

# アルゴリズムと データ構造

コンピュータサイエンスコース  
知能コンピューティングコース

## 第5回

スタック, キュー, ヒープ  
ヒープソート

塩浦昭義

情報科学研究科 准教授

[shioura@dais.is.tohoku.ac.jp](mailto:shioura@dais.is.tohoku.ac.jp)

<http://www.dais.is.tohoku.ac.jp/~shioura/teaching>

# データ構造とは？

- アルゴリズムの中で、与えられた問題に関連するデータ集合を管理するための道具
- 良いデータ構造とは？
  - データ管理に必要な時間が短い
  - シンプル
  - 必要な領域計算量(記憶容量, 領域量)が小さい

# 集合を管理する： 双方向リストの利用

- 双方向リストを利用して集合を表現する
  - 必要な領域計算量は**集合のサイズ**に等しい
  - 整数の追加, 削除は **$O(1)$ 時間**で可能

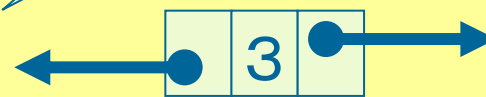
C言語では,  
構造体により  
実現可能

セルの構成:

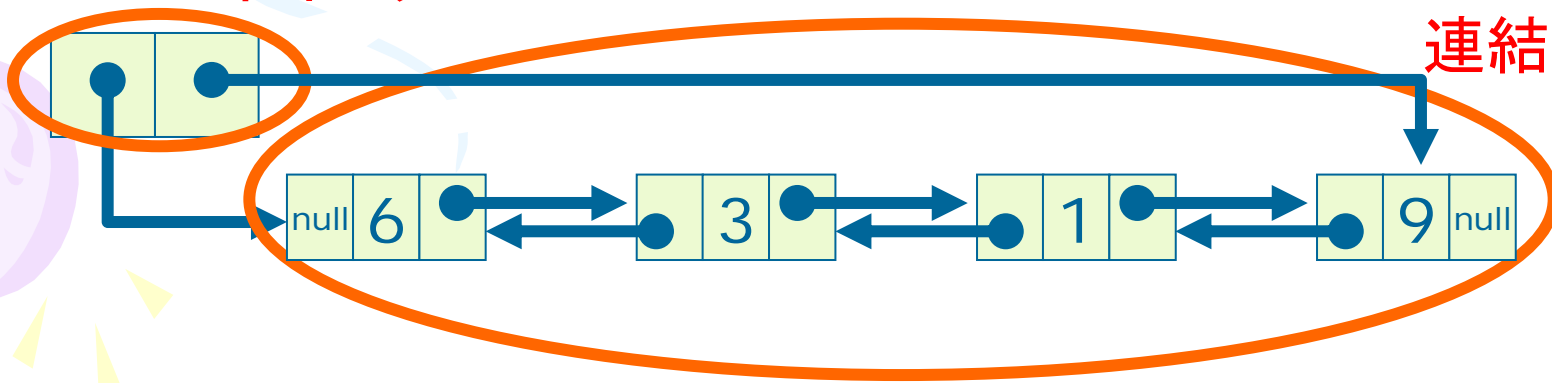
要素(整数)

