アルゴリズム論
（第１３回）

岩手県立大学
Iwate Prefectural University

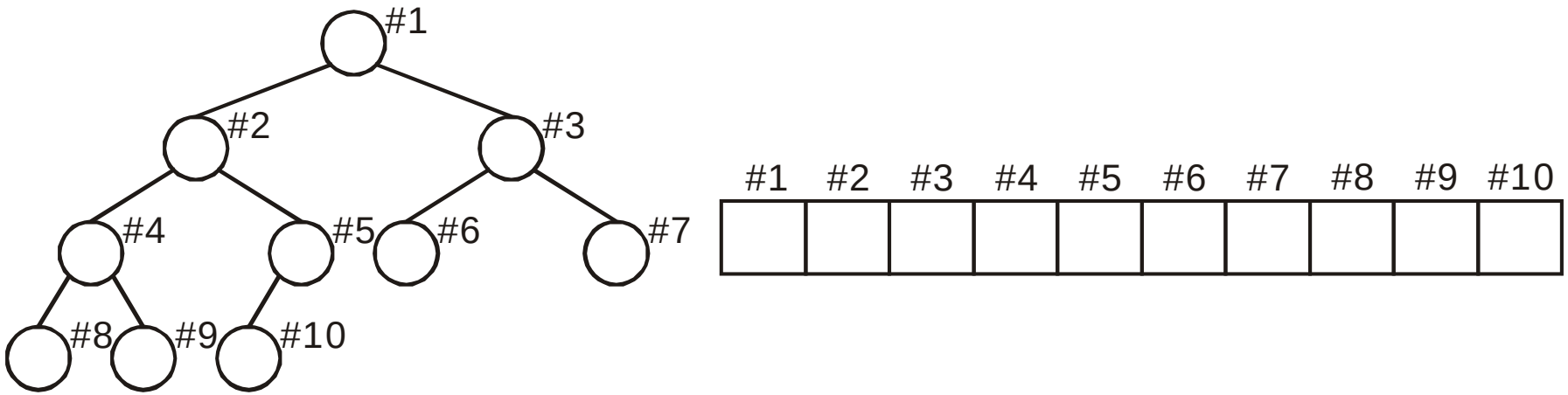
佐々木研（情報システム構築学講座）
講師　山田敬三
k-yamada@iwate-pu.ac.jp

ヒープ

岩手県立大学
Iwate Prefectural University

完全二分木の配列による実現

- 配列と簡単な計算式で実現可能
- あるノード **n** に対して
  - 親ノードは **n/2**
  - 子ノードは **2n** および **2n+1**



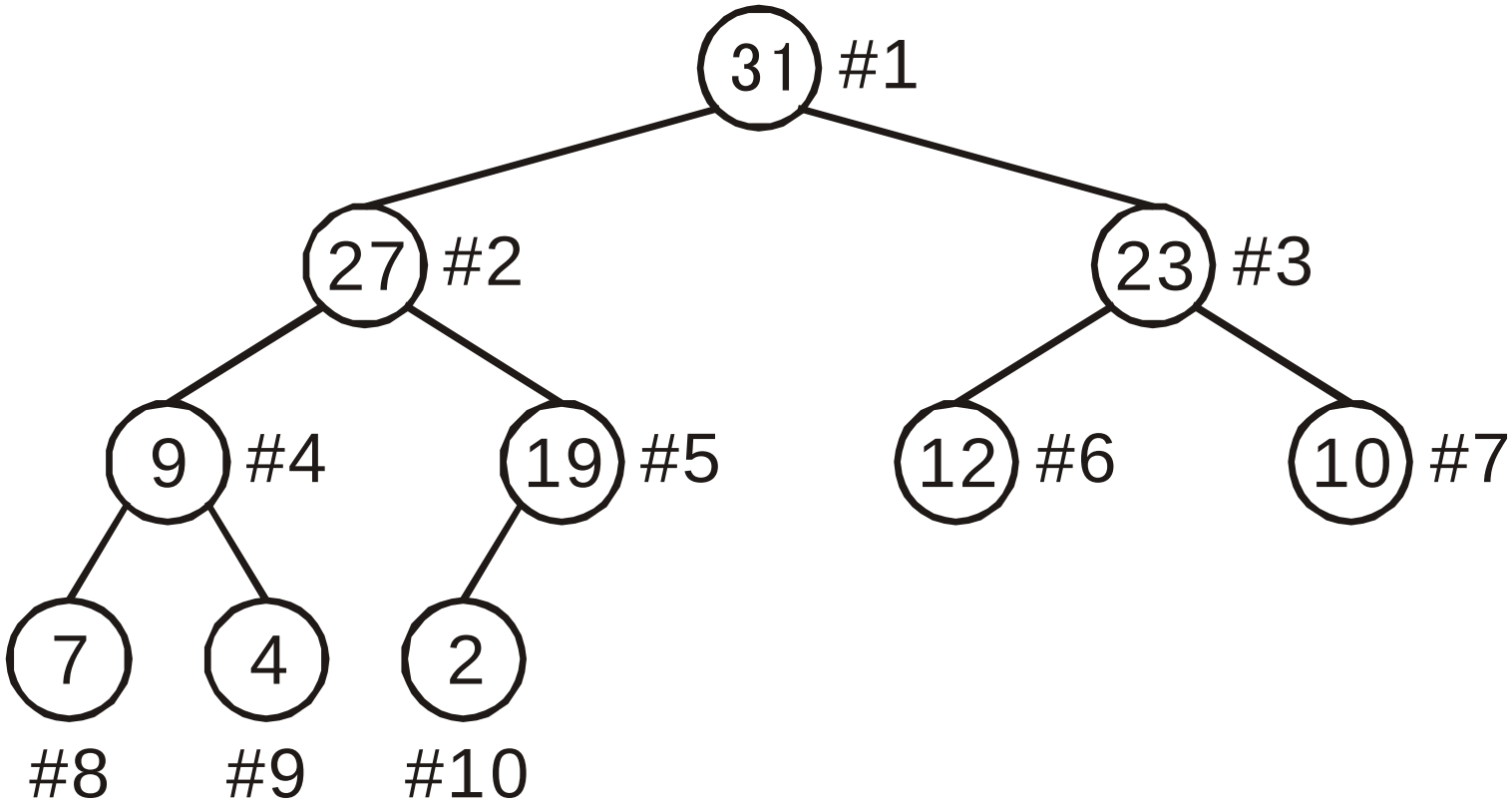
ヒープ(Heap)

