データ構造とプログラミング技法
（第7回）

ーデータの整列ー
交換／併合／分散による方法

ソートアルゴリズムの分類

- キーの比較に基づく方法
  - 選択（n番目に来るキーを**選択**する）
  - 挿入（キーを入れる場所を見つけ、**挿入**する）
  - 交換（キー同士を**交換**する）
  - 併合(整列された短い並びを**併合**していく)
- キーの構造に基づく方法
  - 交換（キー同士を**交換**する）
  - 分散（キーを**分散**させながら整列する）

単純交換法（バブルソート）

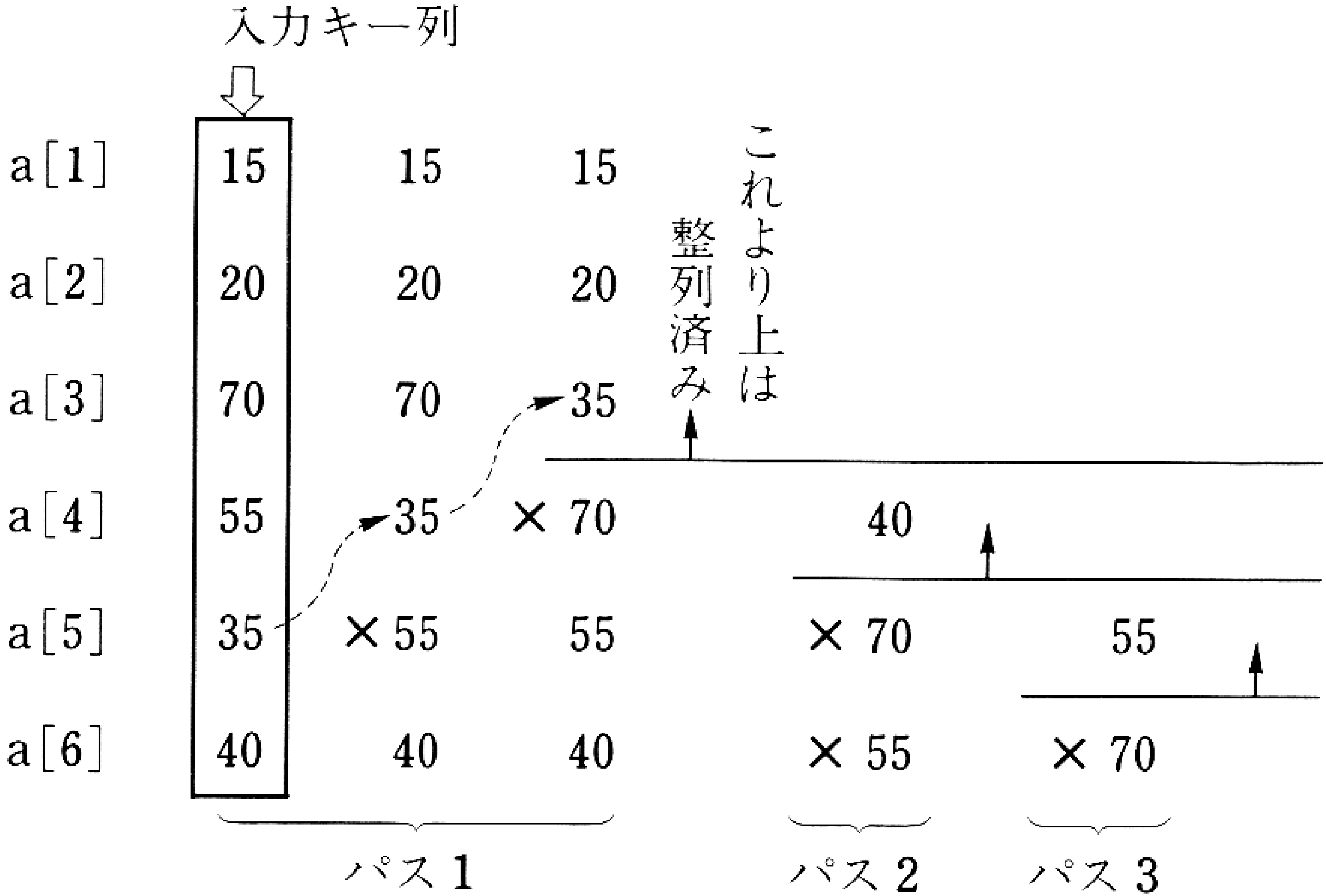


図 7.12　単純交換法の実行例

単純交換法(バブルソート)のアルゴリズム

```
void bubble_sort()
{    int last,sw,i,w;
     last = 1; sw = 0;
     while (sw != N-1) {
          sw = N-1;
          for (i=N-1; i >= last; i--)
               if (a[i] < a[i-1]) {
                    w = a[i]; a[i] = a[i-1]; a[i-1] = w; sw = i;
               }
          last = sw+1;
     }
}
```

←カウンターを下げる下限

←最後に交換を行った場所

図 7·13　単純交換法 (バブル整列法)

シェーカーソート（双方向バブルソート）

バブルソートを双方向に適用する方法

a[1]	15	15	15	15
a[2]	20	20	20	20
a[3]	70	35	35	35
a[4]	55	70	55	40
a[5]	35	55	40	55
a[6]	40	40	70	70