前のセルへのポインタ

次のセルへのポインタ



連結リストの  
最初と最後の  
セルへのポインタ



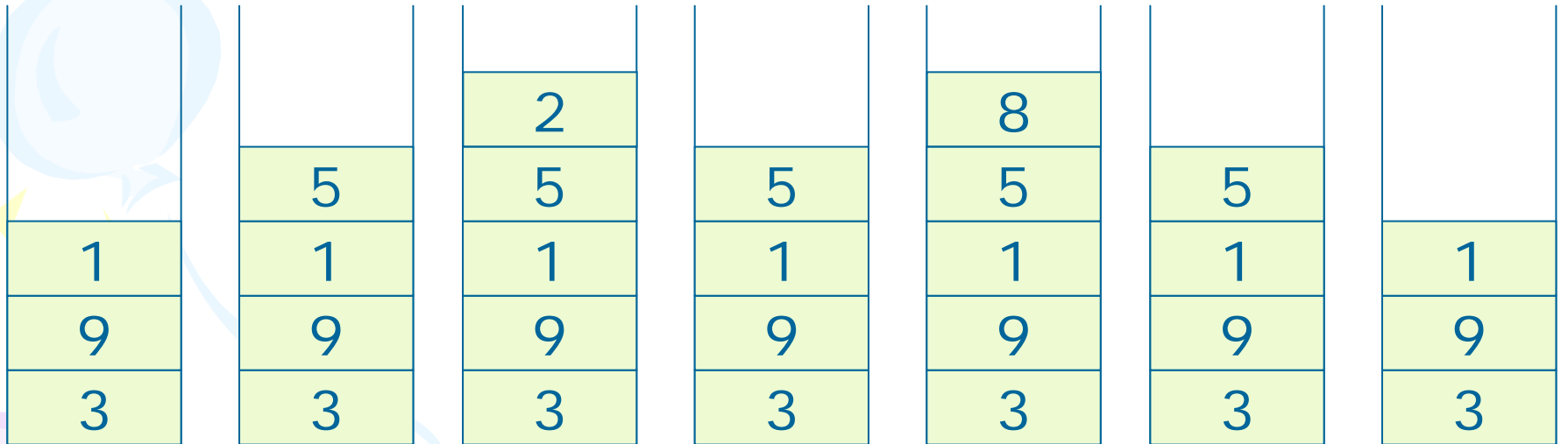
連結リストの  
本体

# 集合に対する特殊な演算と データ構造

- 最も新しい要素を常に削除する  
→ **スタック**
- 最も古い要素を常に削除する  
→ **待ち行列(キュー)**
- 最も小さい(大きい)要素を常に削除する  
→ 優先度付き待ち行列, **ヒープ**

