アルゴリズムと情報処理
（第3回）

慶應義塾大学生命情報学科
榊原康文

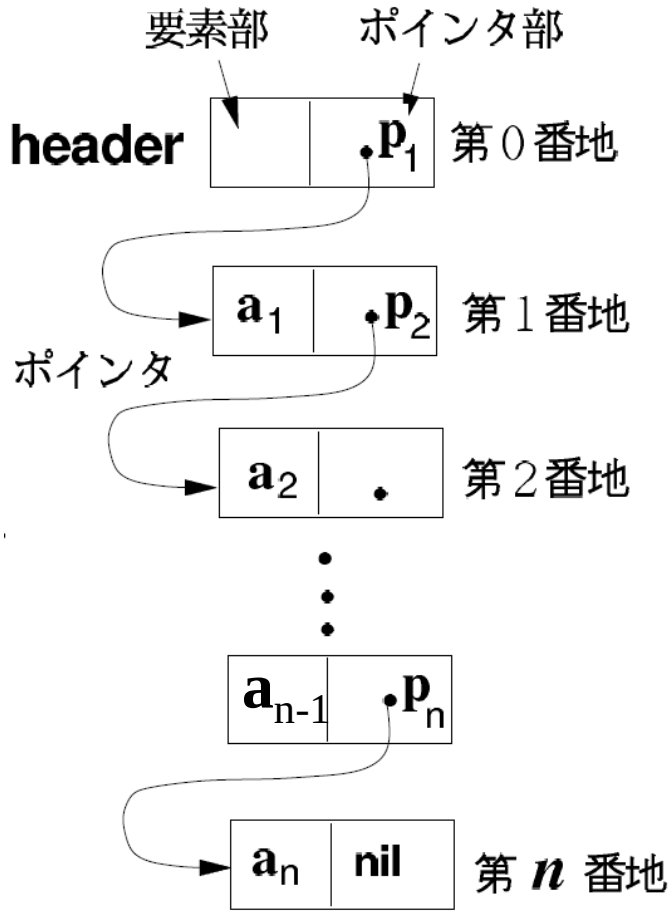
データ構造：連結リスト

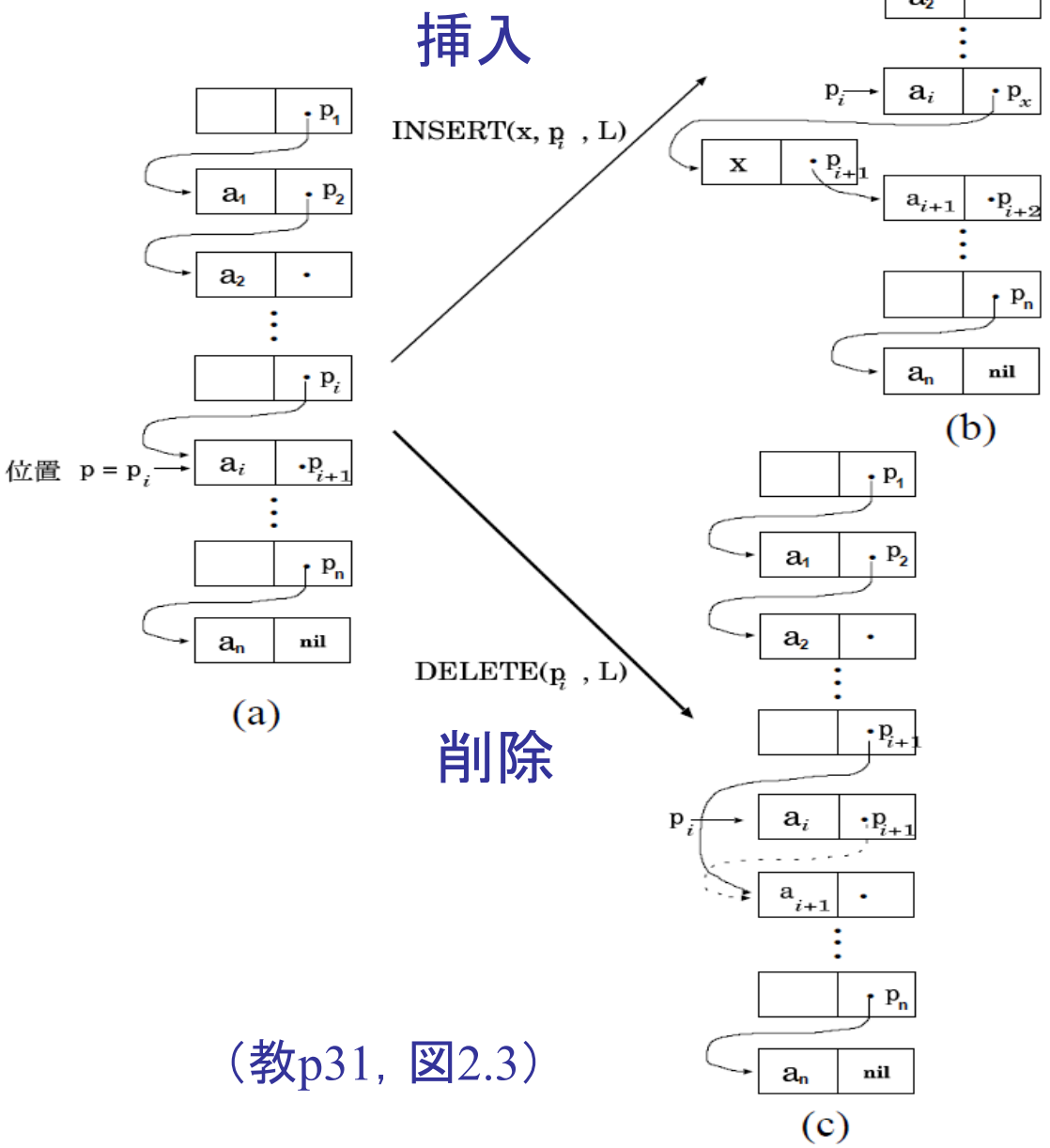
- 連結リストとは：（教科書p-27）
  - セル（ノード）と呼ばれる要素を0個以上，一列に並べたもの
  - セルは要素部とポインタ部という2つの部分に別れている
  - セル間のつながりをポインタによって保持する

