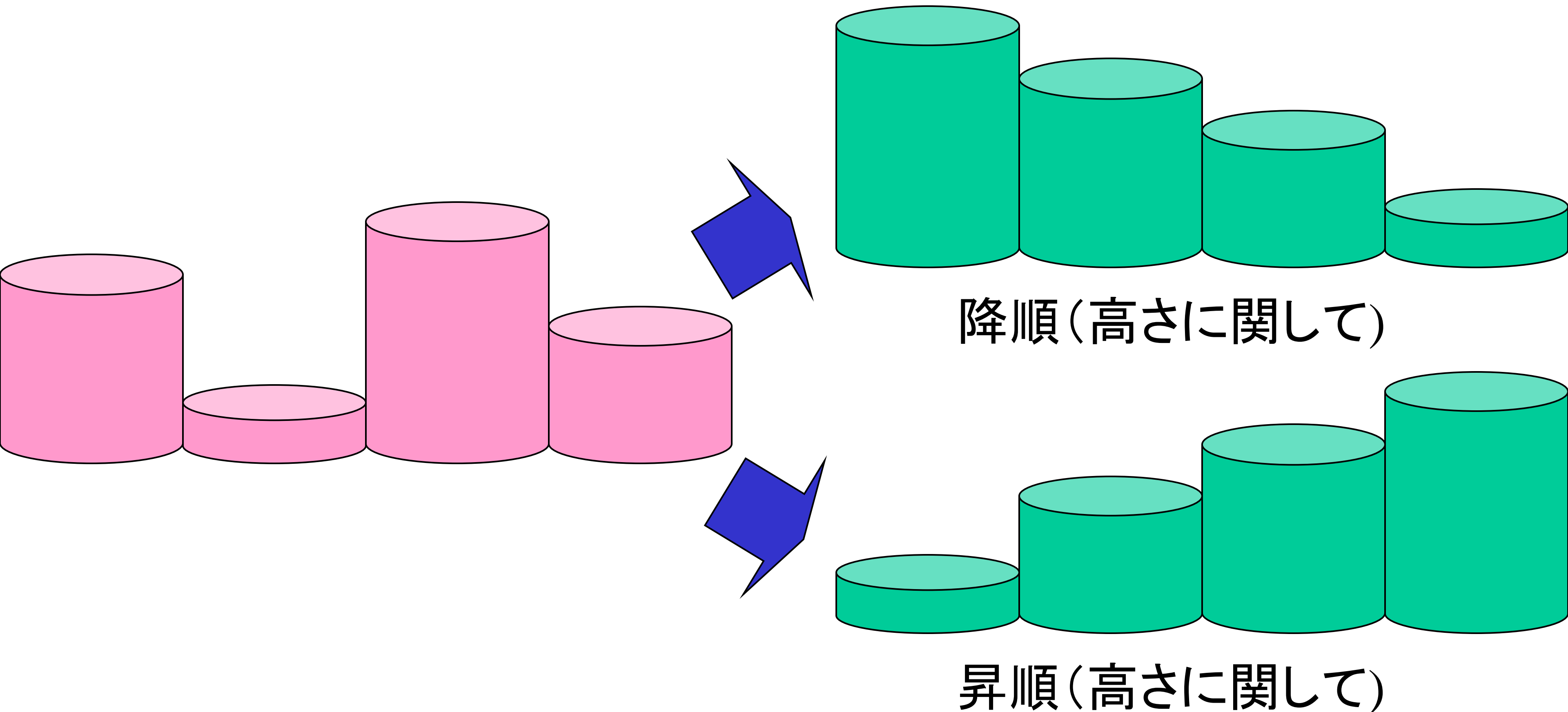
データ構造とプログラミング技法
（第6回）

ーデータの整列ー
選択／挿入による方法

データの整列

- 全順序が定義されている集合の部分集合が与えられたとき、この順序にしたがって、各要素を並べ替える操作。



安定なソート

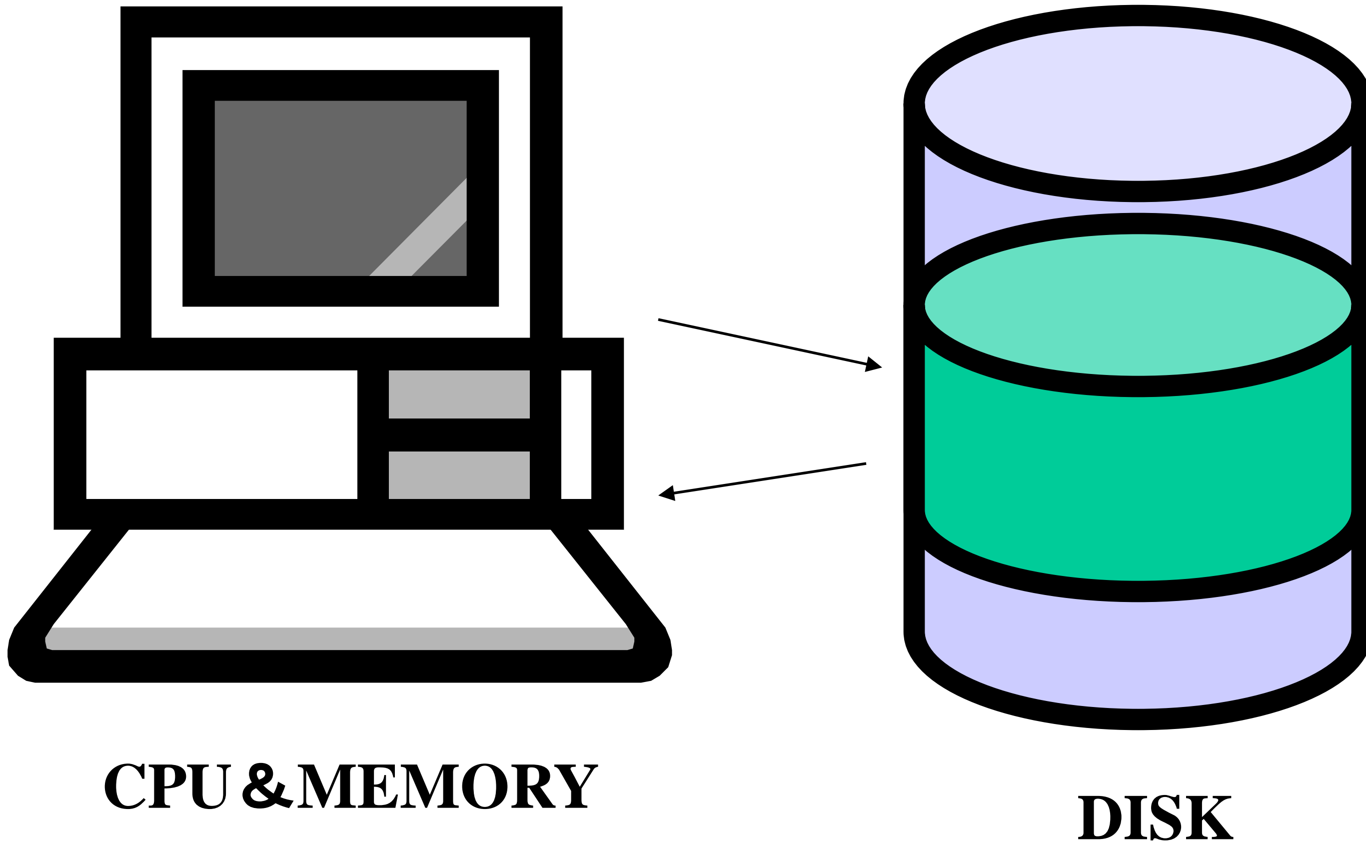
年齢順

田辺	30	800	1
佐々木	40	950	5
田中	40	800	1
