

# 応用情報

1-2-2アルゴリズム

1-2-3プログラミング

60151039 東 裕之輔

# 探索アルゴリズム

- 線形探索・・・前から順に
- 2分探索・・・真ん中のデータを取り、比較していく
- ハッシュ表探索・・・ハッシュ関数を利用し、データからハッシュ値を求める。

# 2分探索

- 2つのデータの関係から前後どちらに目的のデータがあるか特定する。

探索されるデータ

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|

探索するデータ

|   |
|---|
| 3 |
|---|



5で分けると

|   |   |   |   |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
|---|---|---|---|



2で分けると

|   |   |
|---|---|
| 3 | 4 |
|---|---|



3で分けようとする  
と見つかる。

$2^{\log_2 n}$ 回分けるのでオーダーは  
 $O(\log n)$   
となる！

# 整列アルゴリズム

- バブルソート
- 選択ソート
- 挿入ソート
- シェルソート
- クイックソート
- ヒープソート
- マージソート

# 単純挿入法の実行例

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] |        |
|------|------|------|------|------|------|--------|
| —    | 8    | ∴ 3  | 1    | 4    | 6    | ←入力キ一列 |
| —∞   | 3    | 8    | ∴ 1  | 4    | 6    |        |
| —∞   | 1    | 3    | 8    | ∴ 4  | 6    |        |
| —∞   | 1    | 3    | 4    | 8    | ∴ 6  |        |
| —∞   | 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