    $[\_|p_1] \rightarrow [a_1|p_2] \rightarrow [a_2|p_3] \rightarrow \ldots \rightarrow [a_{n-1}|p_n] \rightarrow [a_n|\mathrm{nil}]$
  - 最初のセルはヘッダーとよばれ，要素部は空でポインタ部には最初の要素a_1の納められているセルの番地がはいる
  - 最後の要素a_nを要素部にもつ$n+1$番目のセルのポインタ部にはnilという特殊な値がはいる
  - 連結リストの操作：

    INSERT，DELETE，LOCATE，NEXT，PREVIOUS，など
  - 連結リストの場合，要素の追加・削除は容易であるという利点がある．また途中の位置への挿入・削除は各々その直前，直後のセルを変更するだけで実現できる

データ構造：連結リスト



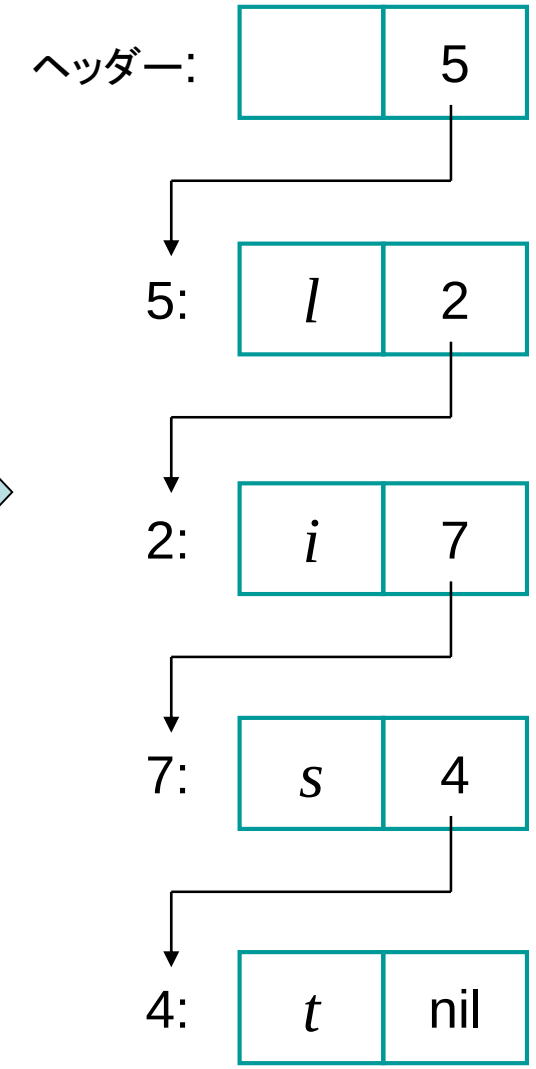


（教p31，図2.3）

データ構造：連結リスト

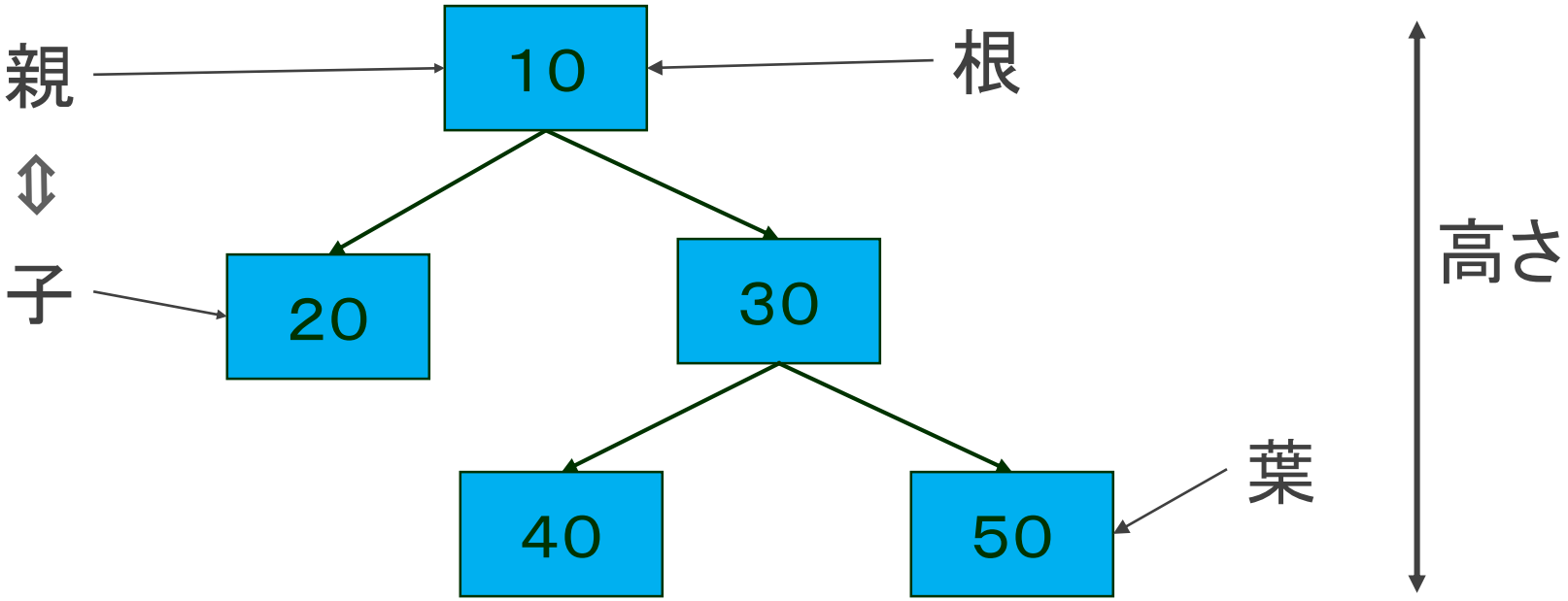
- 配列によるポインタの実現：
  - セルの要素部とポインタ部を2次元配列の要素として表す

