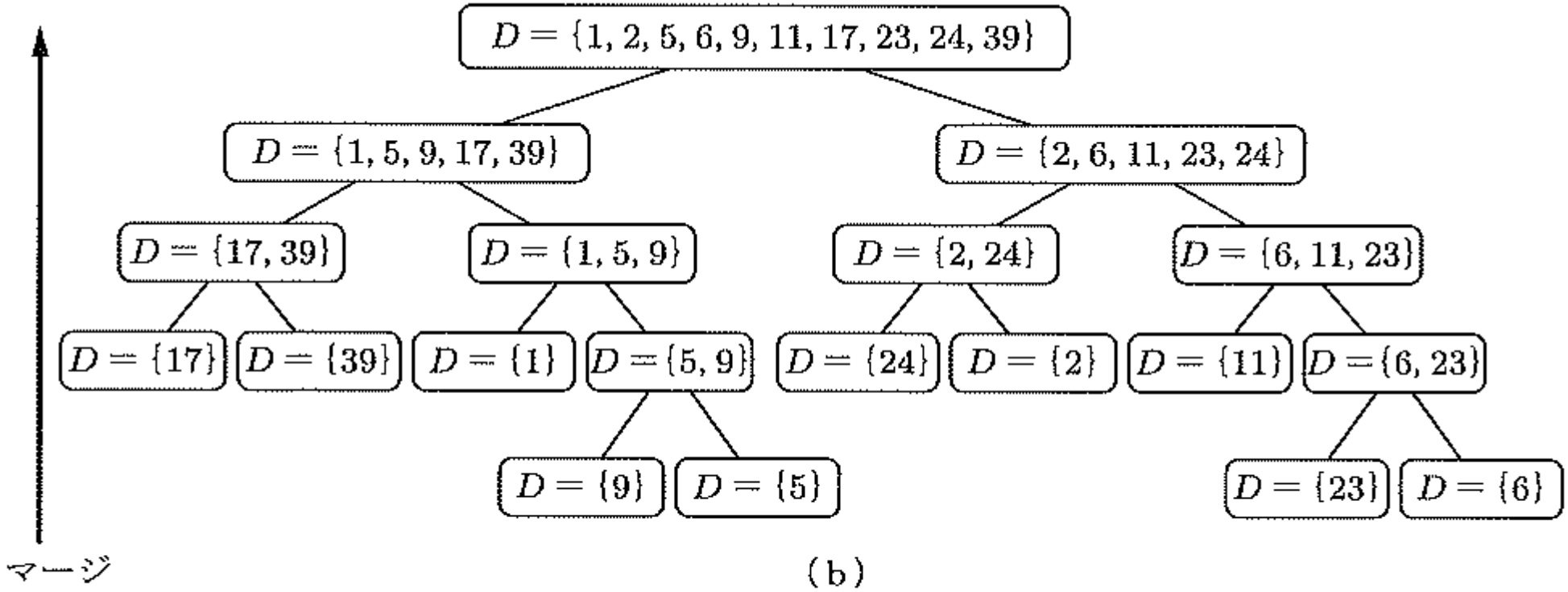
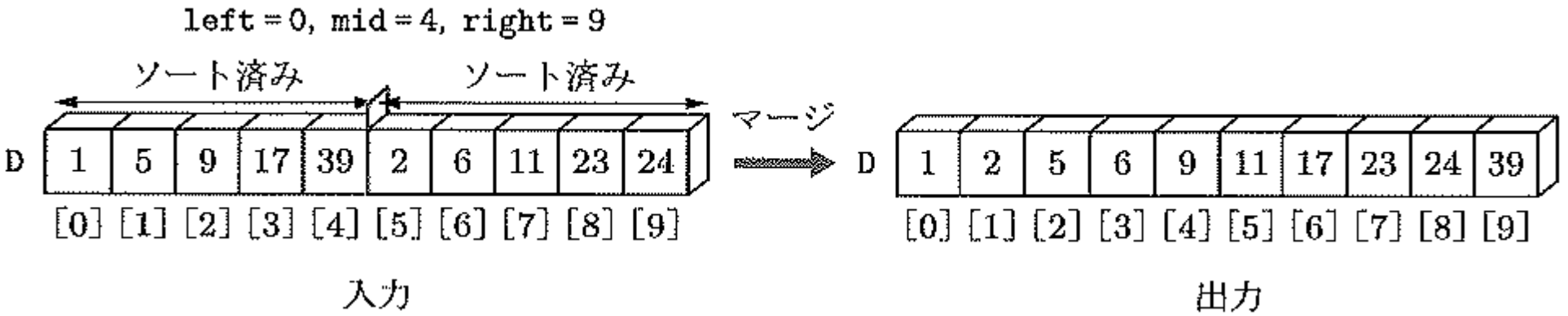
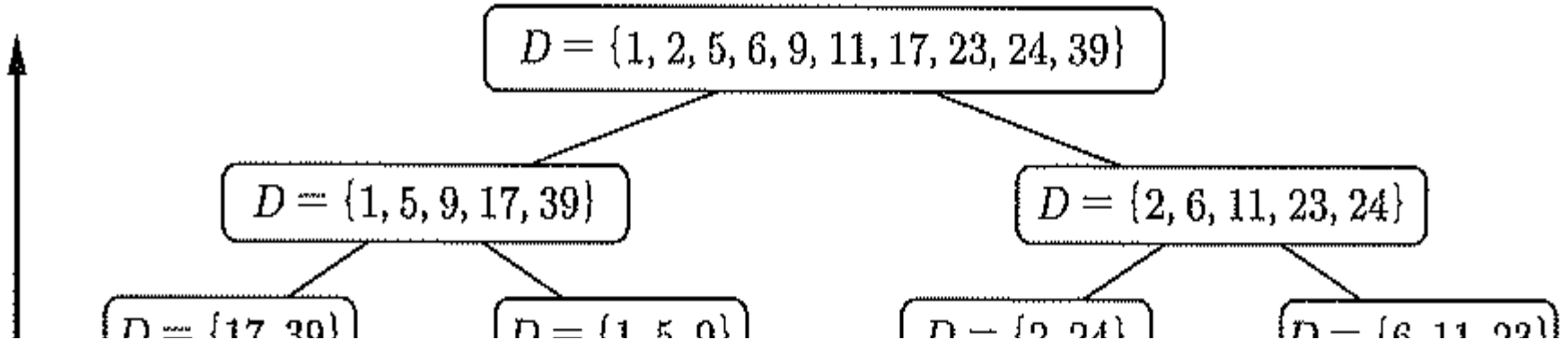