# スタック

- 最も新しい要素を常に削除するときを使うデータ構造
- 次のような図で表現されることが多い



集合  
3, 9, 1

5を  
追加

2を  
追加

最新の  
要素2を  
削除

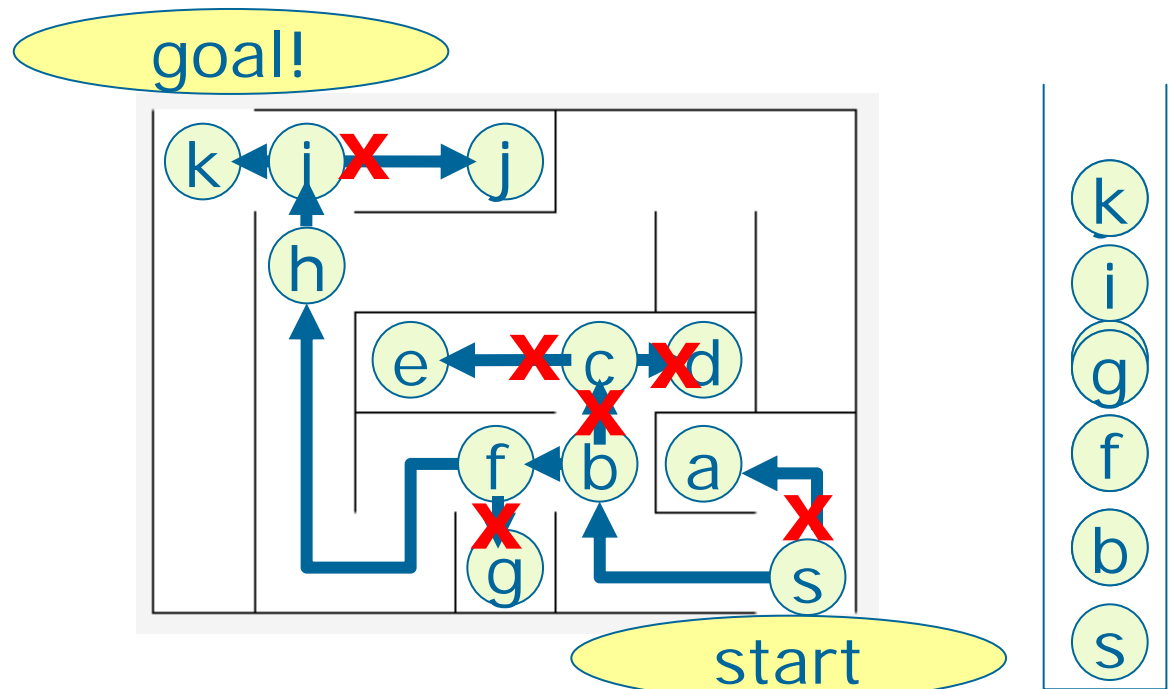
8を  
追加

最新の  
要素8を  
削除

最新の  
要素5を  
削除

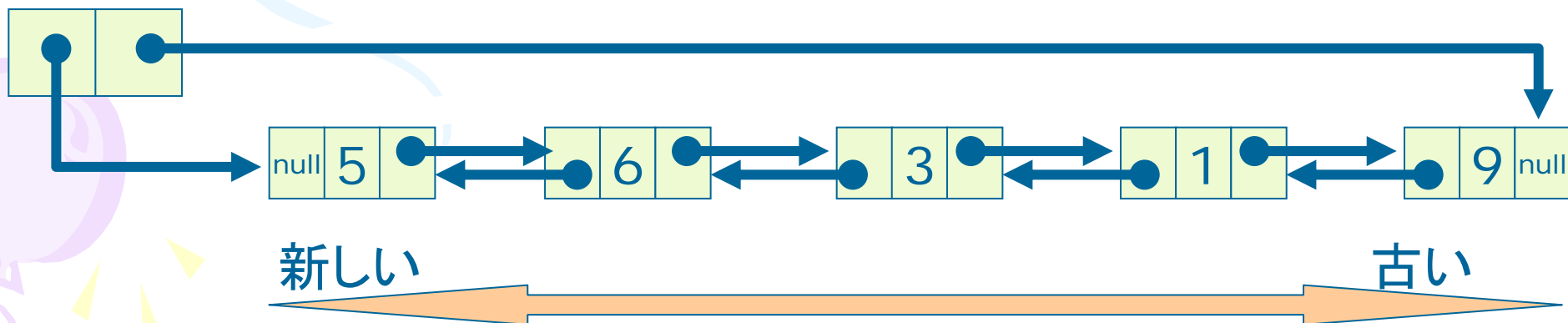
# スタックとバックトラック

- スタックは**バックトラック探索**を行なうときによく使われる
- 例: 迷路の探索: 進んだ道に戻ったりしながらゴールを目指す
  - 可能性がある限り先に進む
    - ゴールへ到達する可能性がなくなったら戻る
  - これまで進んできた道を覚えるために, スタックを使う



# スタックの実現方法

- 双方向リストを使えばよい
  - 要素を追加するとき、**リストの先頭に追加**
    - リストの要素は、新しいものから古いものへと並ぶ
  - 最も新しい要素を削除したい
    - **リストの先頭の要素を削除**すればよい