	要素部	ポインタ部
ヘッダーセル		5
1		
2	*i*	7
3		
4	*t*	
5	*l*	2
6		
7	*s*	4
8		
9		



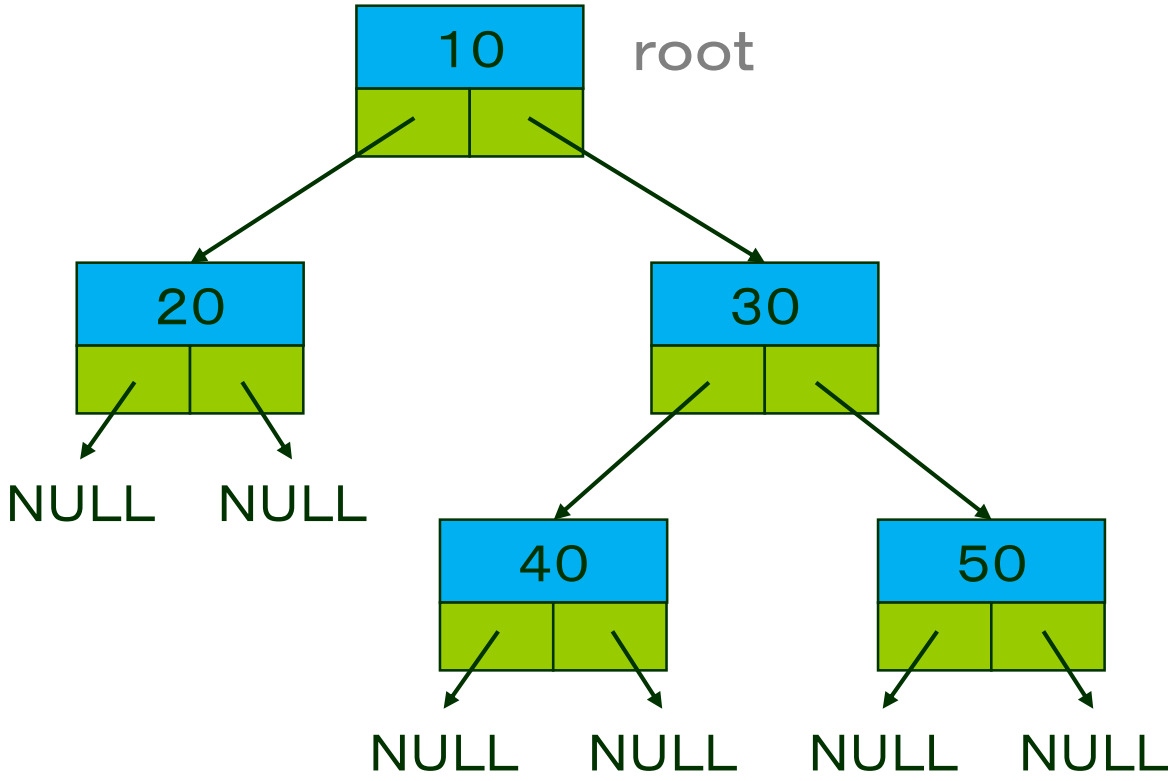
データ構造：木

- 「木」とは：
  - 木は，一つのデータ構造 （グラフ構造の特殊な場合，教科書p-38）
  - 根付き木を考える
  - 木では，リストと異なり枝分かれがある
  - 木では，項目を入れる「ノード（節点）」の中に子供の節点を指示する（複数の）ポインタを入れて表現する



リンクによる二分木

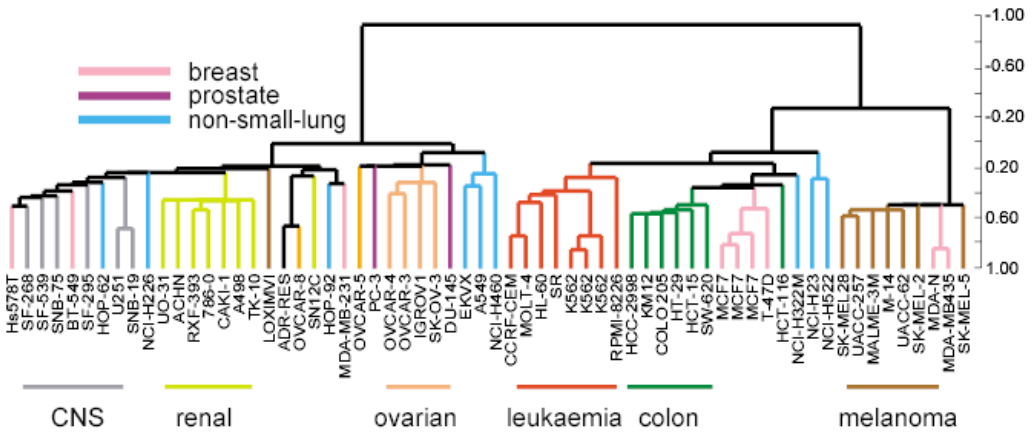
- 木の「根」のノードを設定し，これを「root」で表す
- 各ノードは，高々2つの子を持つ．したがって，各ノードは2つのポインタを持つ　⇒　「2分木」
- 木の中で最後に位置するノードである「葉」のノードのポインタにはNULLを入れる

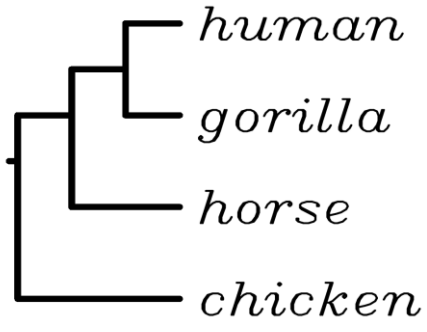


木による表現例

- 生命科学における例
  - 進化系統樹
  - 糖鎖構造
  - RNAの2次構造
  - 細胞系譜
  - 階層的クラスタリング

（階層的クラスタリング）





（進化系統樹）

Table 1: Common monosaccharide names and their symbols.