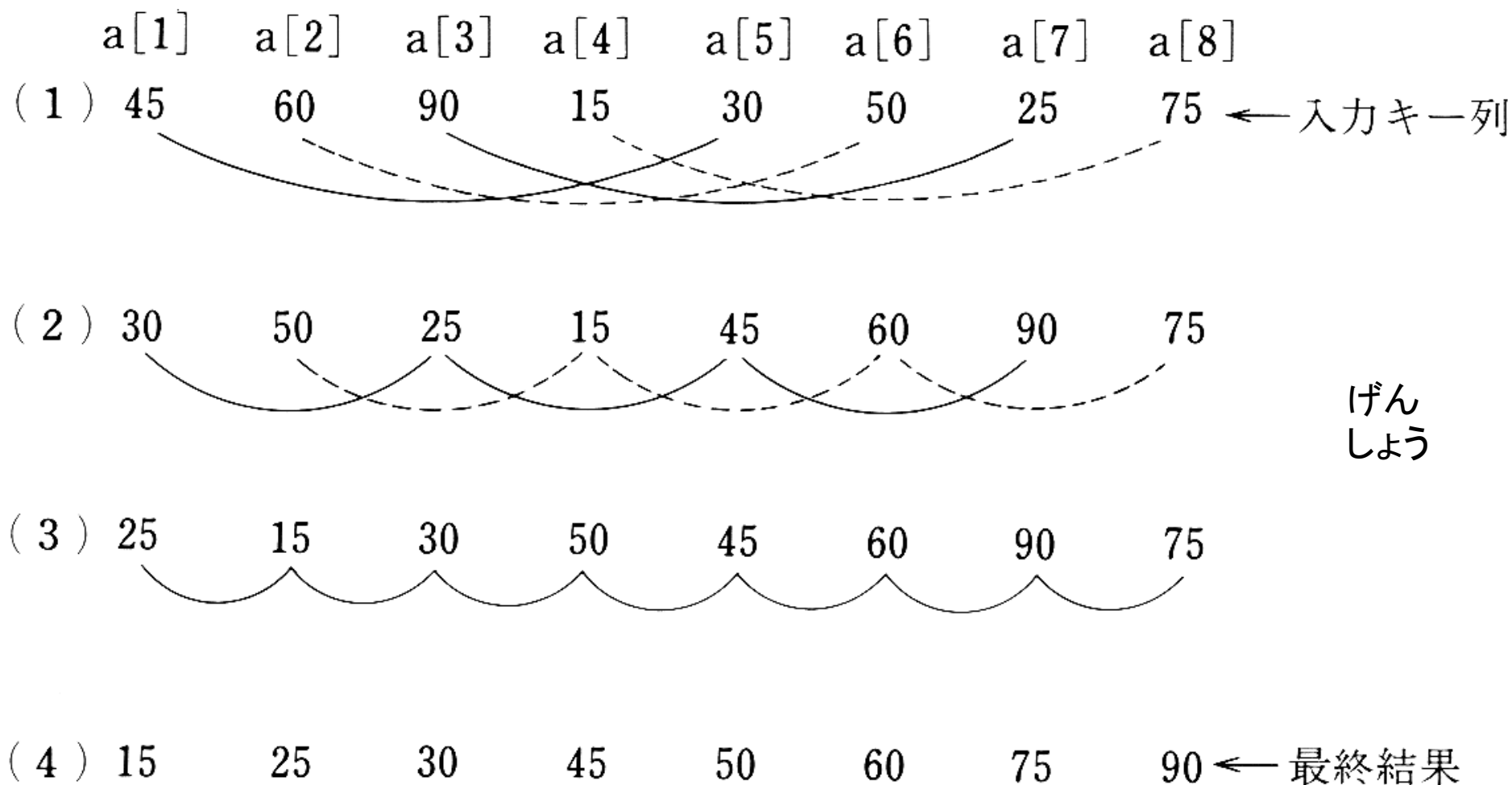


図 7.10 シェル法の実行例 (減少増分列は 4, 2, 1) 1 の時は挿入法

# 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

# クイックソートの手順

40 88 35 70 55 15 10 25 95 60  $\infty$

$i$   $j$

$\rightarrow$   
 $a[i] > 40$

$\leftarrow$   
 $a[j] \leq 40$