- 完全二分木で，以下の条件を満たすもの

  （配列で実現可能）

- ヒープ条件
  - 任意のノードの値は，そのノードのどちらの子の値よりも大きいか等しい．

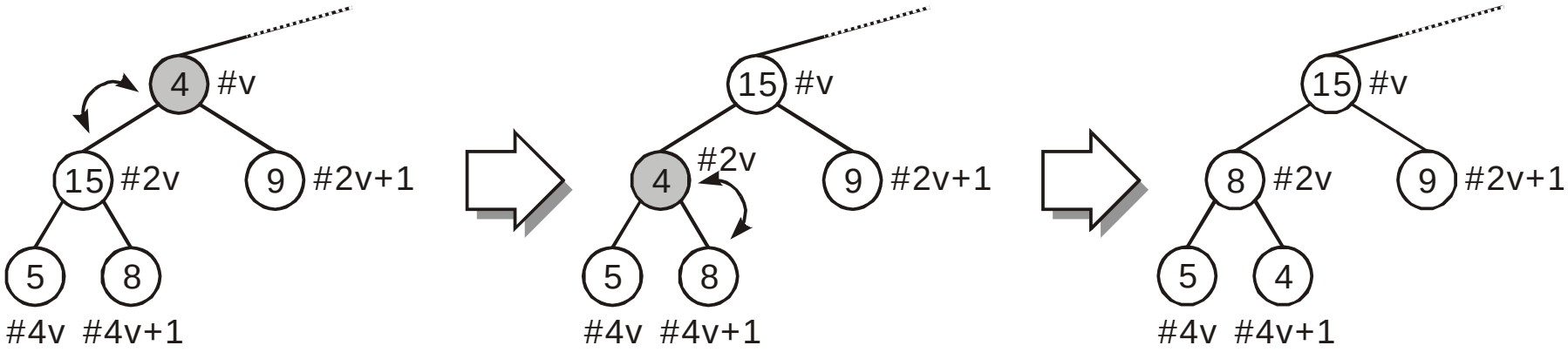
ヒープ(Heap)



下降修復 (Downheap)

- ヒープ条件を満たしていない
  完全二分木をヒープ化する.
  1. ノードvの値がそのどちらかの子の値より小さければ
  2. 値が大きい方の子wの値とvの値を入れ替える
  3. wに対して下降修復を繰り返す.

下降修復 (Downheap)