図 7.4　マージソートの再帰木

マージソート


left = 0, mid = 4, right = 9
ソート済み
ソート済み
D
1 5 9 17 39 2 6 11 23 24
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
入力
マージ
D
1 2 5 6 9 11 17 23 24 39
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
出力


$D = \{1, 2, 5, 6, 9, 11, 17, 23, 24, 39\}$
$D = \{1, 5, 9, 17, 39\}$
$D = \{2, 6, 11, 23, 24\}$

マージソート

アルゴリズム 7.3　マージソート

入力：サイズ n の配列 D[0], D[1], ..., D[n-1]

```
mergesort(D, left, right) {
  mid=(left + right)/2;                                  ---(1)
  if (left < mid) mergesort(D, left, mid);               ---(2)
  if (mid+1 < right) mergesort(D, mid+ 1, right);        ---(2)
  merge(D, left, mid, right);                            ---(3)
}
//mergesort(D, 0, n-1)を実行することにより入力全体のソートが実行される.
```

マージソート

マージの手順

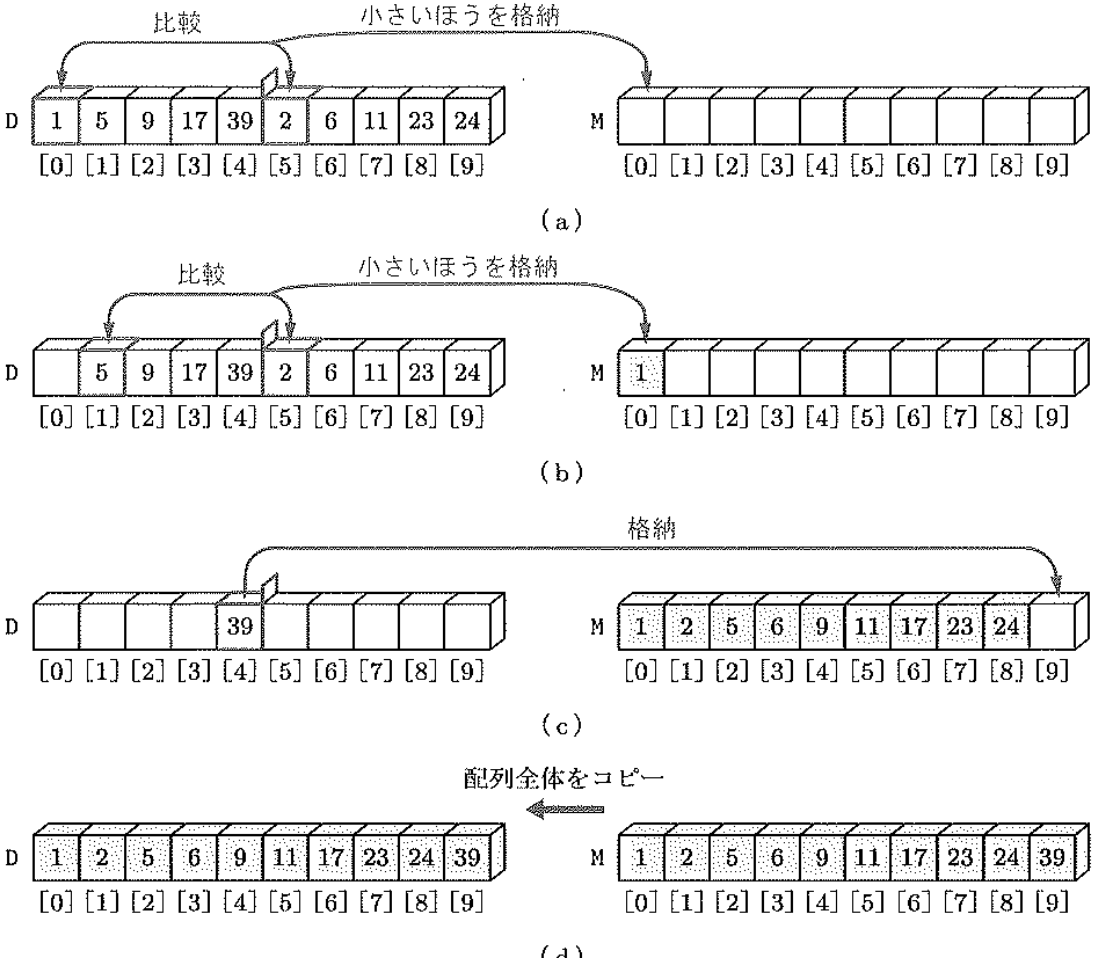
① マージ後のソート列を入れる配列 M を用意する.
② 1つ目のソート済みの列の最小のデータと，2つ目のソート済みの列の最小のデータを比較する.
③ ②の比較において小さいほうのデータをソート済みの列から削除し，配列 M に格納する.
④ ②，③の操作をどちらかのソート済みの列が空になるまで繰り返す.
⑤ 残ったソート済みの列のデータをすべて配列 M に格納する.
⑥ 配列 M のデータをすべて配列 D にコピーする.

マージソート

アルゴリズム 7.4　関数 merge

```
merge(D,left,mid,right)  {
  x=left; y=mid+1;
  for (i=0; i<=right-left; i=i+1)  {
    if (x==mid+1)  { M[i]=D[y]; y=y+1; }          //左のソート列が空の場合