40 25 35 70 55 15 10 88 95 60  $\infty$

$i$   $j$

$\rightarrow$   
 $a[i] > 40$

$\leftarrow$   
 $a[j] \leq 40$

40 25 35 10 55 15 70 88 95 60  $\infty$

$i$   $j$

$\rightarrow$   
 $a[i] > 40$

$\leftarrow$   
 $a[j] \leq 40$

40 25 35 10 15 55 70 88 95 60  $\infty$

15 40



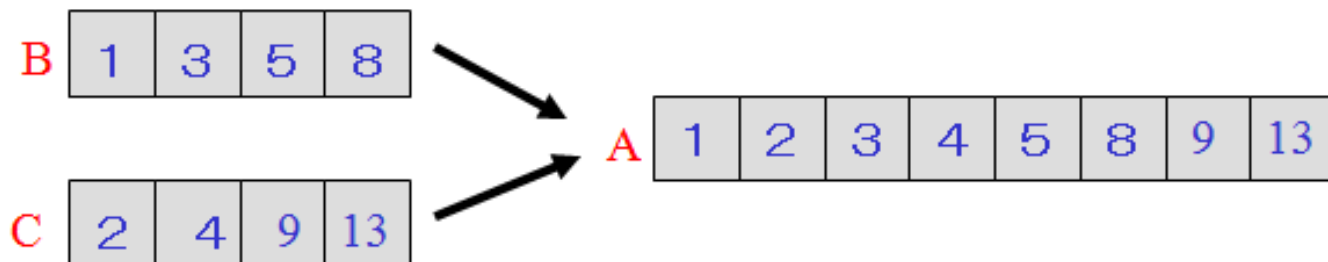
# マージソートの手順

1.分解

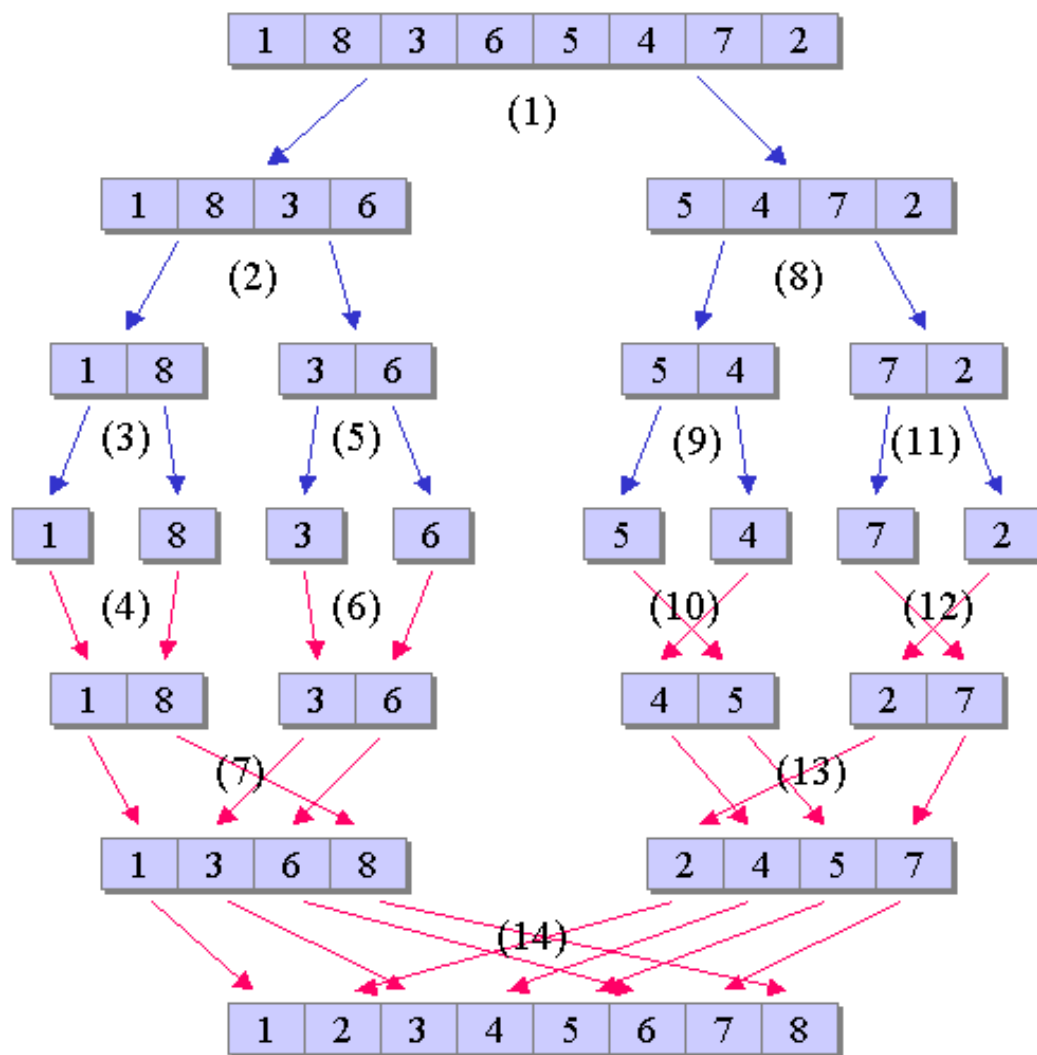


2.併合

↓最後のマージの形



# 图解



1.分解

2.併合

# ヒープソート

- ヒープとは  
親ノードのほうが子ノードより大きい木構造

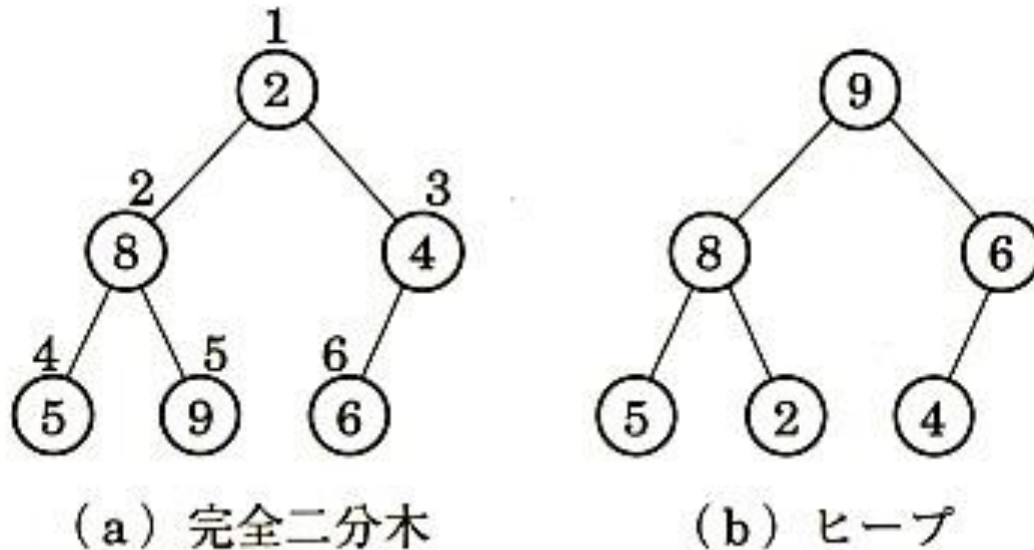
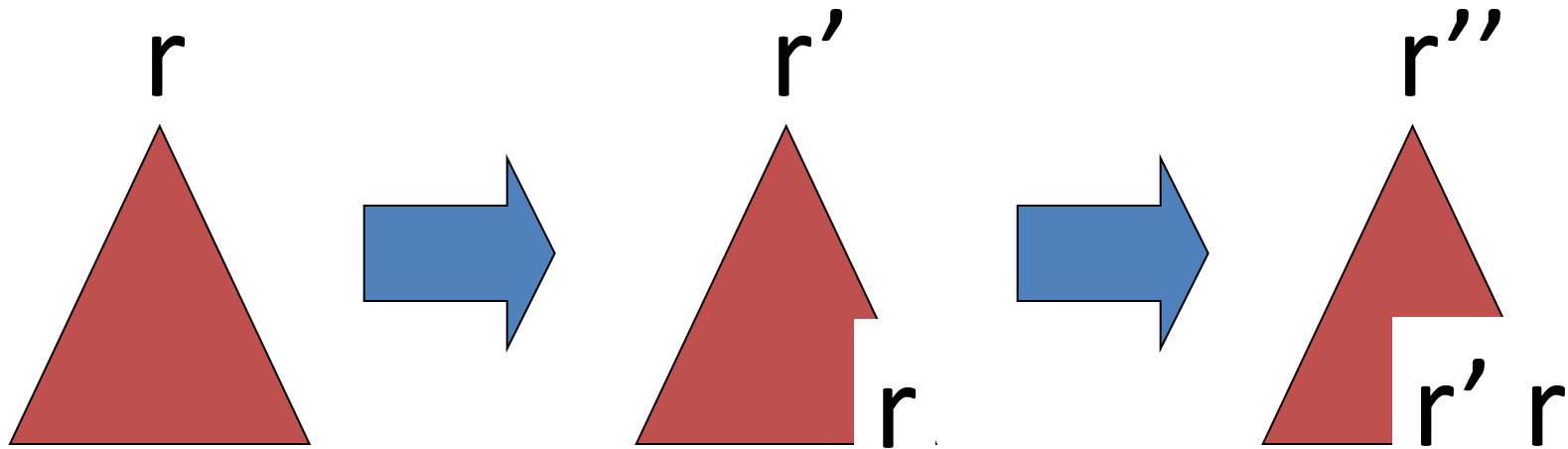