- ∴ **追加も削除も $O(1)$ 時間**でできる

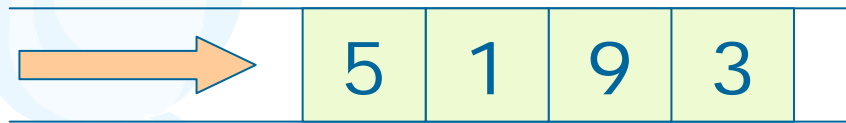


# 待ち行列(キュー)

- 最も古い要素を常に削除するときを使うデータ構造



集合 3, 9, 1



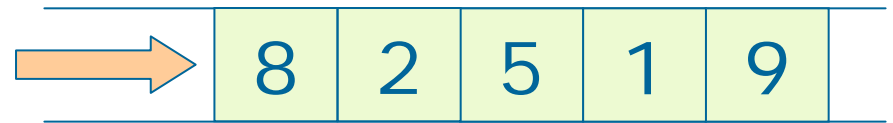
5を追加



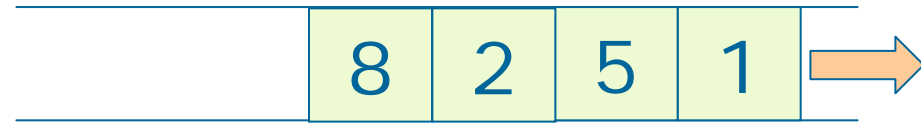
2を追加



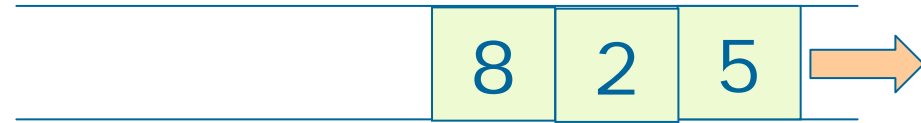
最も古い要素3を削除



8を追加



最も古い要素9を削除



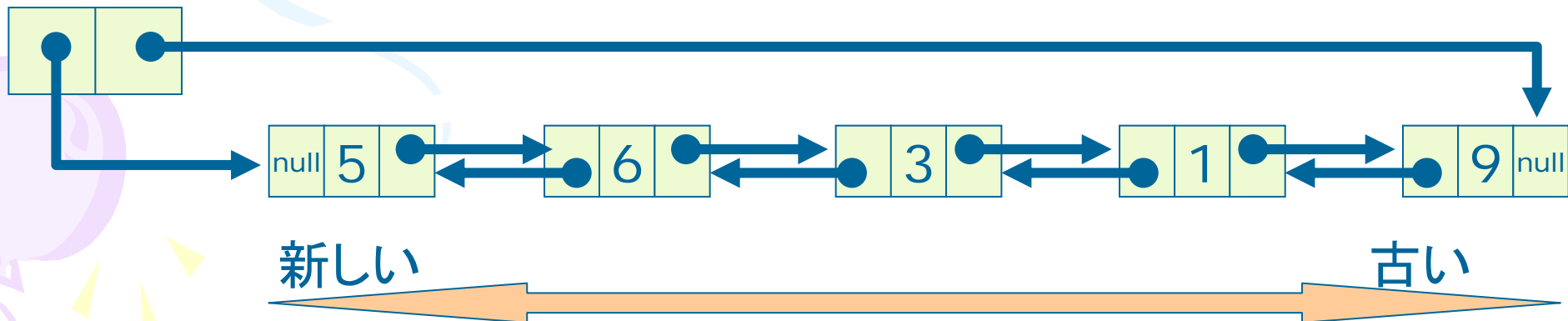
最も古い要素1を削除

ジョブの処理の際に利用



# 待ち行列の実現方法

- 双方向リストを使えばよい
  - 要素を追加するとき、**リストの先頭に追加**
    - リストの要素は、新しいものから古いものへと並ぶ
  - 最も古い要素を削除したい
    - **リストの最後尾の要素を削除**すればよい
- ∴ **追加も削除も $O(1)$ 時間**でできる