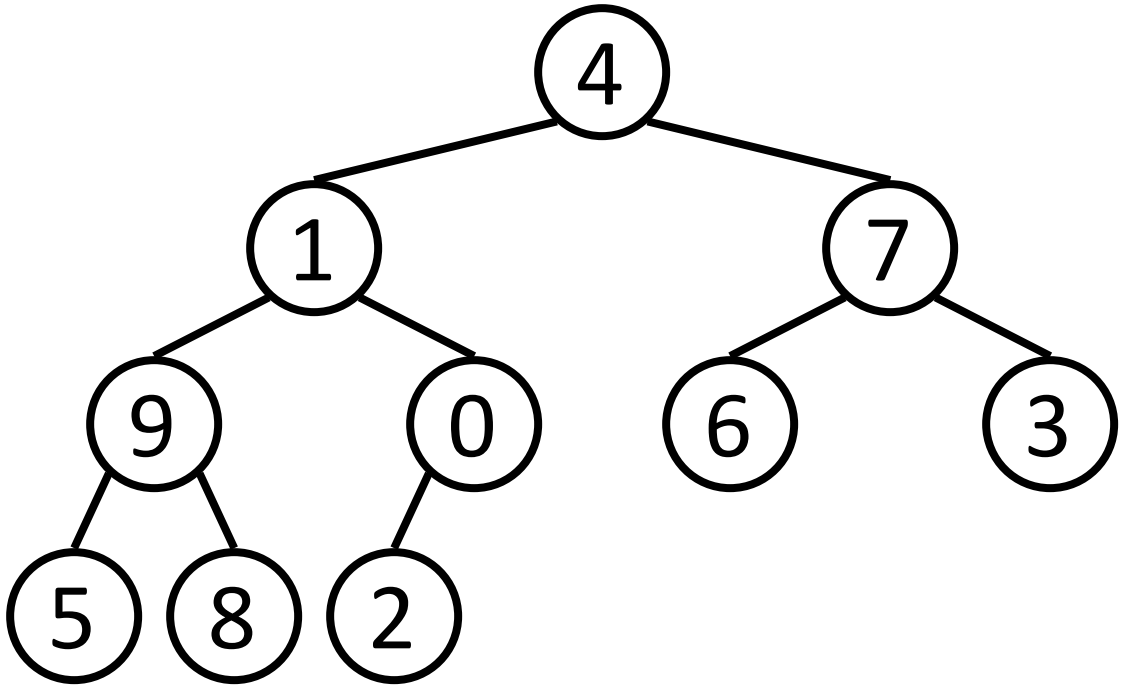


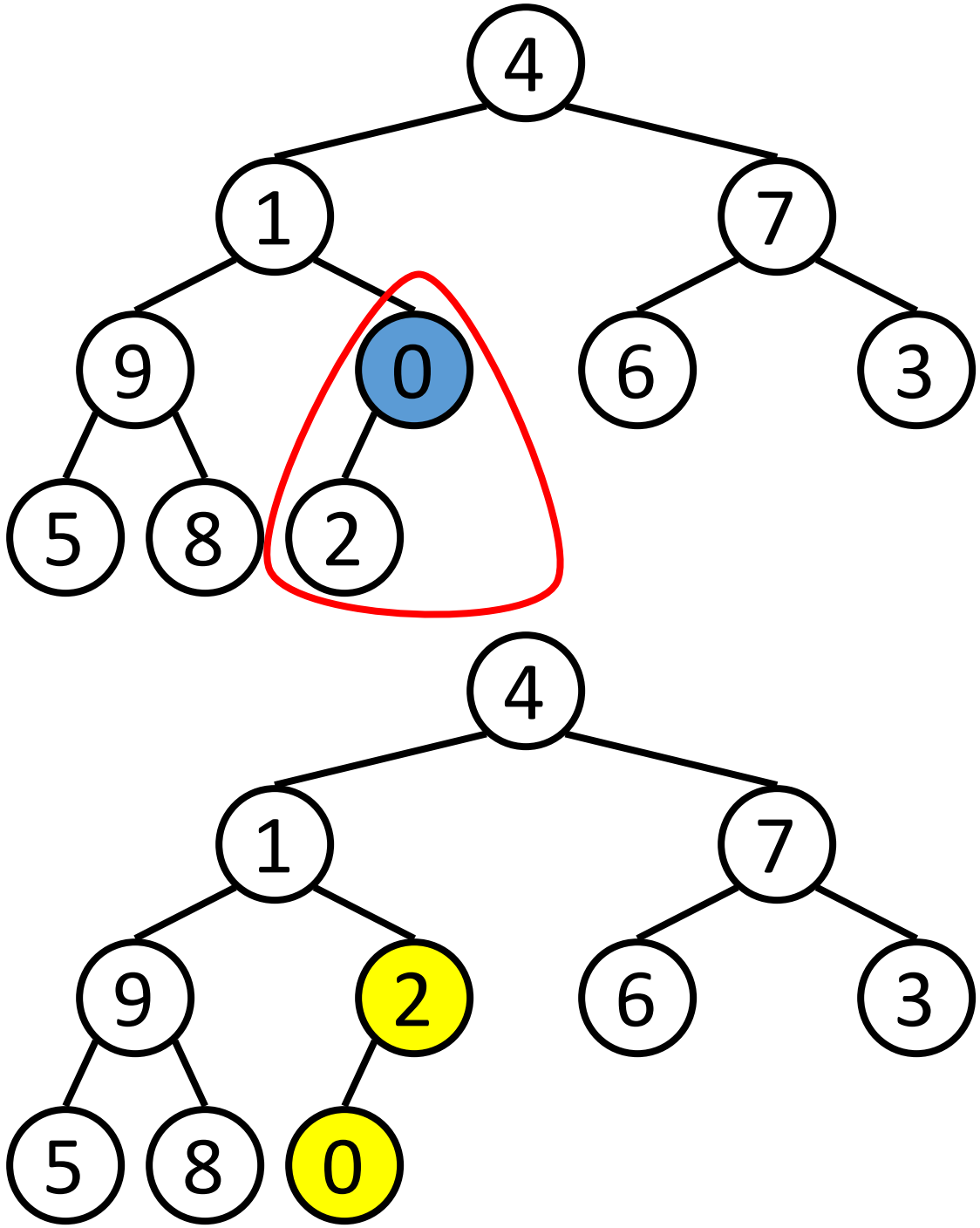
下降修復 (Downheap)

```
if(v>(N/2)) return;
if(右の子がある&&左の子よりも右の子が大きい)
   右の子をwとする;
else
   左の子をwとする;
if(vよりもwが大きい) {
   vとwを交換;
   wを頂点とする部分木に対して下降修復
}
```