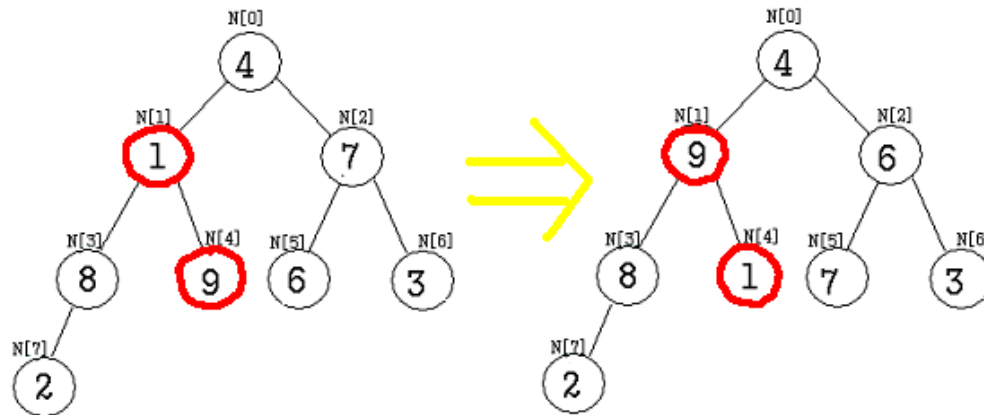
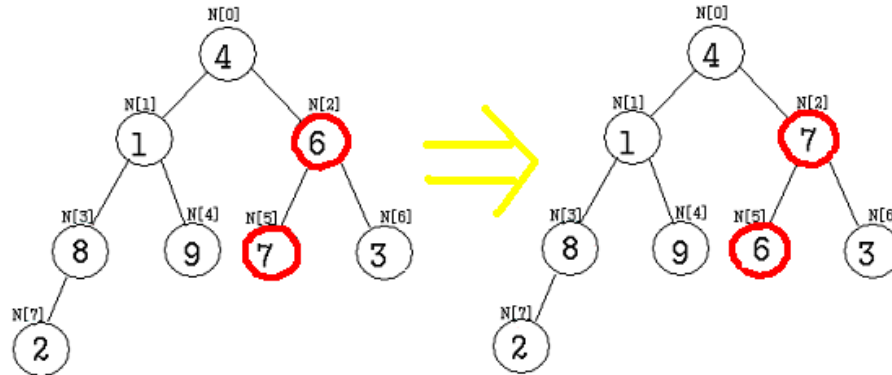
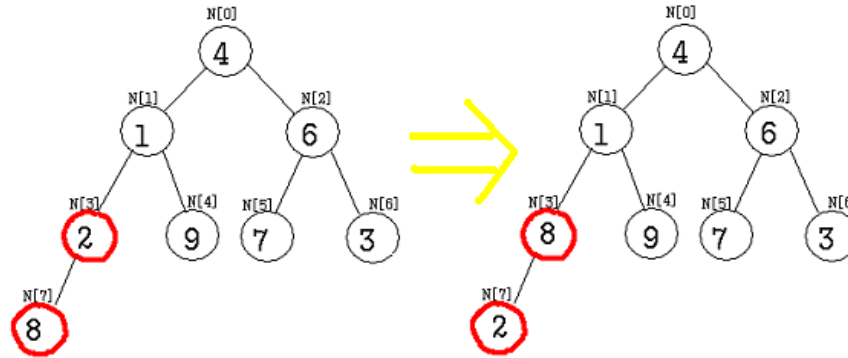
図 7・3 完全二分木とヒープ