山田	49	1050	4

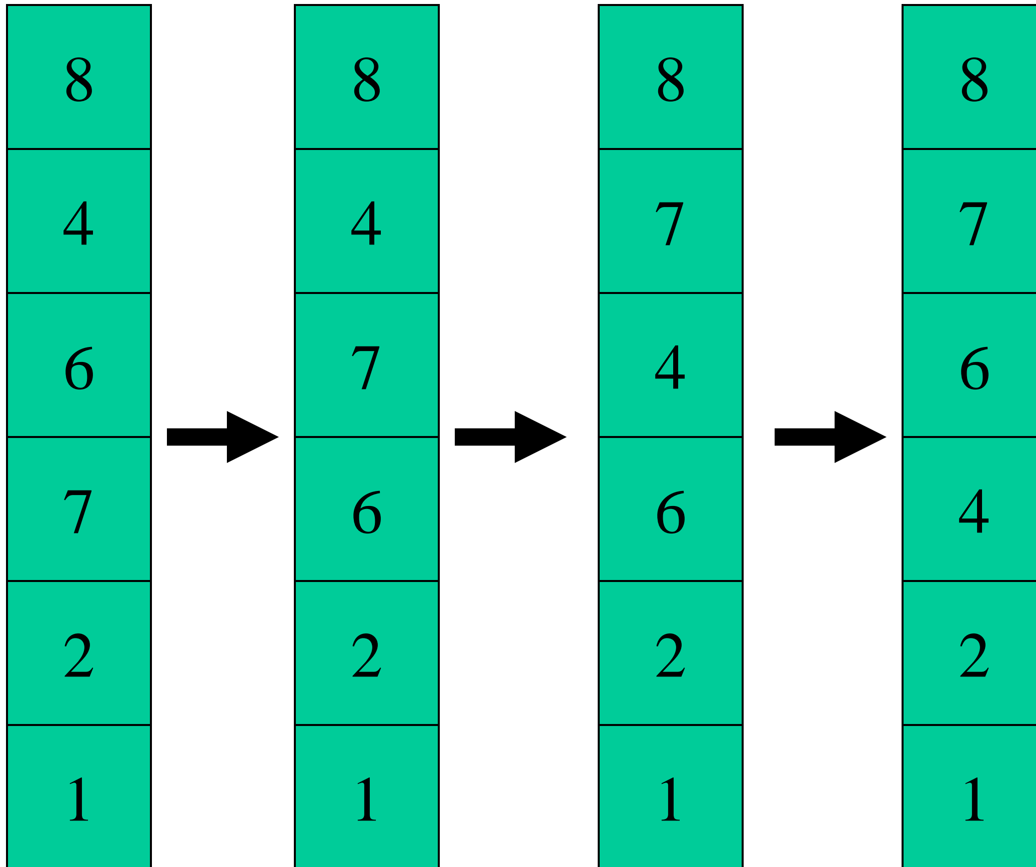
収入順（安定）

田辺	30	800	1
田中	40	800	1
佐々木	40	950	5
山田	49	1050	4

内部ソート/外部ソート



その場整列



余分なメモリ領域を必要としないソート

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）

単純選択法のアルゴリズム

```
void straight_selection()
{    int i,j,p,w;
     for (i=0; i < N-1; i++) {
          p = i; w = a[p];
          for (j = i+1; j < N; j++)
               if (a[j] < w) {p = j; w = a[p];}    ①
          /*w = a[p];*/ a[p] = a[i]; a[i] = w;    ②
     }
}
```