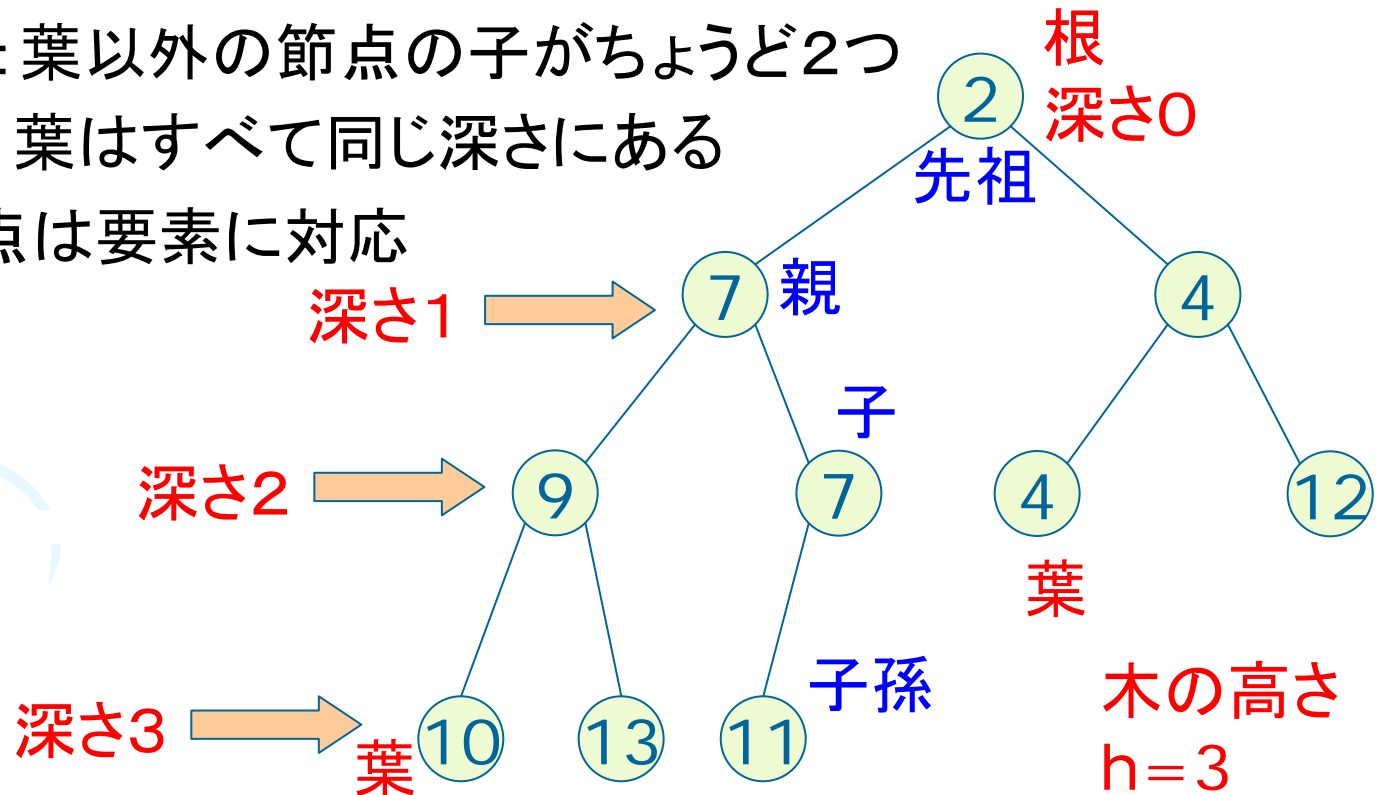
# ヒープ

- 最も小さい(大きい)要素を常に削除するときを使うデータ構造
  - 新しい要素の追加, 最小要素の削除は  $O(\log n)$  時間

- 2分木により表現される

- 2分木: 葉以外の各節点が高々2つの子をもつ根付き木
- 完全2分木: 葉以外の節点の子がちょうど2つ  
葉はすべて同じ深さにある

- 2分木の各節点は要素に対応



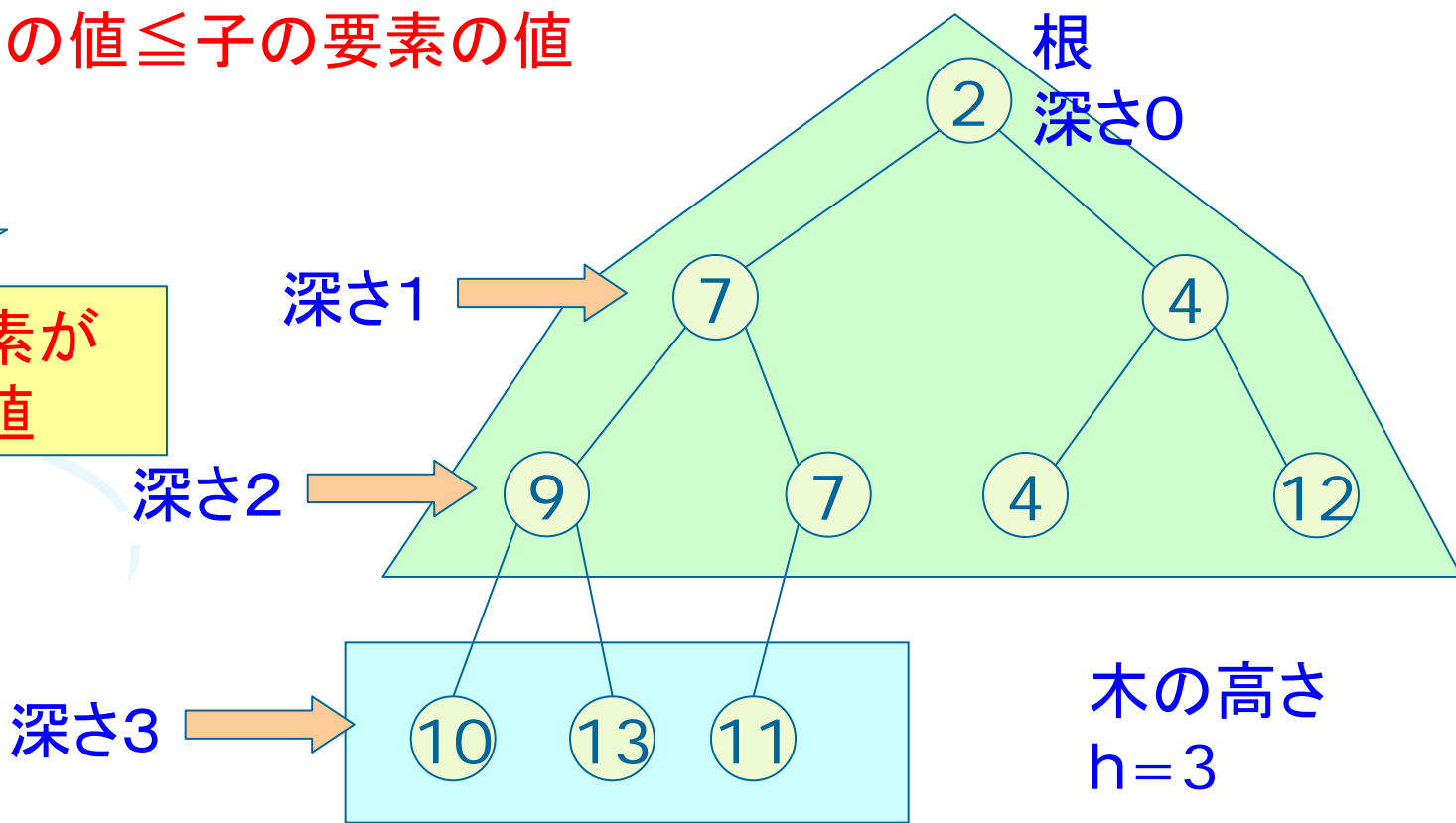
# ヒープの条件

木の高さ  
 $= \lfloor \log_2 n \rfloor$

- 次の条件を満たす2分木をヒープと呼ぶ
- (1) 木の高さが  $h \rightarrow$  深さ  $h-1$  までは完全2分木  
深さ  $h$  では, 左側に葉が詰められている