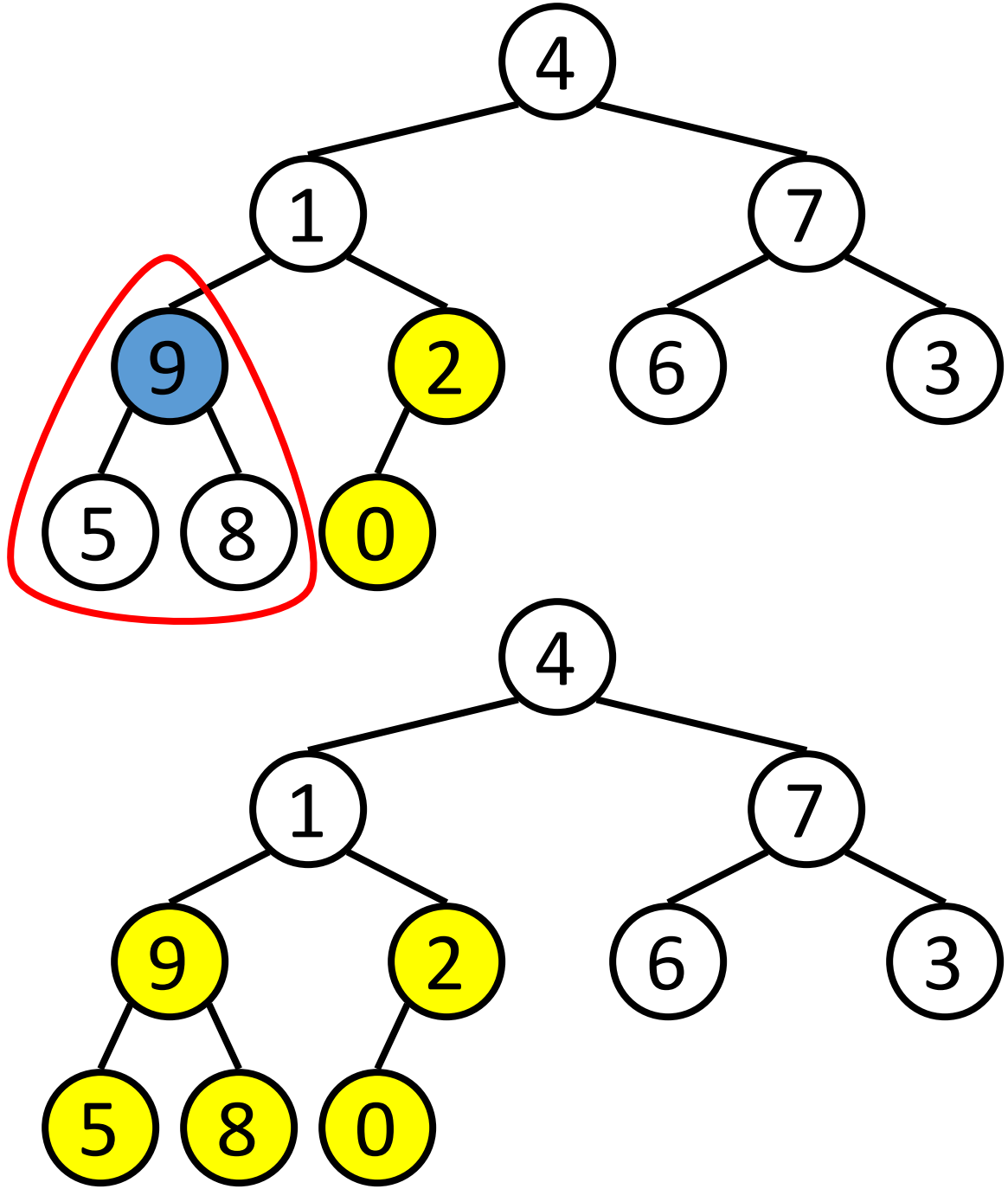
ヒープ化

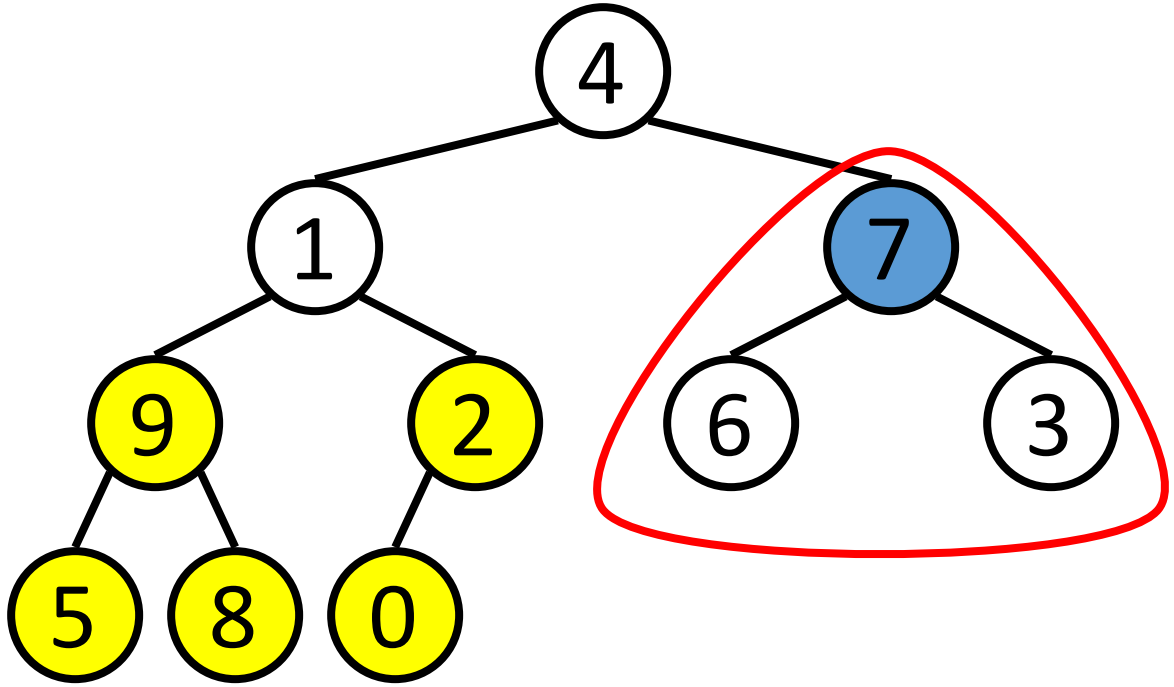
- 大きな木は，下方の部分木から繰り返す．

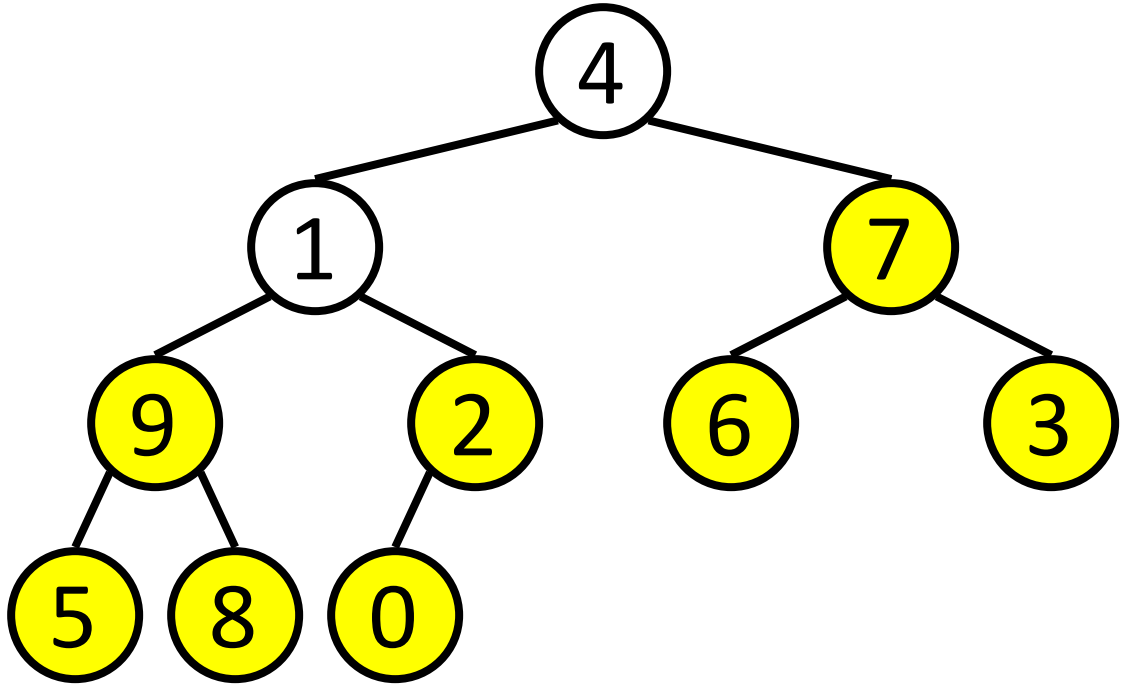


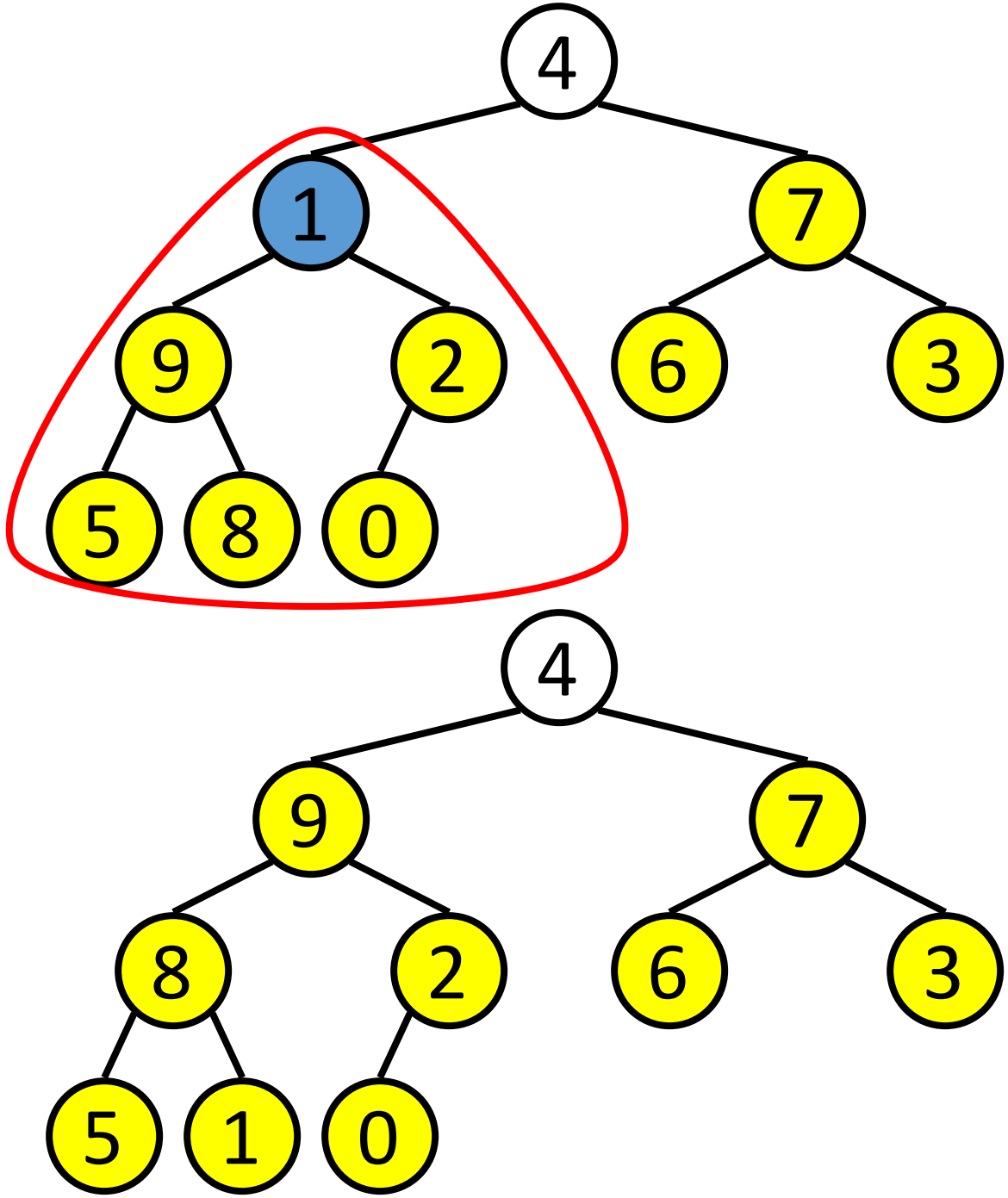


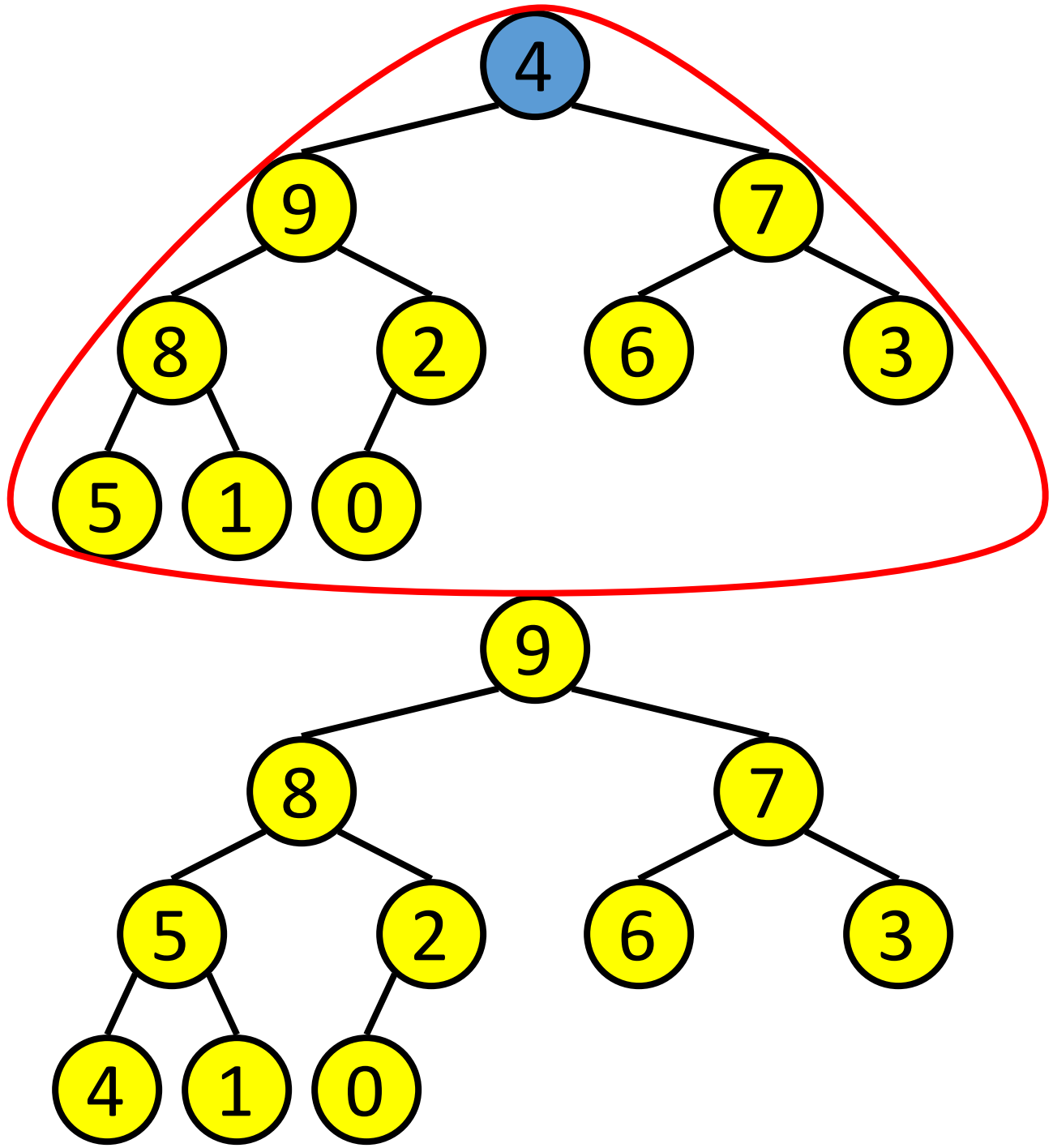
配列


4
1