図 7·1　単純選択法

単純選択法

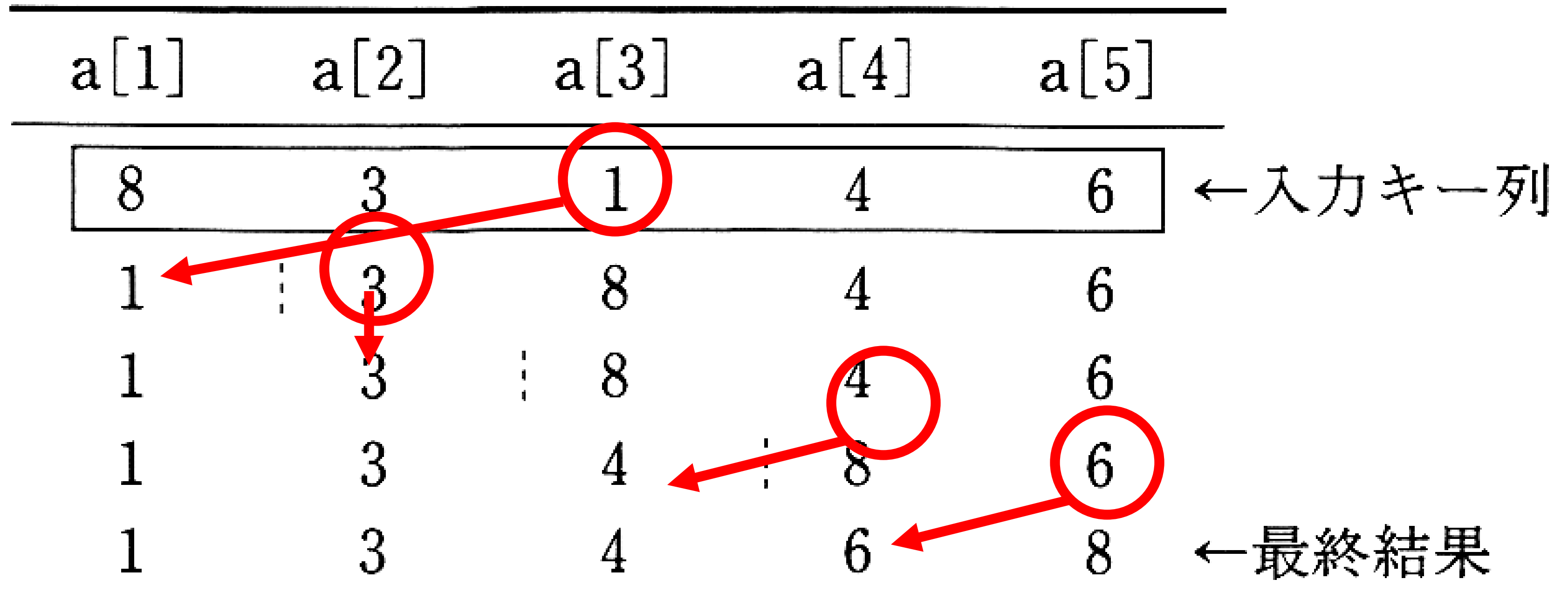
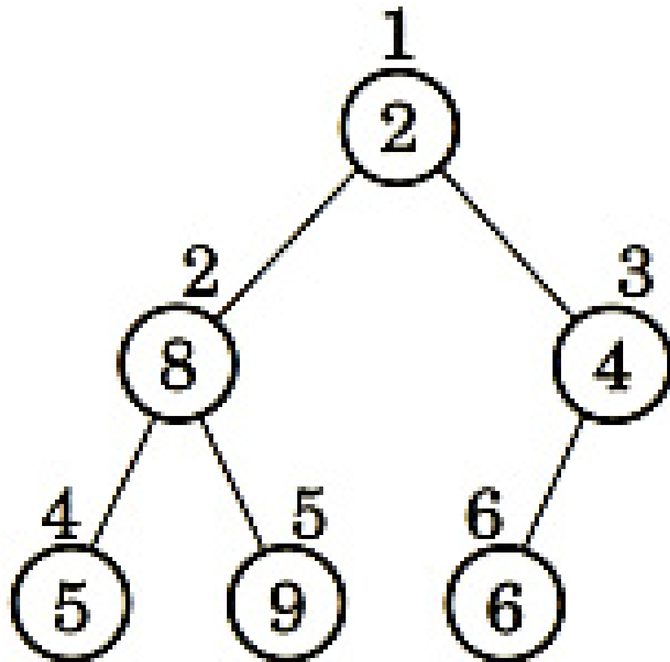
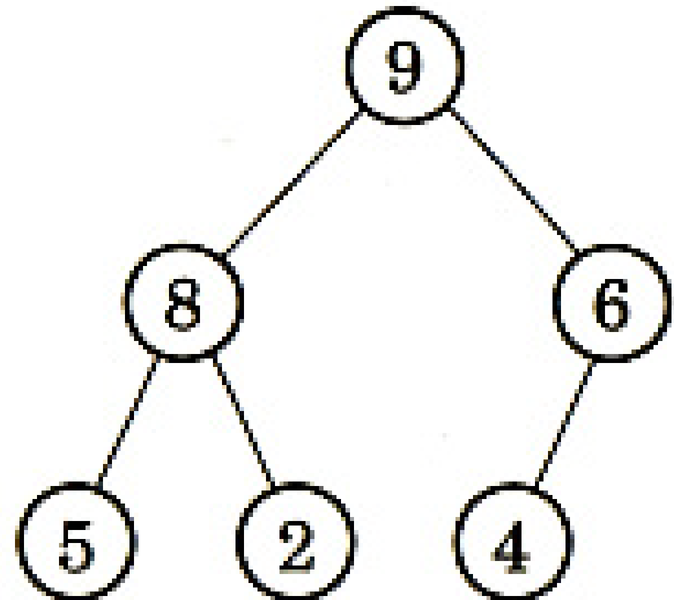