(2) 親の要素の値  $\leq$  子の要素の値

根の要素が  
最小値



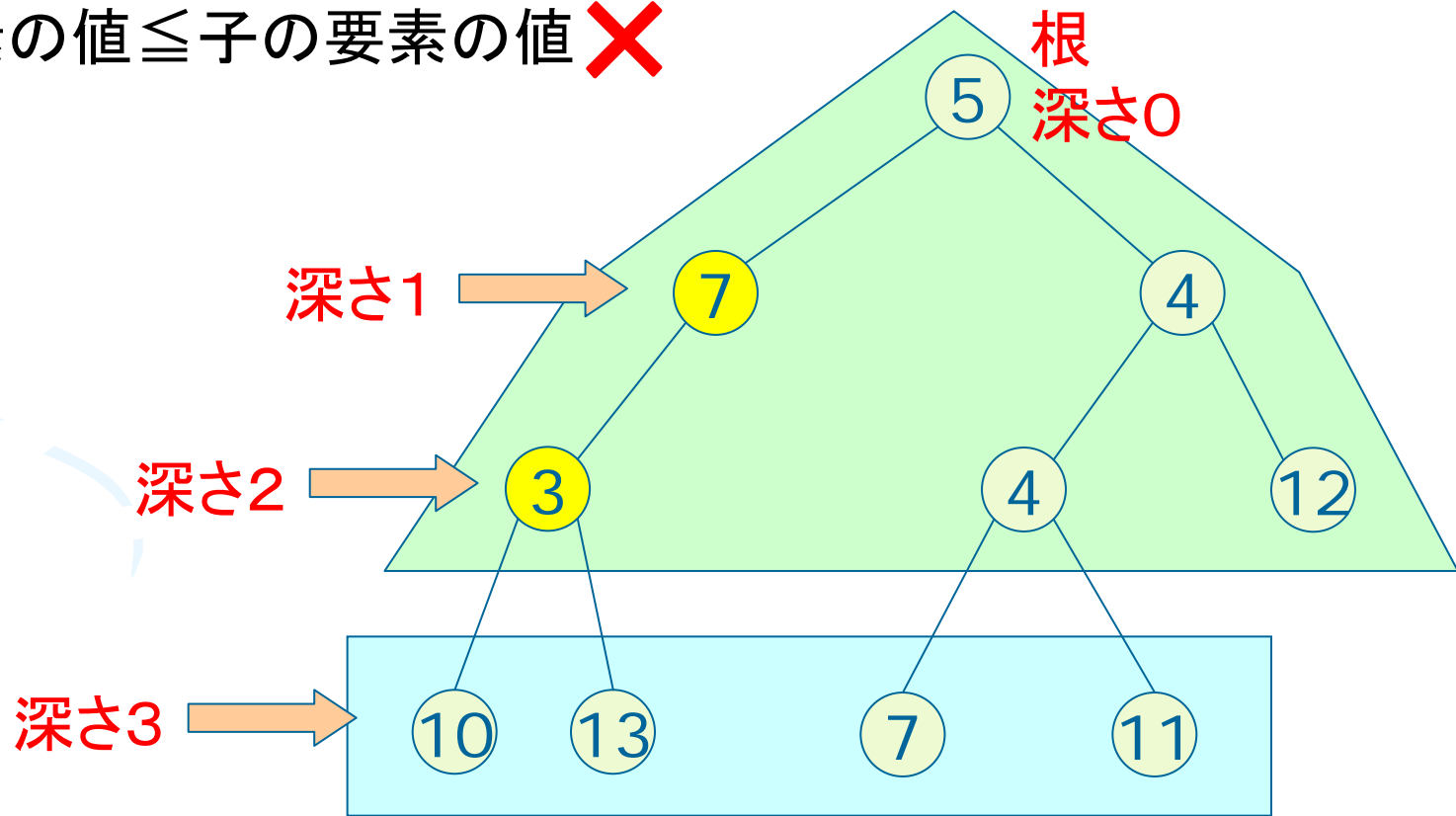
# ヒープではない2分木の例

- 次の条件を満たす2分木をヒープと呼ぶ

(1) 木の高さが  $h \rightarrow$  深さ  $h-1$  までは完全2分木 **×**

深さ  $h$  では, 左側に葉が詰められている **×**

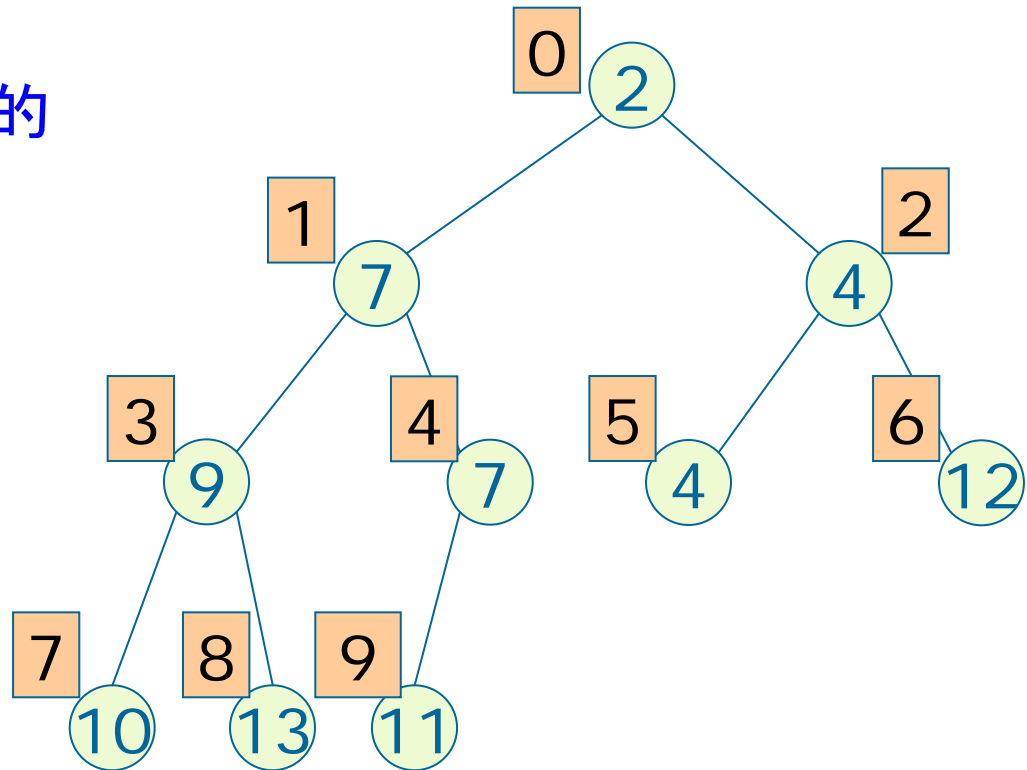
(2) 親の要素の値  $\leq$  子の要素の値 **×**



# ヒープの実現方法

- ヒープは2分木  
→ **ポインタと構造体**を使って  
実現可能