7
9
2
6
3
5
8
0
4
1
7
9
2
6
3
5
8
0


4
1
7
9
2
6
3
5
8
0


4
1
7
9
2
6
3
5
8
0

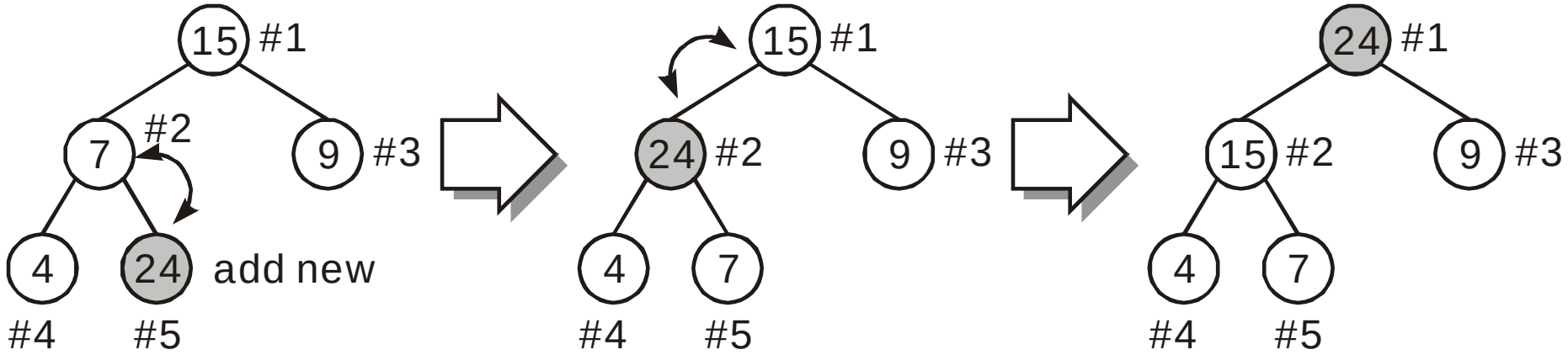

4
1
7
9
2
6
3
5
8
0
4
9
7
8
2
6
3
5
1
0


4
9
7
8
2
6
3
5
1
0
9
8
7
5
2
6
3
4
1
0

上昇修復 (Upheap)

- 新要素を追加した場合，上昇修復を行うことでヒープを復元