処理のオーダはバブルソートと同じ$O(n^2)$。平均処理効率は単純選択・挿入法に劣るが、概ソート列には場合によって適している。

クイックソートの基本的考え方

あるキーが最終的に得られるソート列の中で正しい位置に来ていることを判定するには？

40 25 35 10 15 55 70 88 95 60

全て小さい　全て大きい

10 15 25 35 40 55 60 70 88 95

クイックソートの実行例

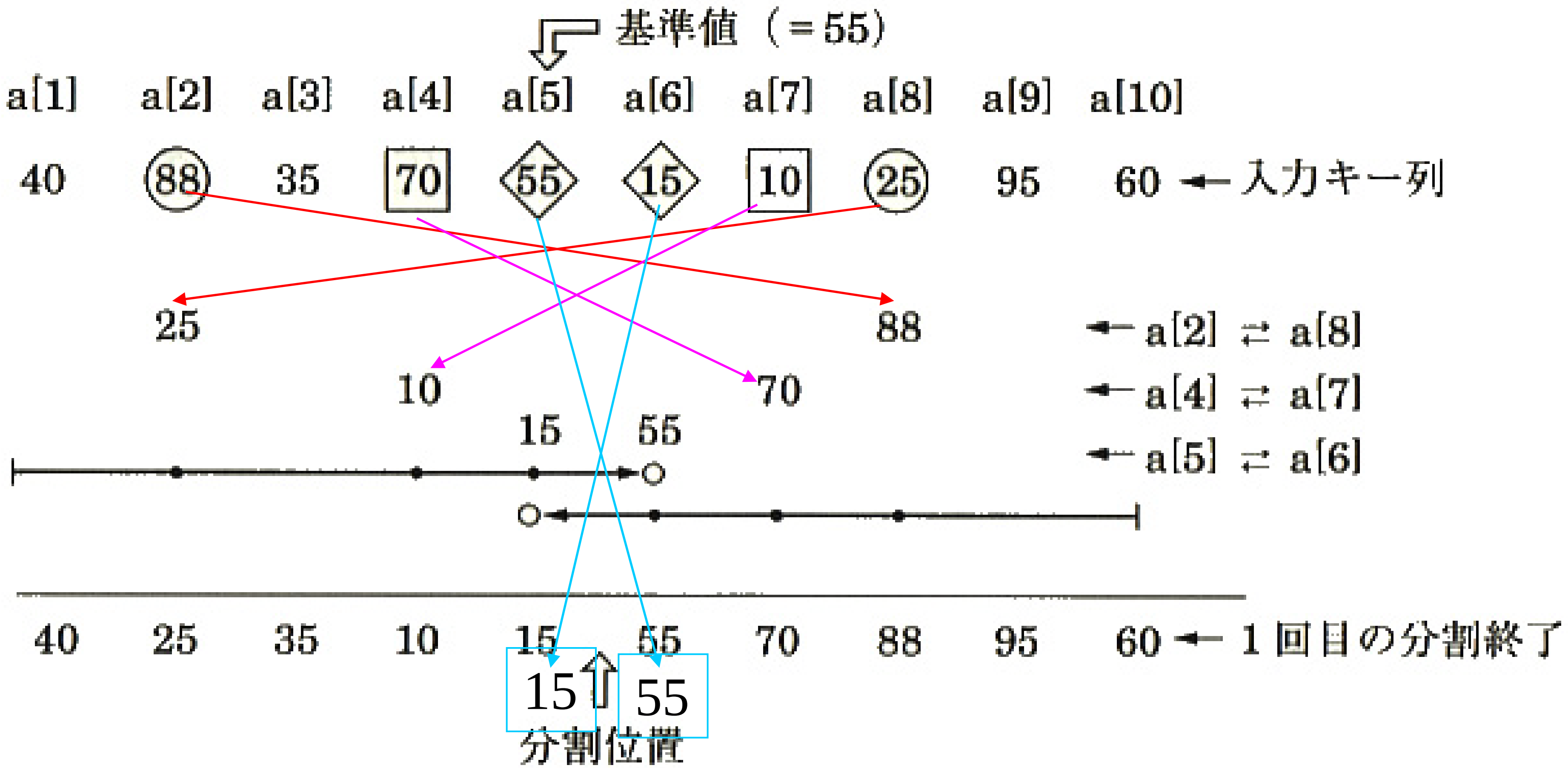


図 7·14　クイックソート法の実行例（最初の１パスのみ）

クイックソートのアルゴリズム

```
int quicksort(int p, int q)
{   int i,j,s,w;
    if (q-p < K)  return straight_sort(p,q);          ①
    else {
        i = p; j = q; s = select();                   ②
        while (i < j) {
            for (; a[i] <= s; i++) ;                  ③
            for (; a[j] >= s; j--) ;
            if (i == q+1 || j == p-1) return 1;       ④
            if (i < j) {w = a[i]; a[i] = a[j]; a[j] = w;}
        }
        if (i == j) {i = i+1; j = j-1;}               ⑤
        quicksort(p,j); quicksort(i,q); return 0;     ⑥
    }
}
```

図 7·15　クイックソート法

クイックソートの手順(間違い)

40 88 35 70 55 15 10 25 95 60 ∞

i j

a[i]>55 a[j]≦55