図 7.2　単純選択法の実行列

ヒープソート:ヒープとは？



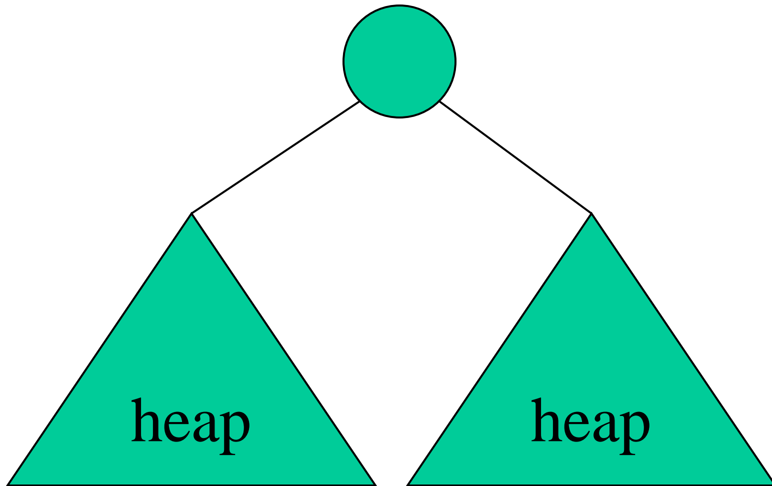


（a） 完全二分木　　　（b） ヒープ

図 7·3　完全二分木とヒープ

ヒープの作り方

部分木がすでにヒープになっている段階から考える.



ヒープの上にあるノードが付け加わると…

Shiftによるヒープの作り方

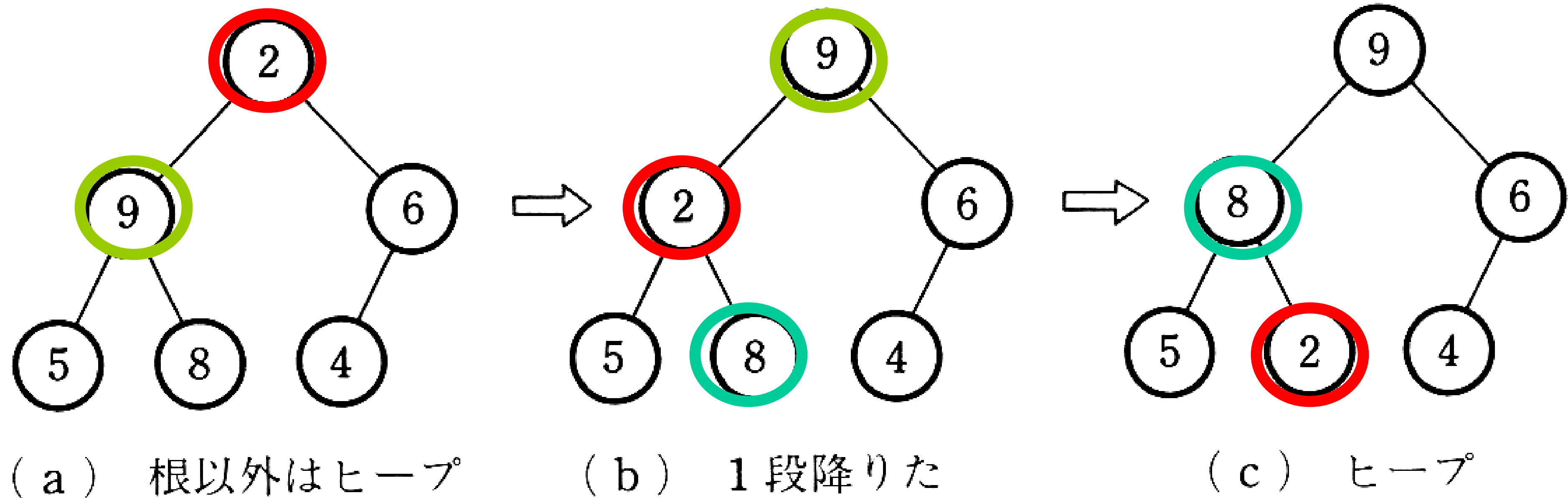
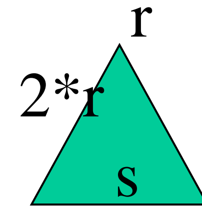