  1. ノード**v**の値がその親**u**の値より大きければ
  2. **u**と**v**の値を交換
  3. **u**に対して上昇修復を繰り返す.

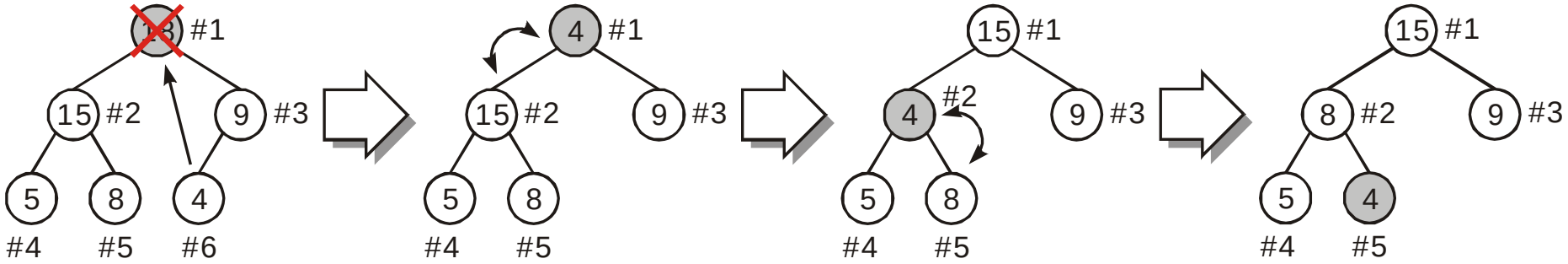
上昇修復 (Upheap)



ノードの削除

1. あるノードを削除した場合
2. **N-1**の要素を削除した部分に
   移動
   ※すなわち，配列の最後の要素
3. そこから下降修復を行うことに
   よりヒープを修復する.