40 25 35 70 55 15 10 88 95 60 ∞

i j

a[i]>55 a[j]≦55

40 25 35 10 55 15 70 88 95 60 ∞

i j

a[j]≦55 a[i]>55

クイックソートの手順

40 88 35 70 55 15 10 25 95 60 ∞

i j

a[i]>40 a[j]≦40

40 25 35 70 55 15 10 88 95 60 ∞

i j

a[i]>40 a[j]≦40

40 25 35 10 55 15 70 88 95 60 ∞

i j

a[i]>40 a[j]≦40

40 25 35 10 15 55 70 88 95 60 ∞

15 40

クイックソートアルゴリズム

```
int quicksort(int p; int q)
{ int i,j,s;
   if (p-q < K) return straight-sort(p,q);
   else { i=p+1; j= q; s= a[p];
            while (i<j) {
                  for(;(a[i]<=s)&&(i<q);i++);
                  for(;(a[j]>s)&&(j>p);j--);
                  if (i<j) swap(&a[i],&a[j]);
            }
      //ここでa[j]<=s, j<=i が成り立っている。
            swap(&a[p], &a[j]);
            quicksort(p,j-1); quicksort(j+1,q);
            return 0;
   }
}
```

交換による方法:まとめ

- バブルソート、シェーカーソート:
  安定、$O(n^2)$、
- クイックソート法:
  安定ではない、計算のオーダ
  理想的な場合は、比較回数、移動回数ともに$O(n \log(n))$、平均的に最も速い。概ソート列には弱い。