Monosaccharide name	Sym.
Glucose	▲
Galactose	●
Mannose	○
N-acetyl neuraminic / sialic acid	◆
N-acetylglucosamine	■
N-acetylgalactosamine	□
Fucose	△
Xylose	▽
Glucuronic acid	◆
Iduronic acid	◆

（糖鎖構造）

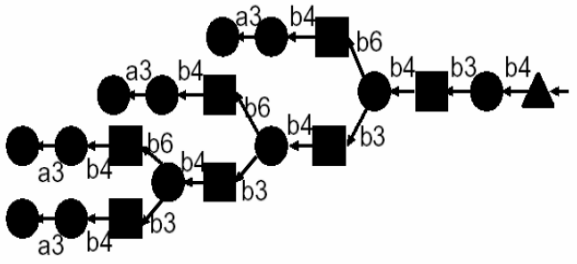
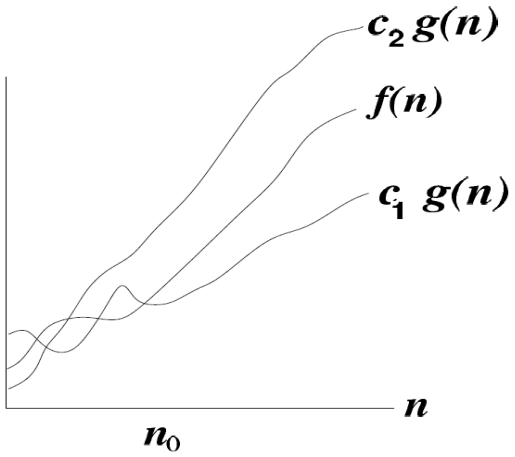


Figure 2: A glycan chain: Eicosasaccharide (KEGG Glycan ID G07296).

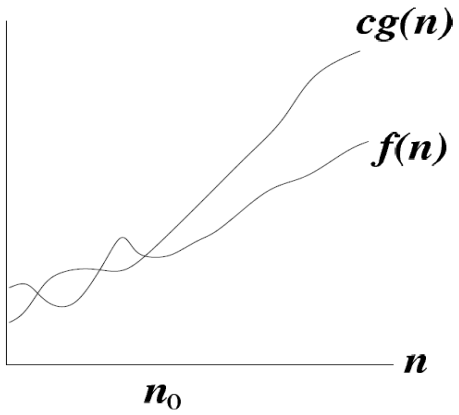
データ構造：ヒープ

- ヒープとは：
  - [形状] 木の高さを h とするとき，深さ (h–1) の部分までみると完全2分木（すなわち，すべての葉の深さが同じ2分木）になっていて，残りの部分の節点（つまり深さ h の葉）はすべて左から順に埋められている
  - [要素間の性質] 親節点の要素はその子節点の要素より順序において大きくない
  - [ヒープの高さ] 要素の個数 n に対して，ヒープの高さは Θ(log n)である **（高さ k のヒープのノード数 ≦ $2^{k+1}-1$ ）**