ノードの削除

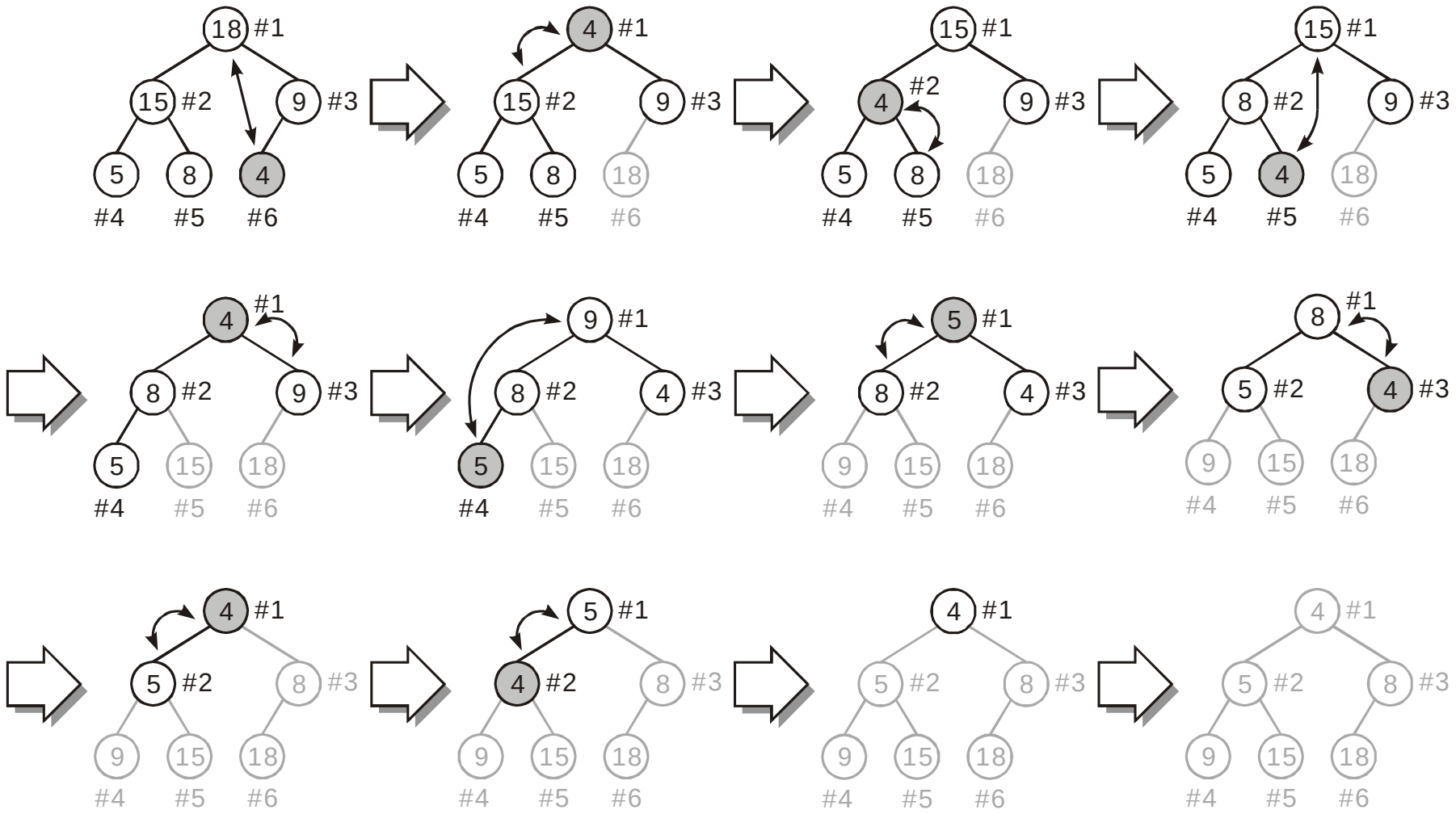


ヒープソート

1. ヒープ化
   - 配列を完全二分木とみなし
     ヒープ化する.
2. ルートの値を取り出す.
   - 最後のノードと交換
     ※ルートの値は最大
3. 残りのノードをヒープ化する.
   - ノードの削除と同様の処理

- 2., 3. を繰り返すことによりソートが可能

ヒープソート


18 #1
15 #2
9 #3
5 8 4
#4 #5 #6
4 #1
15 #2
9 #3
5 8 18
#4 #5 #6
15 #1
4 #2
9 #3
5 8 18
#4 #5 #6
15 #1
8 #2
9 #3
5 4 18
#4 #5 #6
4 #1
8 #2
9 #3
5 15 18
#4 #5 #6
9 #1
8 #2
4 #3
5 15 18
#4 #5 #6
5 #1
8 #2
4 #3
9 15 18
#4 #5 #6
8 #1
5 #2
4 #3
9 15 18
#4 #5 #6
4 #1
5 #2
8 #3
9 15 18
#4 #5 #6
5 #1
4 #2
8 #3
9 15 18
#4 #5 #6
4 #1
5 #2
8 #3
9 15 18
#4 #5 #6
4 #1
5 #2
8 #3
9 15 18
#4 #5 #6

ヒープソートの計算量

- 交換回数
  - ヒープの木，$\log_2 N$段
  - N個の要素に対して操作
  - 最悪$\log_2 N$段分の交換
  - $O(N\log_2 N)$
- 比較回数
  - 交換回数と同じ
  - 左右の比較を行うから2倍
  - $O(2N\log_2 N) \rightarrow O(N\log_2 N)$

マージソート

岩手県立大学
Iwate Prefectural University

内部ソートと外部ソート

- 内部ソート
  - 主記憶を使用
- 外部ソート
  - ファイルを直接操作してソートを行う.
- 一般に，主記憶より補助記憶の方が容量が大きい

ファイルのmerge

- P.64～P.65のプログラム解説
- ファイルaaa, ファイルbbbから同時に１つずつ数値を読み込み、小さい値をファイルcccに書き込む
- aaa,bbbのどちらかが終わるまで続ける
- どちらかが終わったら、数値が残っているファイルの中身をcccへ書き込む