図 7.4 ふるい（sift）操作の例

Shiftの手続き

完全二分木の第r要素以降、s番目までをheapにする。

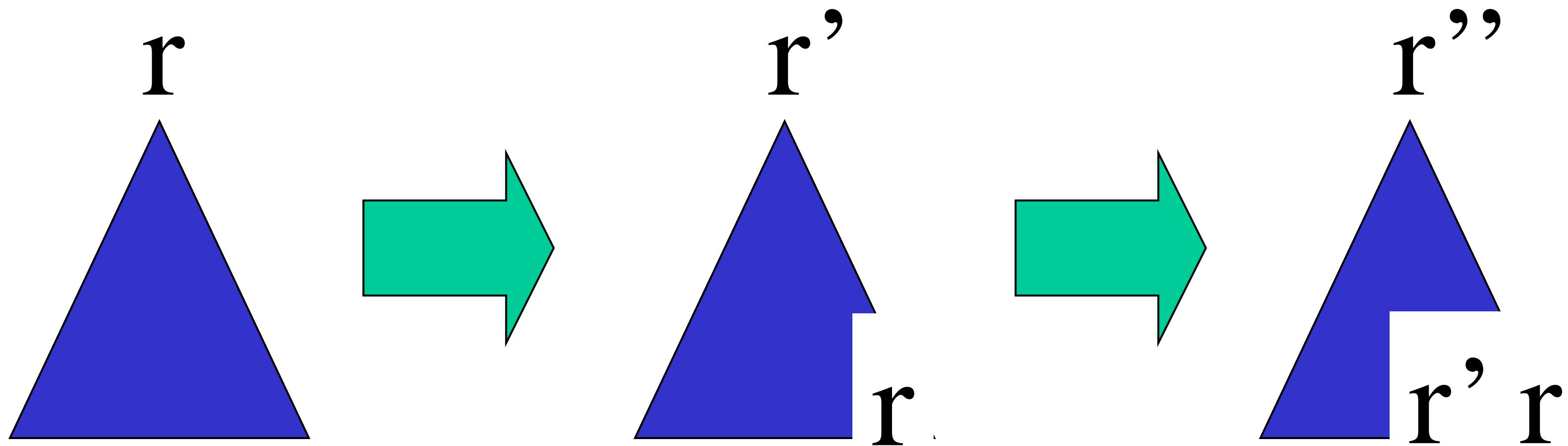



```
void sift(int r, int s)
{   int i,j,w;
    i = r; j = 2*r; w = a[r];
    while (j <= s) {
        if (j < s && a[j] < a[j+1]) j++;     ①
        if (w >= a[j])  break;
        a[i] = a[j]; i = j; j = 2*i;
    }
    a[i] = w;
}
```