（記号Θの意味：）

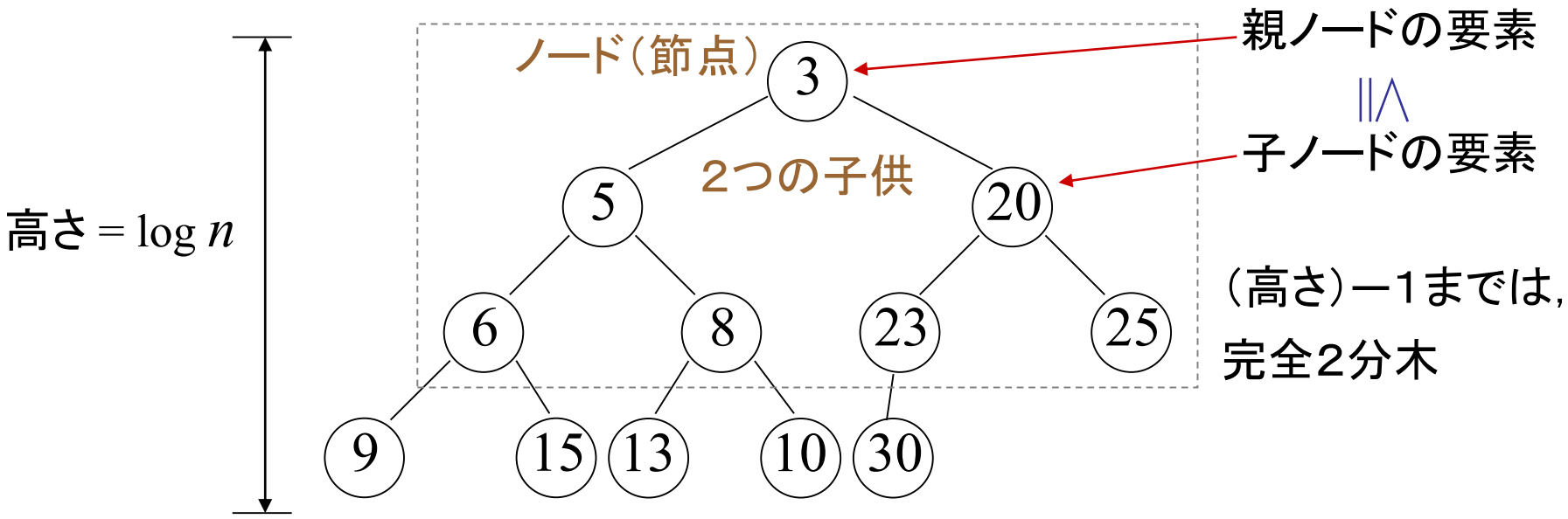


(c) $f(n)=\Theta(g(n))$

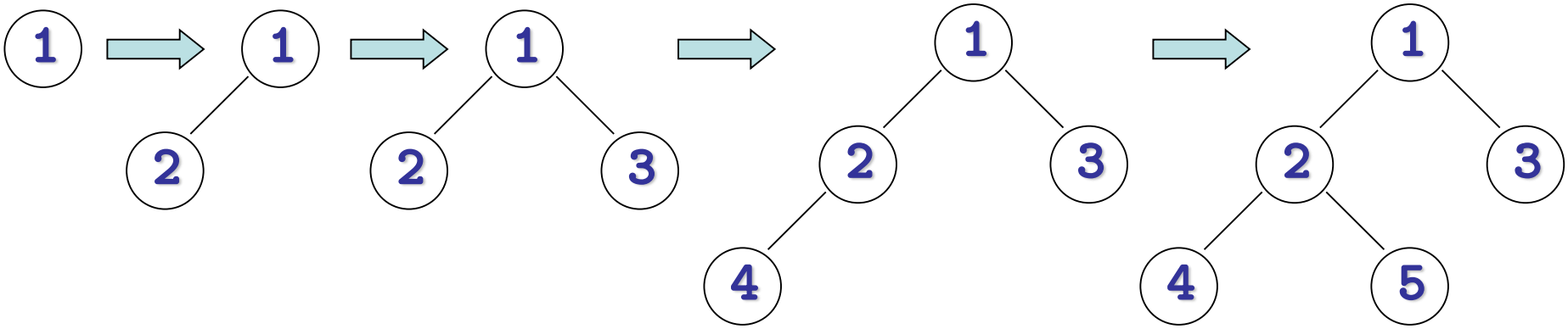


(a) $f(n)=O(g(n))$

データ構造：ヒープ

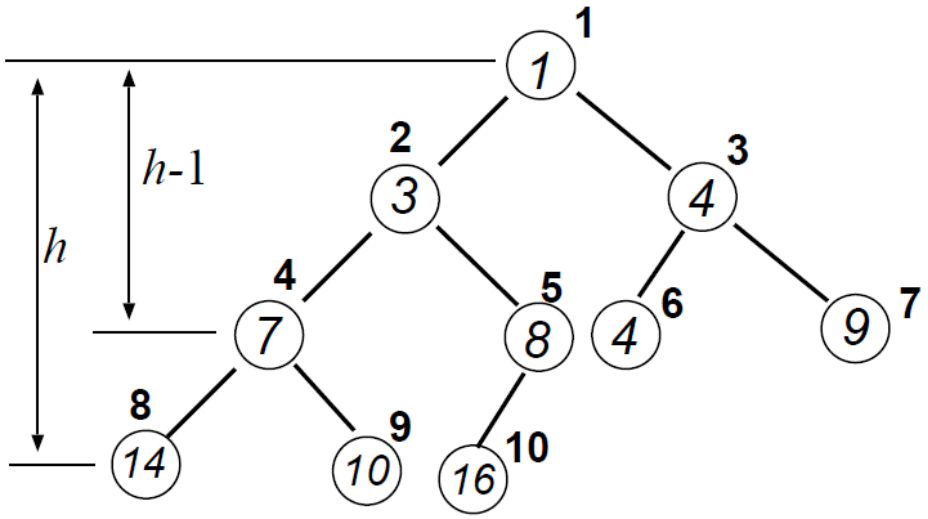


ノードの増加順番：

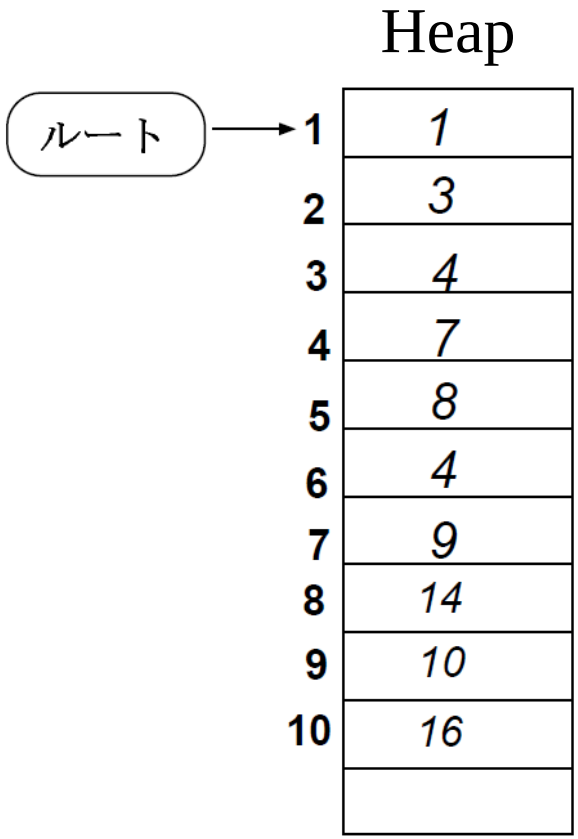


データ構造：配列によるヒープの実現

- 配列によるヒープの実現：
  - 根の要素は，**Heap[1]**へ入れる
  - 任意の i に対して，**Heap[i]の左の子はHeap[2i]へ，Heap[i]の右の子はHeap[2i+1]**へ入れる