- ヒープは**配列**を使っても  
実現可能 ← **こちらが一般的**

0	1	2	3	4	5	6	7	8	9	10
2	7	4	9	7	4	12	10	13	11	-



根から葉に向かって  
上から下に, 左から右に  
番号を付ける  
→この順番で配列に  
入れる

# ヒープの配列による実現

根と最後の要素は簡単に見つかる:

$A[0]$  は根

$A[n-1]$  は最後の要素

( $n$ : 要素の数)

与えられた要素  $A[k]$  の子が簡単に見つかる:

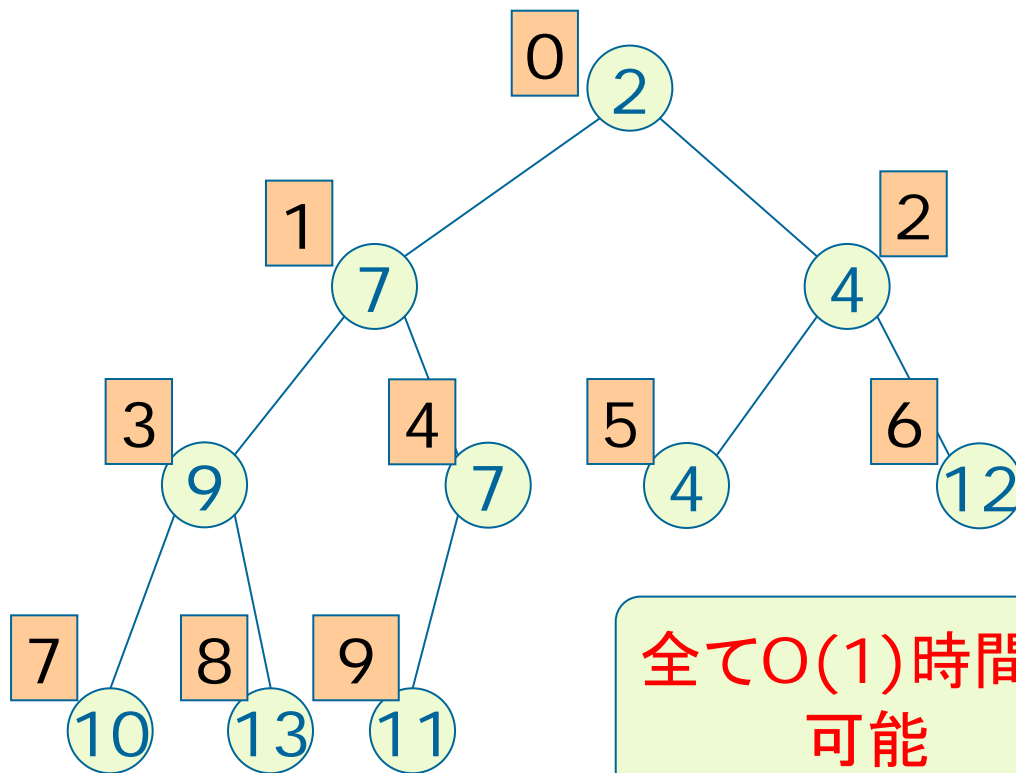
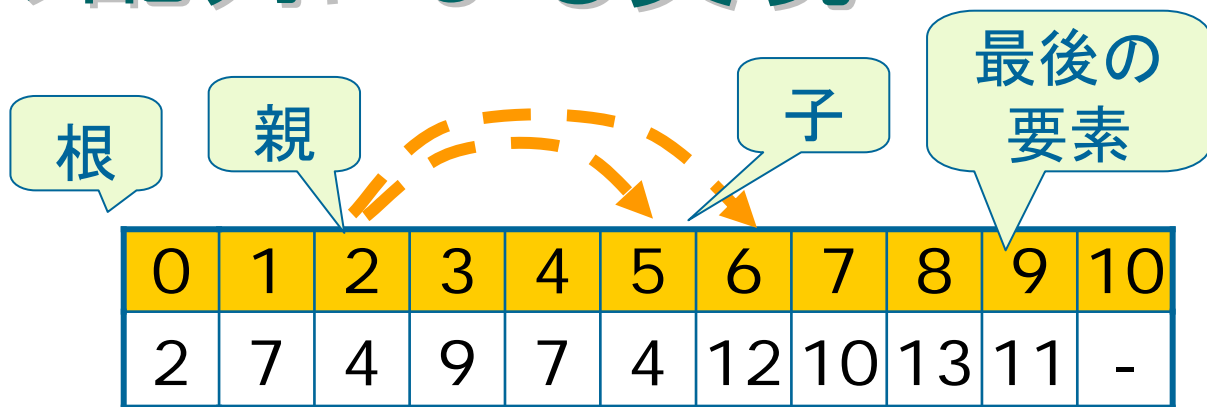
左の子  $A[2k+1]$