図7·5　ふるい操作

ヒープソートの基本的考え方

ヒープの根は最大値なので、それを最後尾にまわせばよい。

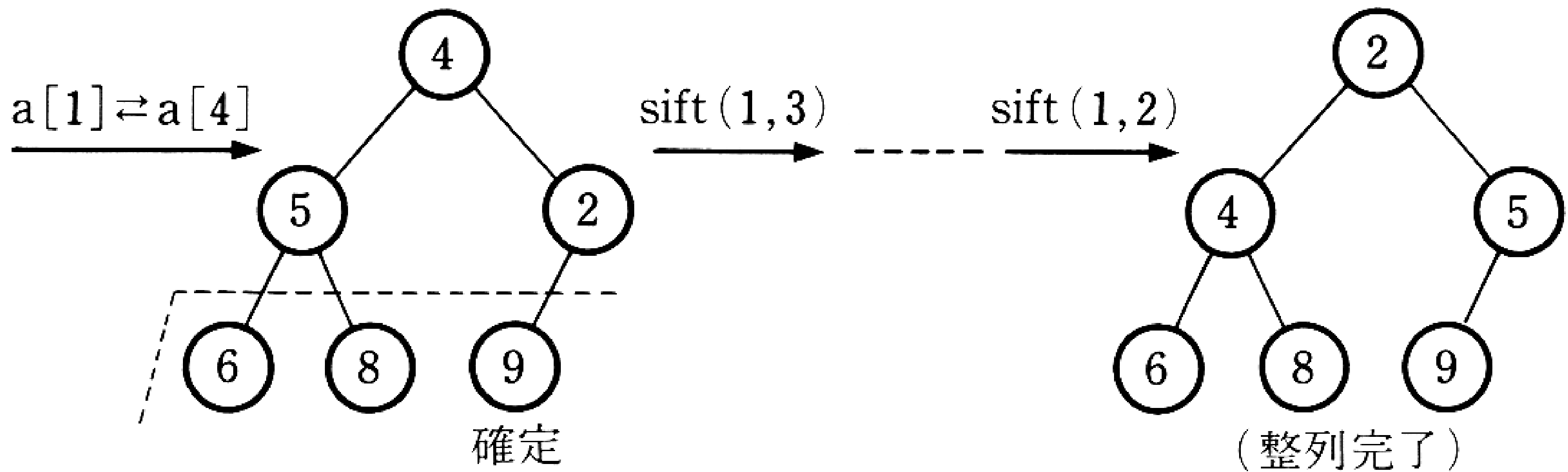


ヒープソートのアルゴリズム

```
void heapsort()
{    int r,m,w;
     r = N/2;                                 }①
     while (r > 0) sift(r--,N);               }
     m = N;                                                                          }②
     while (m > 1) {w = a[1]; a[1] = a[m]; a[m] = w; sift(1,--m);}                   }
}
```

図 7·6　ヒープ整列法

ヒープソートの計算過程



選択による方法:まとめ

- 単純選択法:
  安定ではない、キー比較回数$O(n^2)$
  キー移動回数n-1
- ヒープソート法:
  安定ではない、計算のオーダ
  $O(n \log(n))$、ヒープの形が徐々に変化するので、迅速に最大のキーが選択できる。

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)

単純挿入法のアルゴリズム

```
void straight_insertion()
{   int i,j,w;
    a[0] = M;
    for (i = 2; i < N; i++) {
        w = a[i]; j = i-1;
        while (a[j] > w) {a[j+1] = a[j]; j--;}   ①
        a[j+1] = w;
    }
}
```

図 7·8　単純挿入法

単純挿入法の実行例

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	
ー	8	3	1	4	6	←入力キー列
$-\infty$	3	8	1	4	6	
$-\infty$	1	3	8	4	6	
$-\infty$	1	3	4	8	6	
$-\infty$	1	3	4	6	8	←最終結果