ソートアルゴリズムの分類

- キーの比較に基づく方法
  - 選択（n番目に来るキーを**選択**する）
  - 挿入（キーを入れる場所を見つけ、**挿入**する）
  - 交換（キー同士を**交換**する）
  - 併合(整列された短い並びを**併合**していく）
- キーの構造に基づく方法
  - 交換（キー同士を**交換**する）
  - 分散（キーを**分散**させながら整列する）

併合による方法

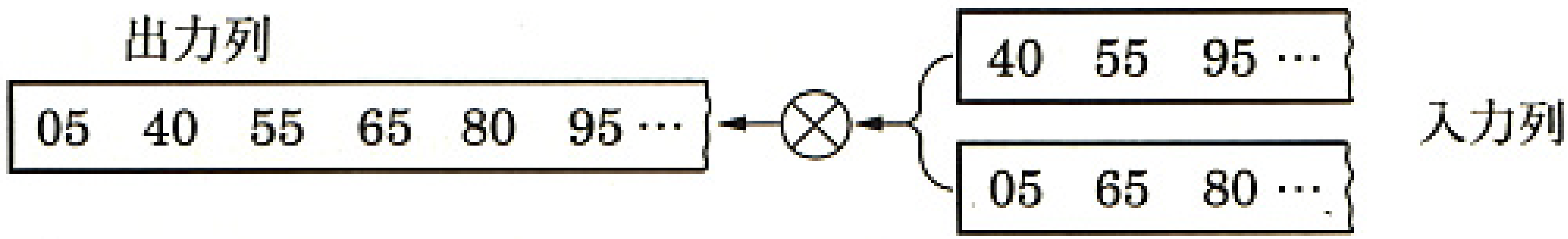


図 7·16 併合の模式図

併合整列法の手順

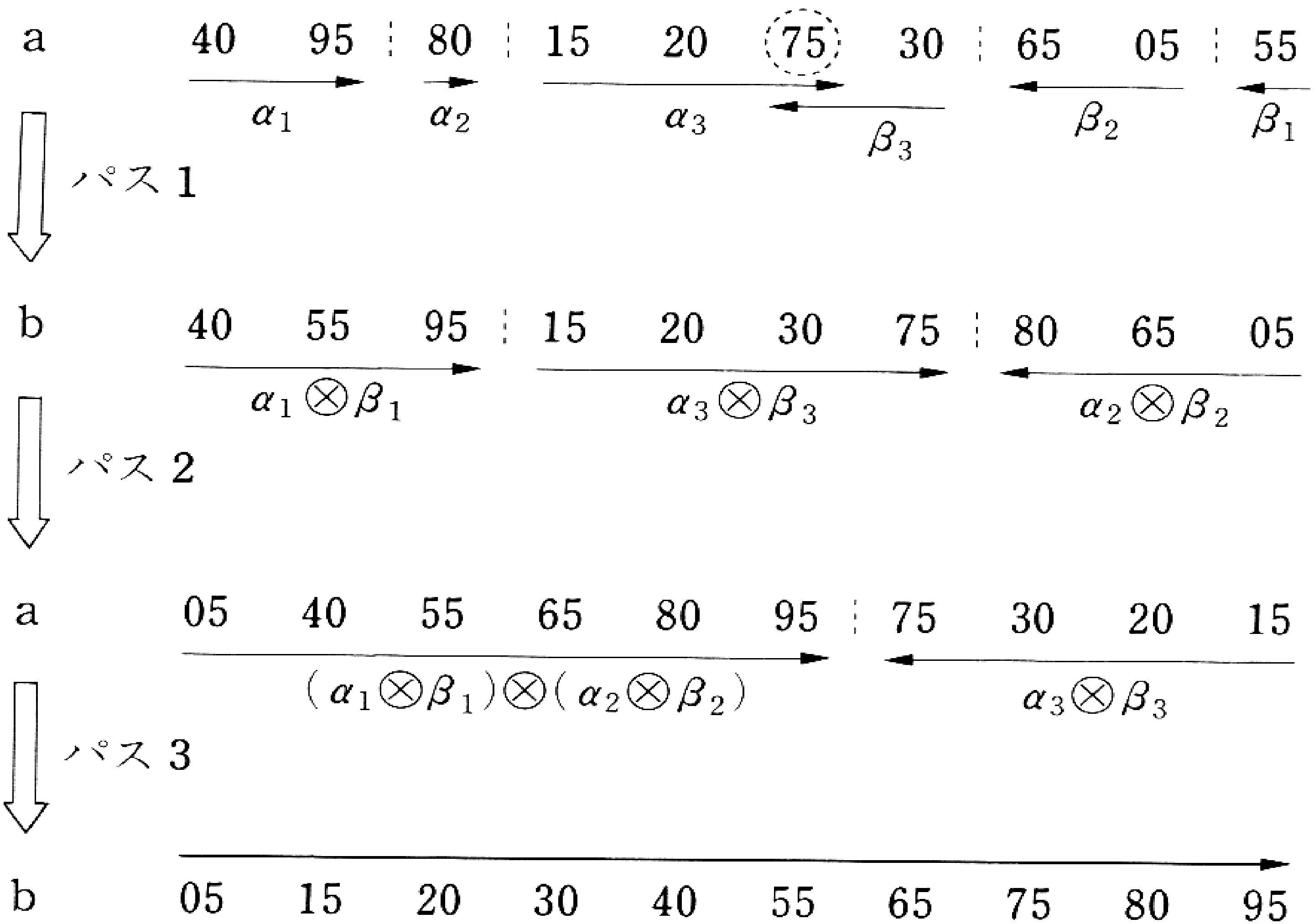


図 7.17　併合整列法の実行例（a，b 二つの配列を使う）