# ヒープソートの基本的考え方

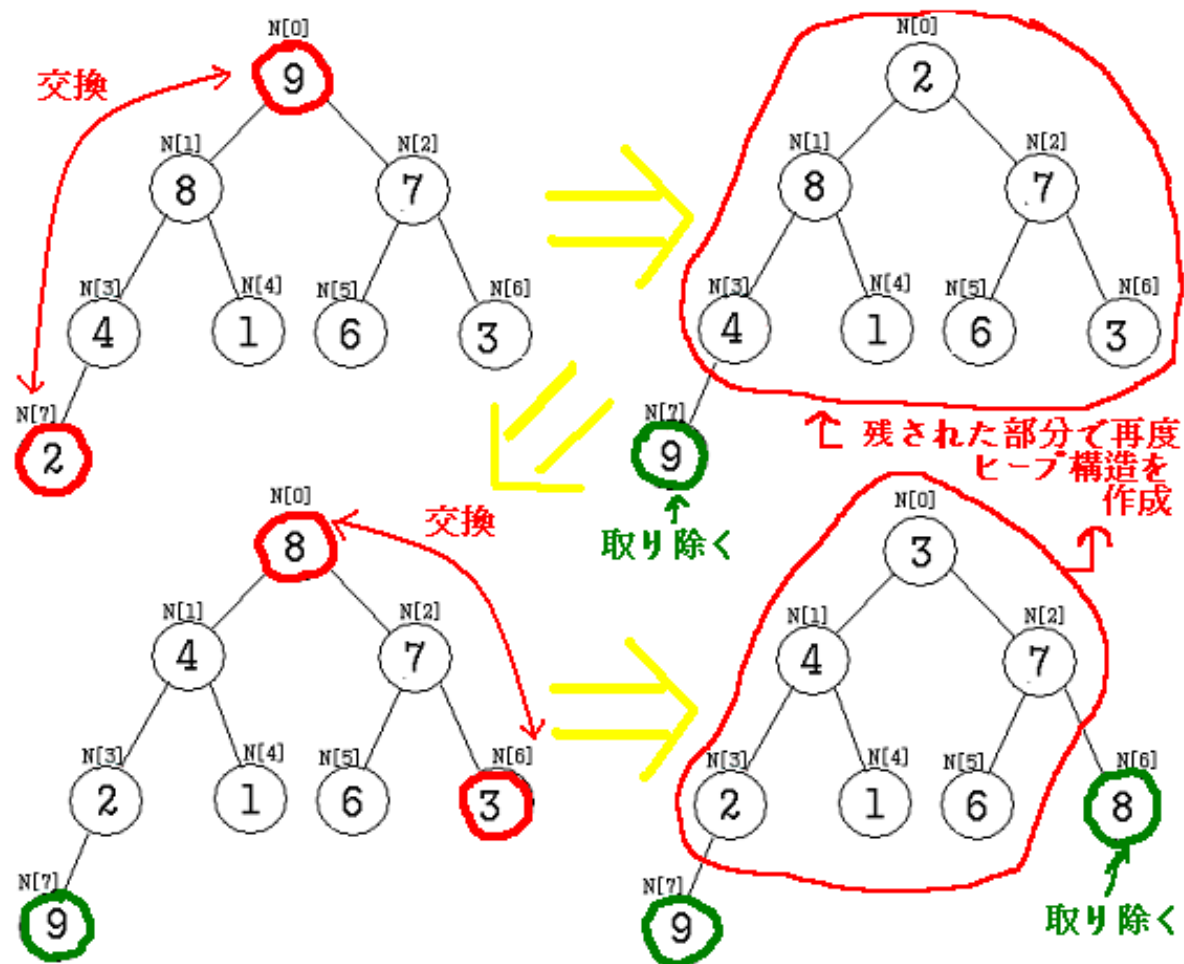
ヒープの根は最大値なので、それを最後尾にまわせばよい。



# まず木構造をヒープにする



# 最大値を最後尾に回す



# プログラム構造

- 再利用可能 リューザブル  
再び使用可能(電源を切らなければ終了できない)
- 再入可能 リエントラント  
同時に2つ利用可能 (それぞれ独立して管理する領域が必要)
- 再帰可能 リカーシブ  
自分自身を呼び起こす
- 再配置可能 リロケータブル  
プログラムをメモリ上で再配置することが可能なプログラム

覚えよう！ ➡再帰可能なら再入可能なら再利用可能

# 問1

- 解答ウ

- 右

$$X=1 \times M \times (M-1) \times (M-2) \times (M-3) \cdots 2 \times 1$$

- 左

$$X=1 \times 2 \times 3 \times \cdots (\text{終わりは} M \text{にしなければならぬ})$$

# 問2

- 解答ア
- 2分探索・・・整列されたものは有利
- 線形探索・・・単純に一番上から見るので使用頻度が高いものには有利
- ハッシュ関数・・・コードから一意に求められる

# 問3

- 解答ア

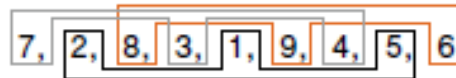
減分増分列はの場合データ数は9なので

$$9/3=3$$

$3/3 = 1$ (最後は挿入ソートのため**必ず最後は1で終わる**)

〔手順1回目〕

- (1) [データ数(9)]  $\div 3 \rightarrow 3$ で,  $H = 3$
- (2) データ列を3ごとの部分列とし, それぞれの部分列を, 挿入法を用いて整列します。次のようなかたちになります。



それぞれの部分列ごとに整列すると, 次のようになります。 |

## 問4

- 解答ア
- $F(n+1)+0.2 \times F(n)=2 \times F(n)$   
↓
- $F(n+1)/F(n)+0.2=2$   
↓
- $F(n+1)/F(n)=1.8$

## 問5

- 解答 **イ**

呼び出し元ごとに確保された記憶領域に局所  
変数が割り当てられる