(a) ヒープの例（ ◯ 内が要素）



(b) 配列による **(a)** の実現

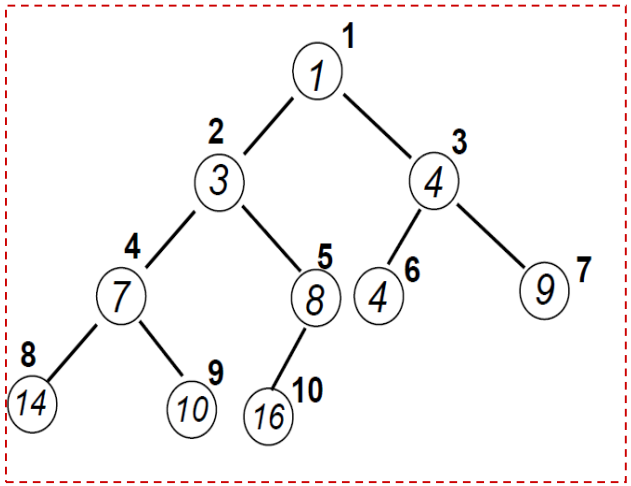
ヒープ上の操作①: DELETE_MIN

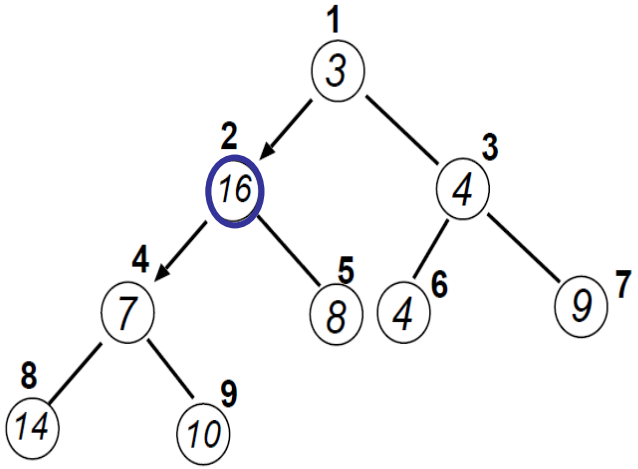
- DELETE_MIN:

  Heap上の順序に関して，最小の要素をHeapから取り出す

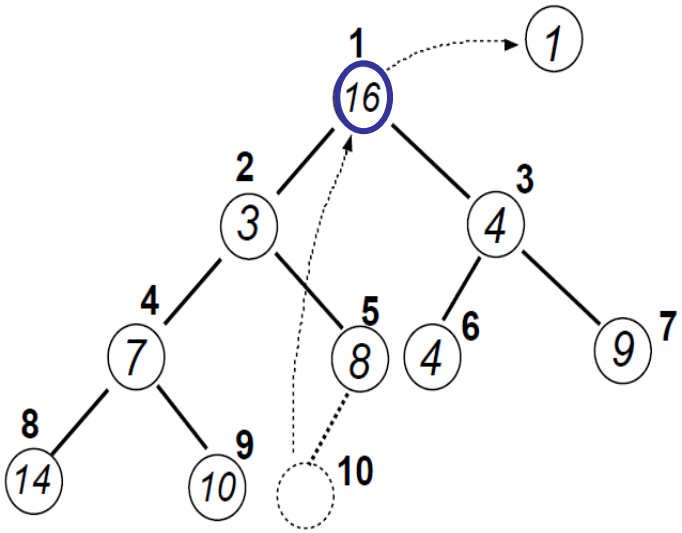
  – ヒープの性質から，根には最小の要素が貯えられているので，根に対応する配列上のHeap[1]を取り出せばよい
  – これだけでは他の残された木がヒープの性質をみたさないので以下の修正を行う:
    ① 木の（最も深い）葉のうちで最も右にあるもの（配列上では最後の要素）を新しい根として移動する
    ② 二つの子の小さい方と比較して親の方（の要素が順序の上で）が大きければその子と交換する
    ③ この操作を根から葉へ下向きに交換の必要がなくなるまで繰り返す
    ④ この操作はヒープの高さに比例する時間$\Theta(\log n)$ で実行できる

ヒープ上の操作①: DELETE_MIN

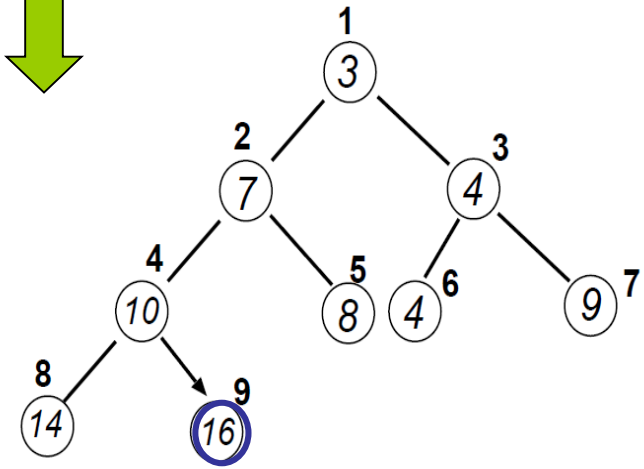




(b) 根と小さい方の子節点との交換



(a) 最後の葉の要素を根に移動



(c) 3回目の交換で終了

ヒープ上の操作②: INSERT

- INSERT:

  Heapに要素xを挿入する

  – 要素 x を挿入後は，Heapはヒープの性質を保つ必要がある:

    ① 今あるヒープの（最も深い葉のうちで）最右の葉のすぐ右隣の位置へ（要素 x ）おく

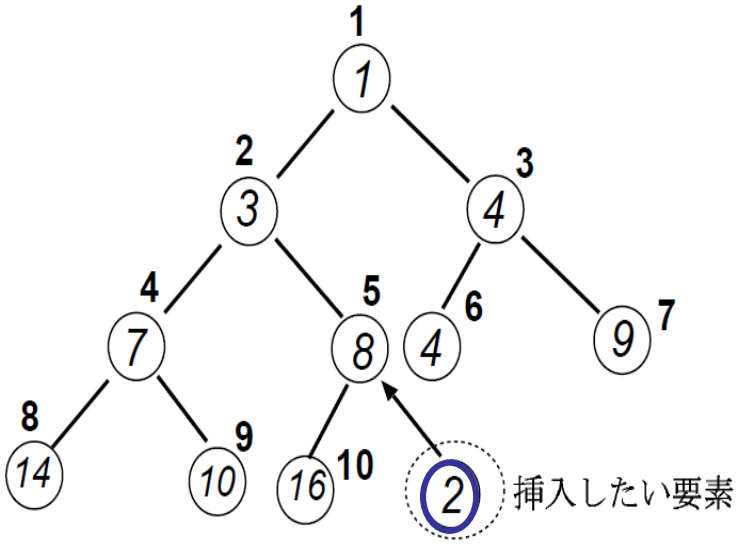
    ② もし（今ある）ヒープが完全2分木ならば，最左の葉の左の子として登録する