連の連結現象

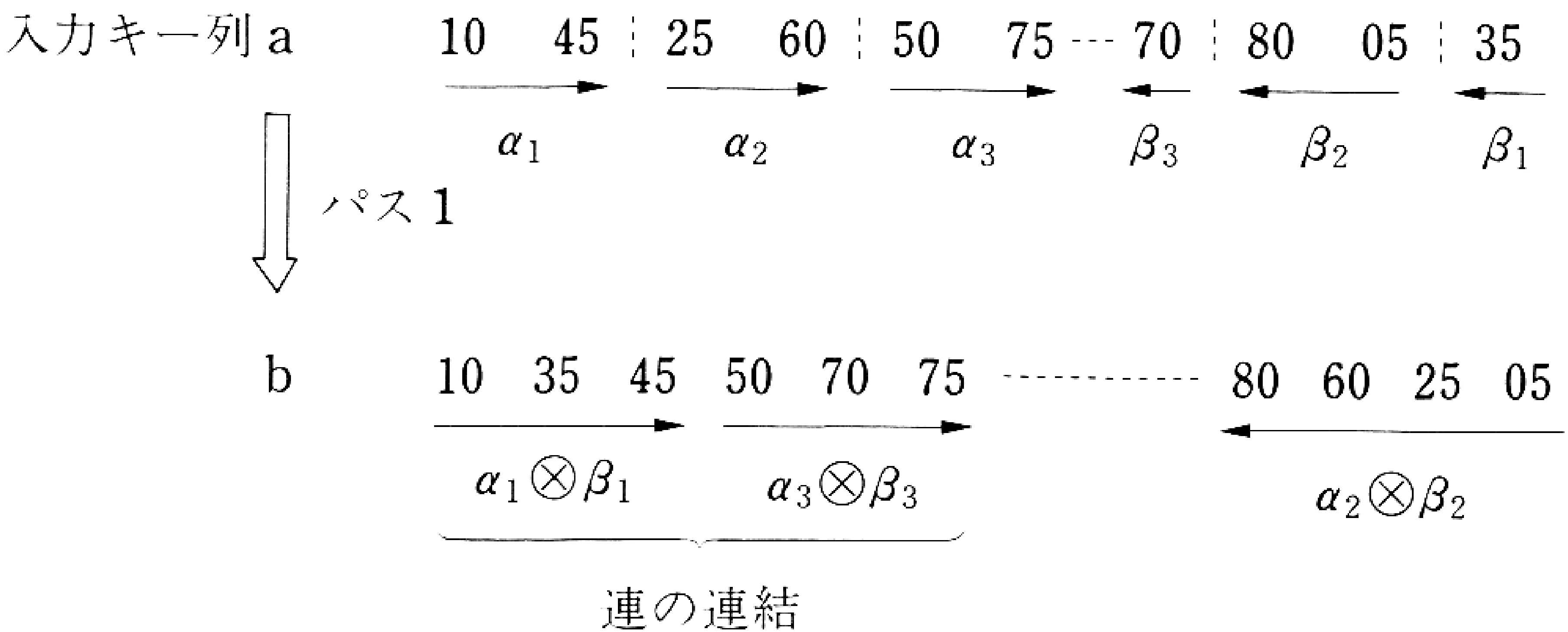


図 7.18 連の連結現象

併合による方法：まとめ

- マージソート： $O(n \log(n))$
  ヒープソート＞マージソート
  ＞クイックソート
  安定。外部整列に向いている

ソートアルゴリズムの分類

- キーの比較に基づく方法
  - 選択(n番目に来るキーを**選択**する)
  - 挿入(キーを入れる場所を見つけ、**挿入**する)
  - 交換(キー同士を**交換**する)
  - 併合(整列された短い並びを**併合**していく)
- キーの構造に基づく方法
  - 交換(キー同士を**交換**する)
  - 分散(キーを**分散**させながら整列する)