自然マージ（merge）ソート

- 連（run)：順序つけられた部分

  f: 29 32 34 21 19 50 10 43 33 49 100 60

下線で示した連をfa, fbのファイルに分配

  fa: 29 32 34 19 50 33 49 100

  fb: 21 10 43 60

これから、fa, fbをマージしてfに書き込む

  f: 21 29 32 34 10 19 43 50 60 33 49 100

これを繰り返す

自然マージ（merge）ソート

f: 21 29 32 34　10 19 43 50 60　33 49 100

⇒　fa: 21 29 32 34 33 49 100

　　fb: 10 19 43 50 60

f :10 19 21 29 32 34 43 50 60 33 49 100

⇒　fa: 10 19 21 29 32 34 43 50 60

　　fb: 33 49 100

f: 10 19 21 29 32 33 34 43 49 50 60 100

ソート終了

ファイルのmerge

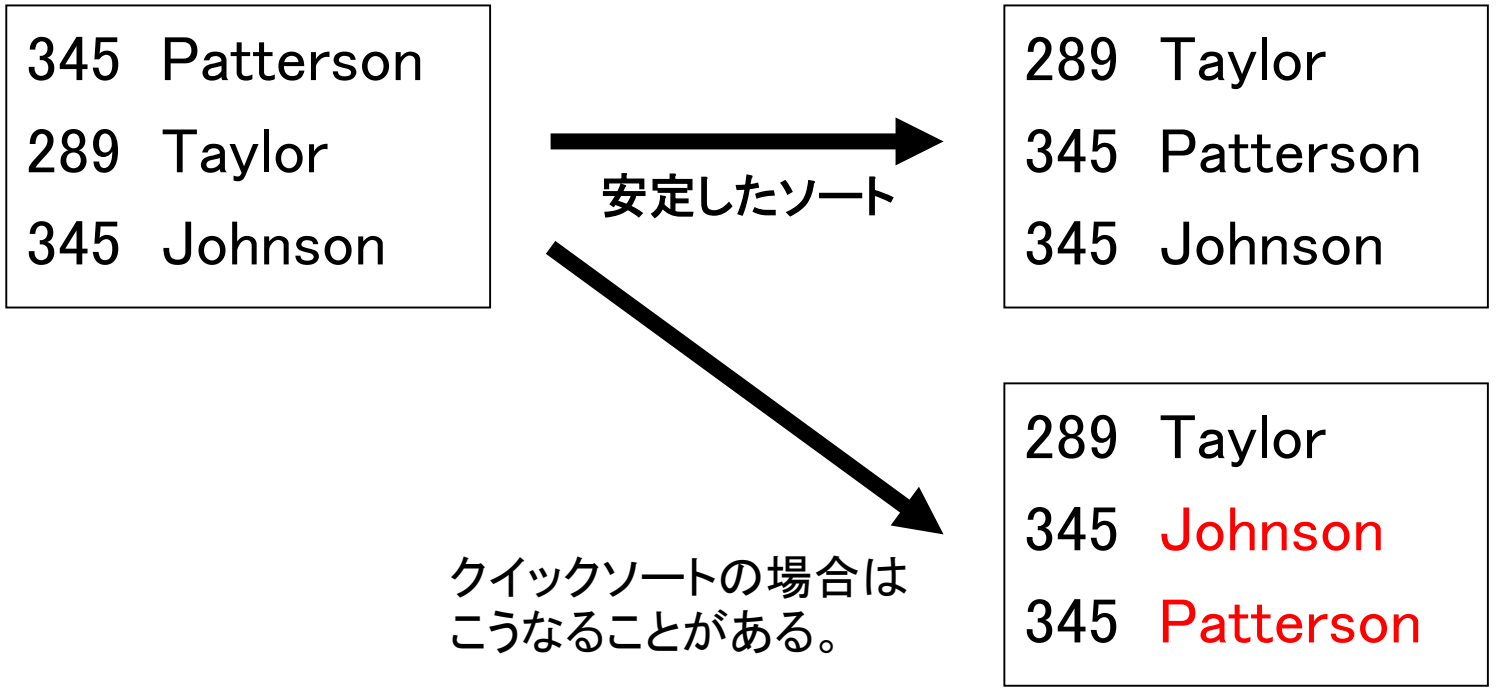
- P.66～P.69のプログラム解説
- main: メインプログラム
  - 初期設定
  - ファイルからデータ入力と、結果出力
- nmsort: distributeとmergeを繰り返してソート
- distribute: 連ごとにファイルへ分散
  - copyarunを呼び出してfa, fbへ連をコピー
- merge: 2つのファイルfa, fbを連ごとにマージ
- copyarun: fから連を抽出する

ランダムアクセスを使ったソート

- ランダムアクセスを行う関数
  - fseek(fp, offset, code)
    - ファイルの読み取り（書き込み）開始場所を指定する。
    - fseek(fp, 0L, SEEK_END)でファイルの長さが確認できる。
  - ftell(fp)
    - 現在のファイルの読み取り（書き込み）開始位置を確認する。
- ランダムアクセスを使用してクイックソートを行う(P.71～73)

安定な(stable)ソート

同じキーを持つレコードの順番がソート後も保持されるソートのこと。



ソートの評価

- クイックソートも自然マージソートも O(n log n)
- 自然マージソートはクイックソートに比べて多くのファイル空間を使用する。
- ハードウェアの条件によっては，クイックソートが遅い場合もある.
  - ハードディスク: ○Qsort　×Msort
  - 仮想ディスク:　×Qsort　○Msort

ソートの評価

1. 自然マージソートは一時的なファイルを使用するが、クイックソートは使用しない
2. クイックソートはランダムアクセスに基づいているため、ランダムアクセスが出来ないハードウェアや開発言語では実現不可
3. すでに，ほぼ昇順になっている場合は，自然マージソートの方が速い.
4. 自然マージソートは安定な(stable)ソートである.（クイックソートは違う）
5. クイックソートのキーの比較回数は、自然マージソートに比べてはるかに低い。