    ③ （挿入された要素 x の）その親と比較し，親の方が大きい限り次々と交換していく

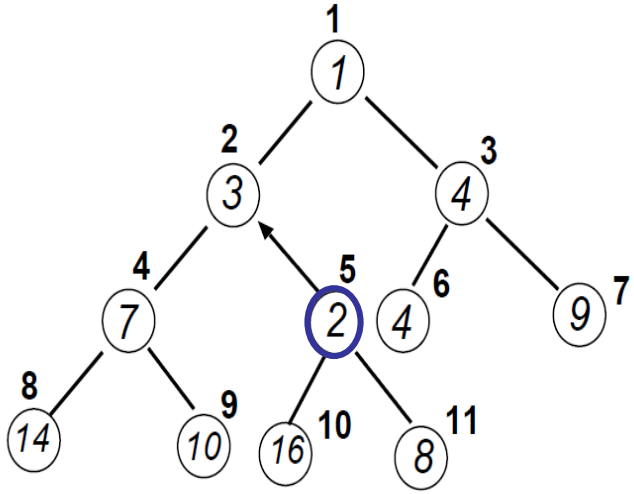
    ④ この操作を根から葉へ下向きに交換の必要がなくなるまで繰り返す

    ⑤ この操作はヒープの高さに比例する時間$\Theta(\log n)$ で実行できる

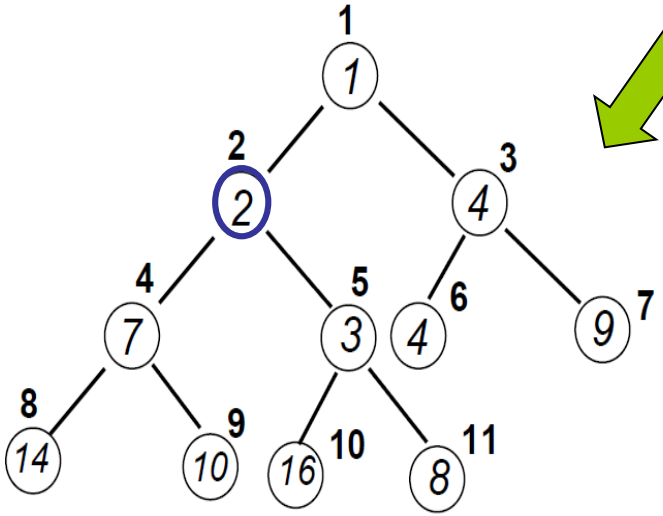
ヒープ上の操作②: INSERT



(a) 新しい要素の追加



(b) 親との交換



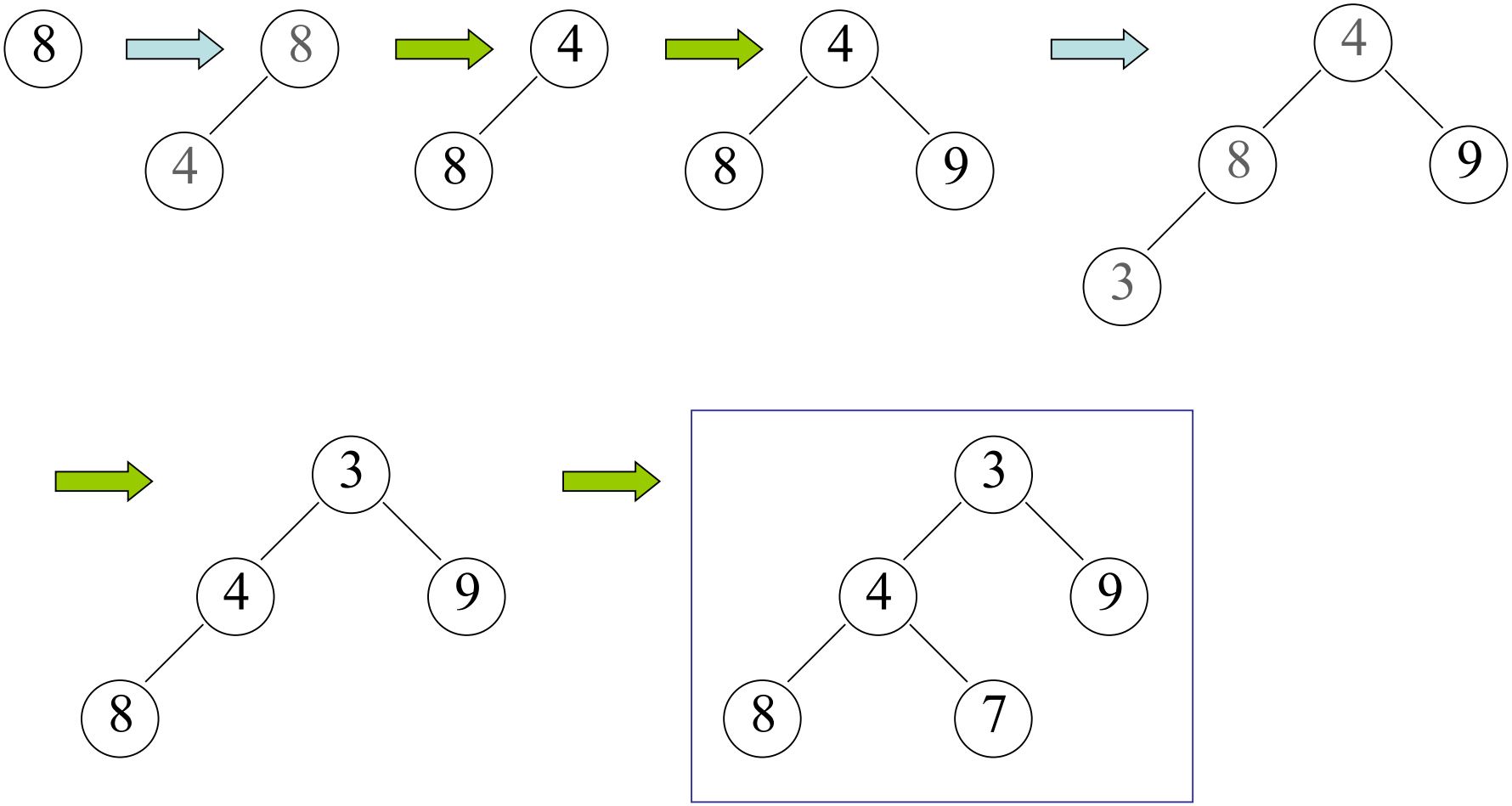
(c) 2回目の交換で終了

ヒープソート

- ヒープソート:
  1. 空のヒープを用意する
  2. 要素 A[1],...,A[n]の順にINSERTを用いて n 個の要素からなるヒープを作る
  3. DELETE_MINを n 回用いて要素を小さい順に取り出して整列させる
  4. 二つの操作に要する手間は，おのおの1回ごとに$O(\log n)$であったから，整列に要する時間は$O(n \log n)$となる

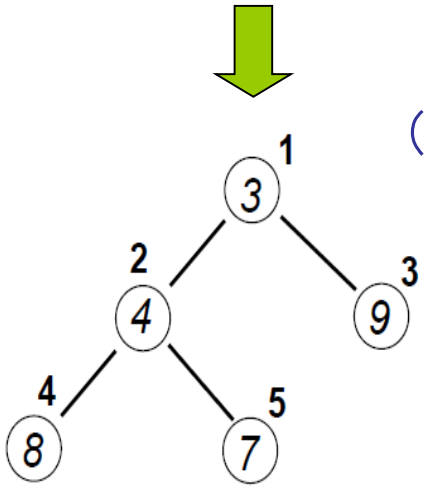