交換による方法:基数交換法

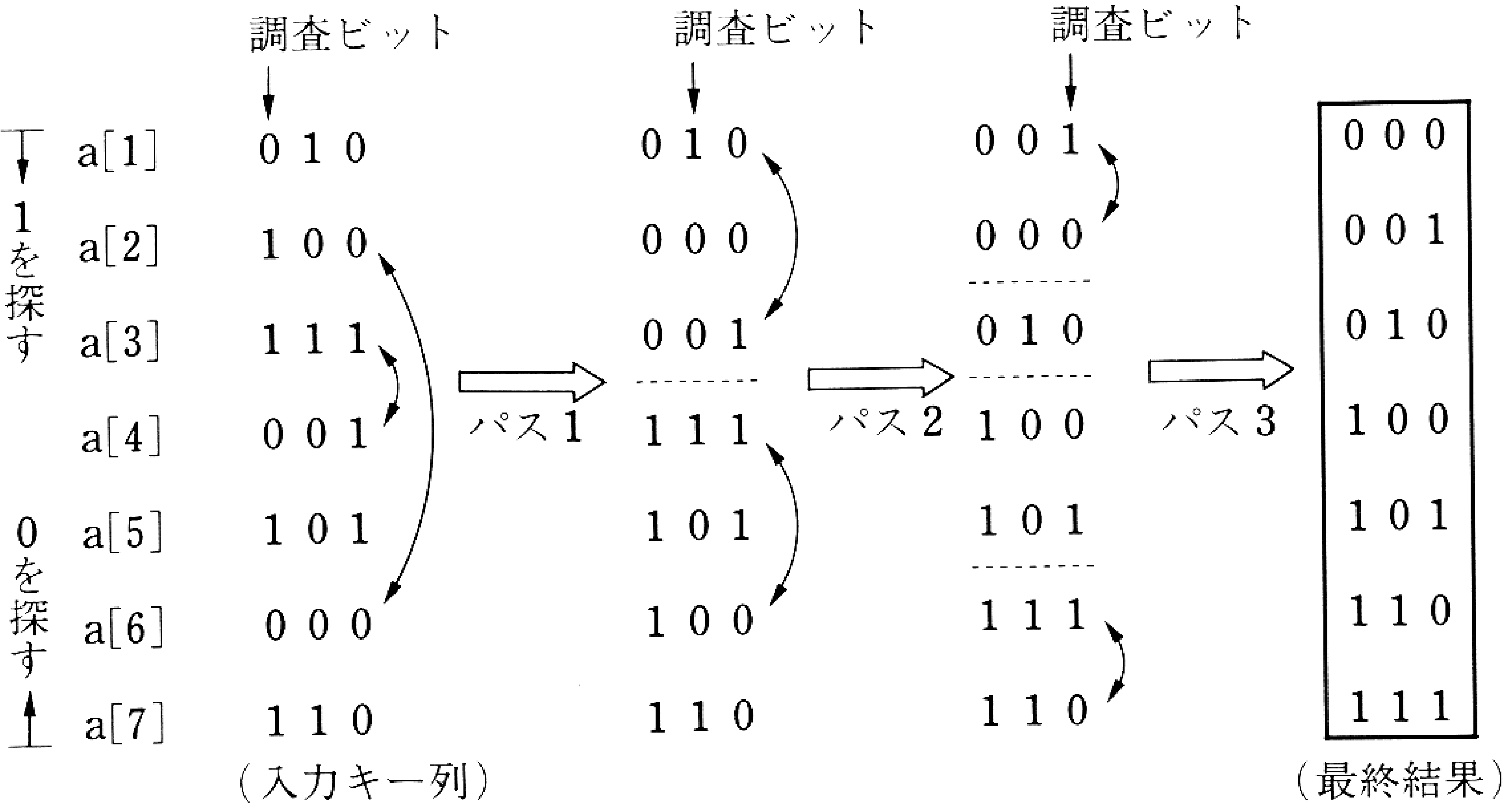
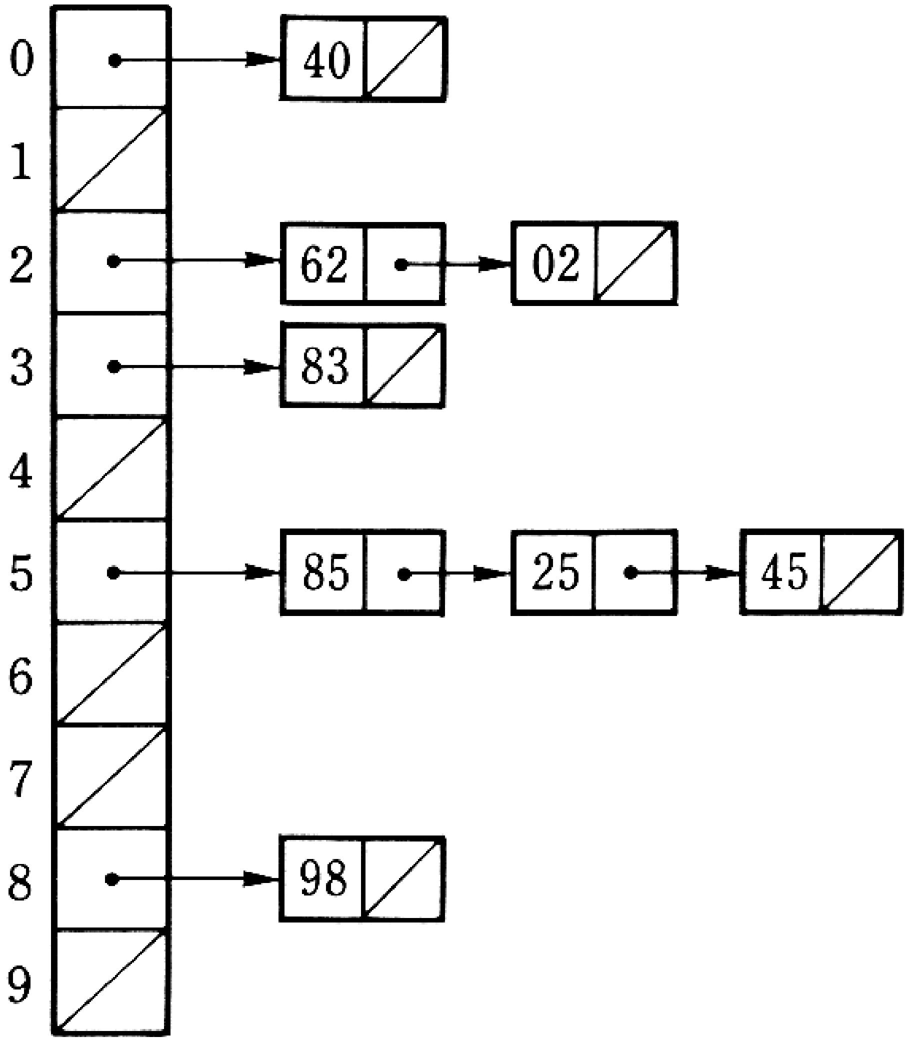
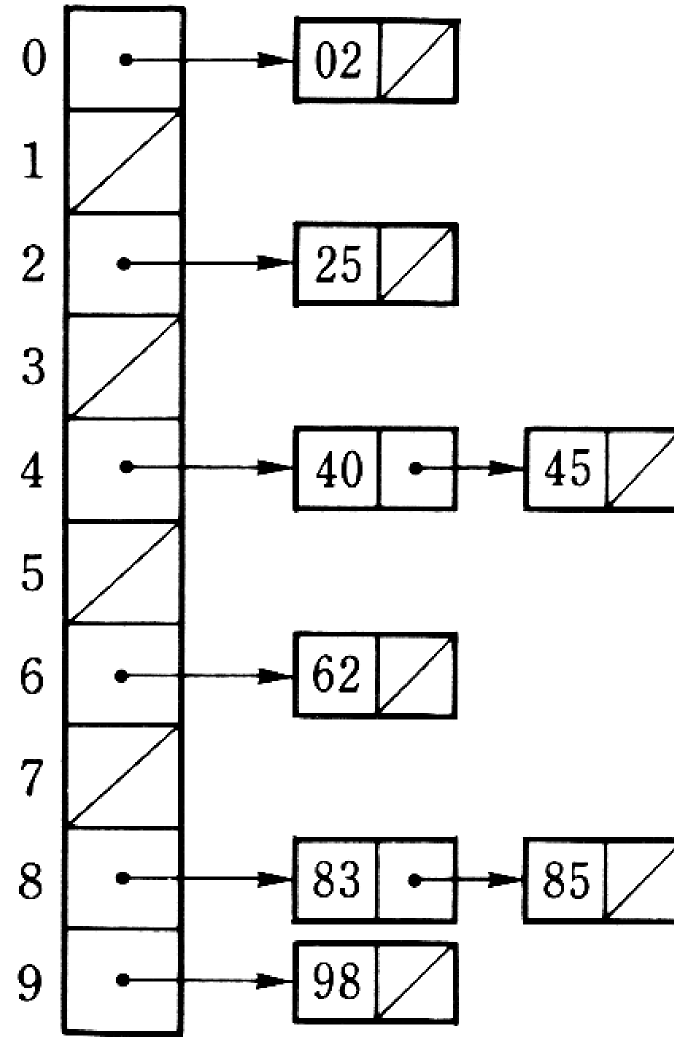


図 7.19　基数交換法の実行例

分散による方法:基数整列法



（ａ） １の桁による分散



（ｂ） 10の桁による分散

キーの構造による方法：まとめ

- 基数交換法

  $O(n \log(n))$
  安定ではない

- 基数整列法

  $O(n \log(n))$　安定　その場整列
  ではない

P178問題：2

```
int NonRecursiveQuicksort(int n);
{
        int i, j, s, w;
        stack S;
        push(1,n,S);
        while (!empty(S)) {
                pop(p,q,S);
                if (q-p<K) straight_sort(p,q);
                else {
                        i=p; j=q; s=select();
                        while (i<j){
                                for (;a[i]<=s;i++);
                                for (;a[j]>=s; j--);
                                if (i==q+1 || j==p-1) return 1;
                                if (i<j) {w=a[i]; a[i]=a[j];a[j]=w;}
                        }
                        if (i==j) {i=i+1; j=j-1;}
                        if  (p<j) push(p,j,S) ;  if (i<q) push(i,q,S);
                }
        }
}
```