図 7.9　単純挿入法の実行例

シェル法の実行例

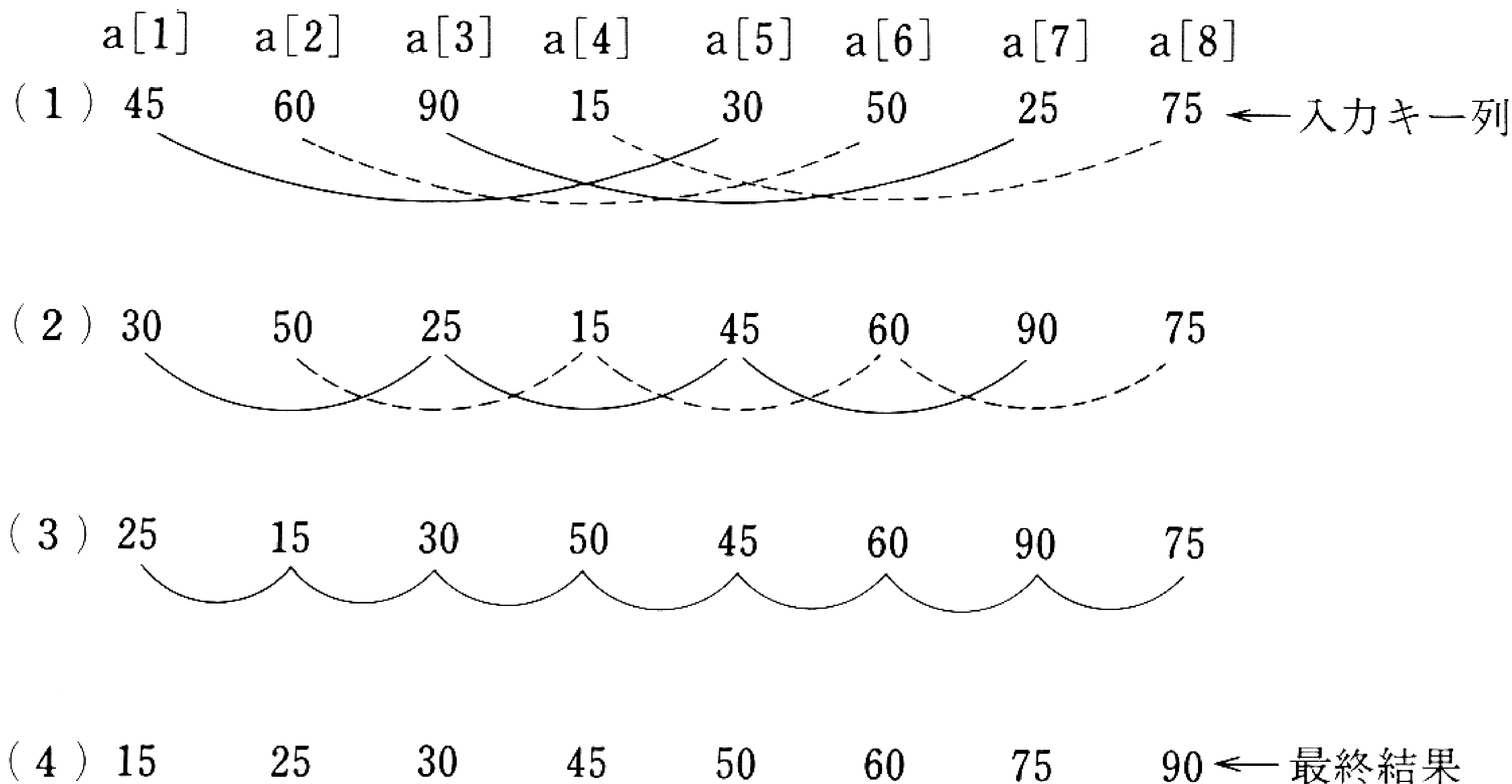


図 7.10　シェル法の実行例（減少増分列は 4, 2, 1）

シェル法のアルゴリズム

```
void Shellsort()
{    int i,j,k,w;
     for (k=0; k < M; k++)
          for (i = d[k]+1; i <= N; i++) {
               w = a[i]; j = i-d[k];
               while (j > 0 && a[j] > w) {
                    a[j+d[k]] = a[j]; j = j-d[k];
               }
               a[j+d[k]] = w;
          }
}
```

図 7·11　シェル法 (減少増分列は d[0], ···, d[M−1])

挿入による方法:まとめ

- 単純挿入法:
  安定、キー比較回数$O(n^2)$、
  キー移動回数$O(n^2)$
- シェル法:
  安定ではない、計算のオーダは解析的に議論しにくい、概整列の効果が大きい。