ヒープソート

入力列 (8, 4, 9, 3, 7)

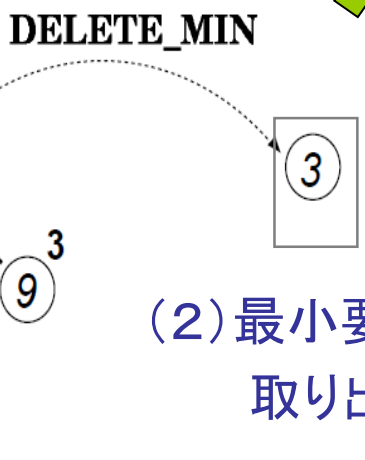


ヒープソート

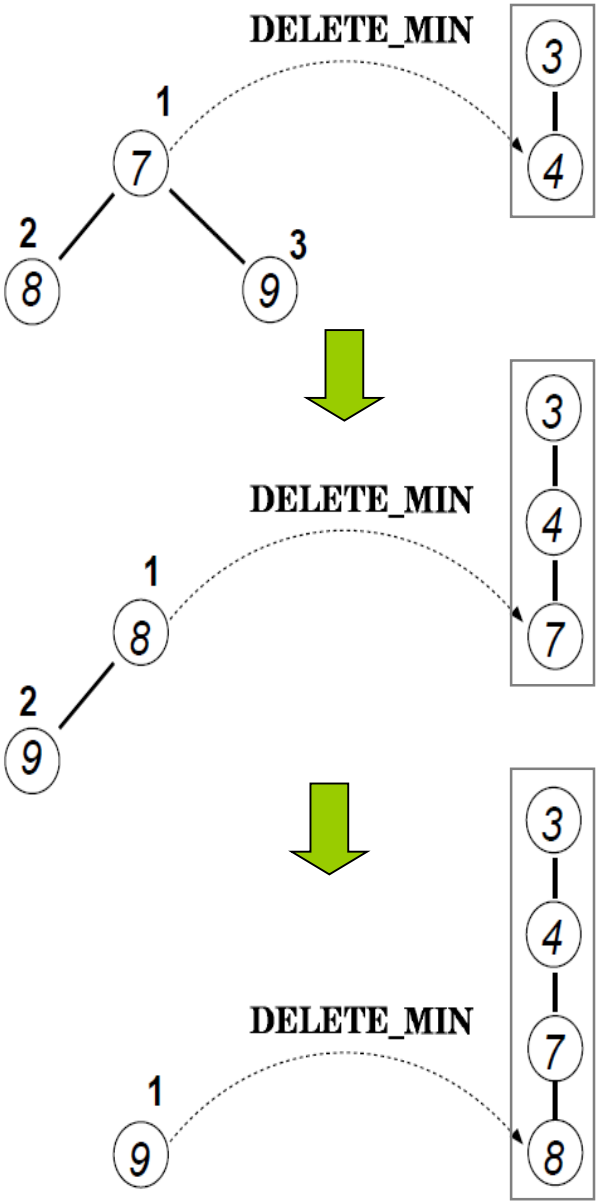
入力列 (8, 4, 9, 3, 7)



（1）入力要素列に対するヒープの作成



（2）最小要素の取り出し



ヒープソートの計算量と計算量のまとめ

- ヒープソートの計算量解析：

  $O(n \log n)$

- ソートアルゴリズムの計算量のまとめ：

ソート名＼効率の基準	最悪の場合（上界）	下界	平均的
バブルソート	$O(n^2)$	$\Omega(n \log n)$	—
バケットソート	$O(m+n)$	—	—
基数ソート	$O(km+kn)$	—	—
マージソート	$O(n \log n)$	$\Omega(n \log n)$	$\Theta(n \log n)$
ヒープソート	$O(n \log n)$	$\Omega(n \log n)$	$\Theta(n \log n)$
クイックソート	$O(n^2)$	$\Omega(n \log n)$	$\Theta(n \log n)$