    else if (y==right+1)  { M[i]=D[x]; x=x+1; } //右のソート列が空の場合
    else if (D[x]<=D[y])  { M[i]=D[x]; x=x+1; }
                                          //左のソート列の最小値が小さい場合
    else { M[i]=D[y]; y=y+1; }            //右のソート列の最小値が小さい場合
  }
  for (i=left; i<=right; i=i+1)  { D[i]=M[i]; }  //配列Mを配列Dにコピー
}
```

マージの様子


比較
小さいほうを格納
D
1 5 9 17 39 2 6 11 23 24
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
M
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
(a)
比較
小さいほうを格納
D
5 9 17 39 2 6 11 23 24
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
M
1
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
(b)
格納
D
39
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
M
1 2 5 6 9 11 17 23 24
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
(c)
配列全体をコピー
D
1 2 5 6 9 11 17 23 24 39
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]
M
1 2 5 6 9 11 17 23 24 39
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

マージソートの計算量

クイックソートと同じ考え方で求まる。
計算量の考え方としては、これまでの木に関する計算方法と同じ。

$$T(n) = \underbrace{T\left(\frac{n}{2}\right)}_{\text{左の部分の再帰}} + \underbrace{T\left(\frac{n}{2}\right)}_{\text{右の部分の再帰}} + \underbrace{cn}_{\texttt{mid} \text{ の計算と関数 } \texttt{merge} \text{ の実行}}$$

$$T(n) = O(n \log n)$$