右の子  $A[2k+2]$

与えられた要素の親が簡単に見つかる:

$A[k]$  の親は

$A[\lfloor (k-1)/2 \rfloor]$



全て  $O(1)$  時間で可能

# 最小の要素を削除する

- 根の要素が最小値

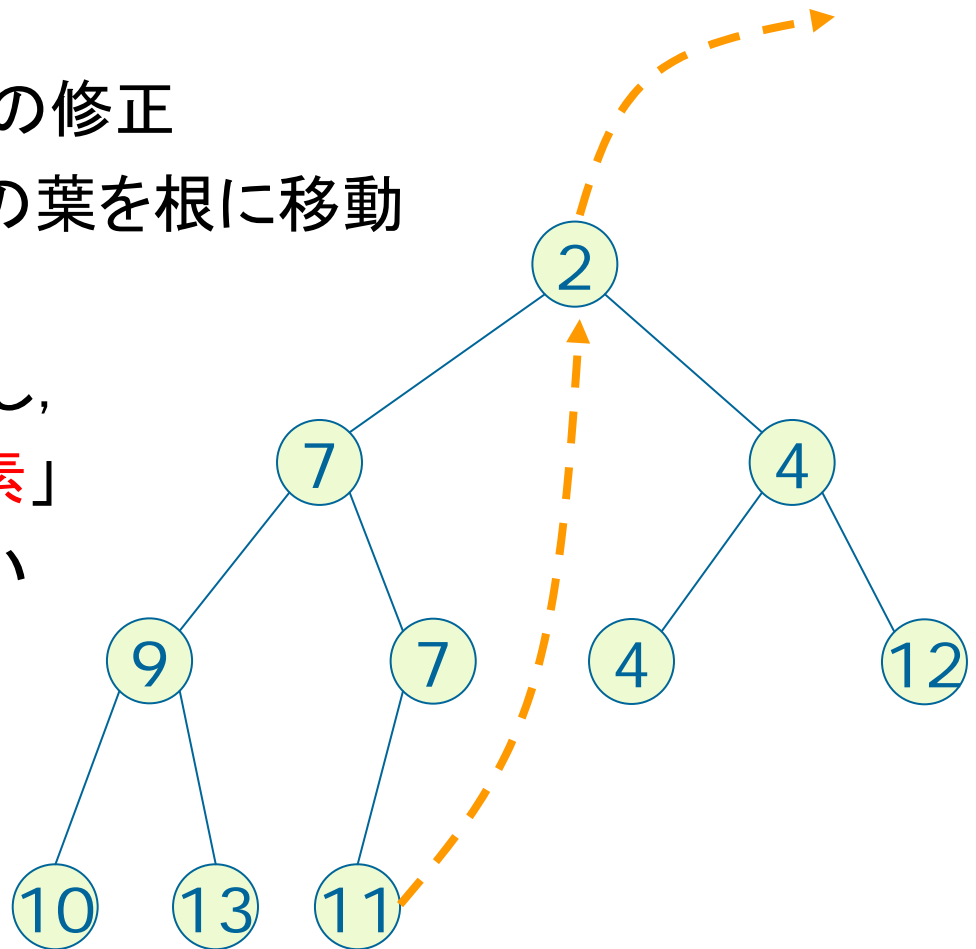
(1) 根の節点を削除

→ ヒープでなくなる → 木の修正

(2) 一番深いレベルの右側の葉を根に移動

→ 2分木になった

しかし、新しい根に対し、  
「親の要素  $\leq$  子の要素」  
の条件は成り立たない



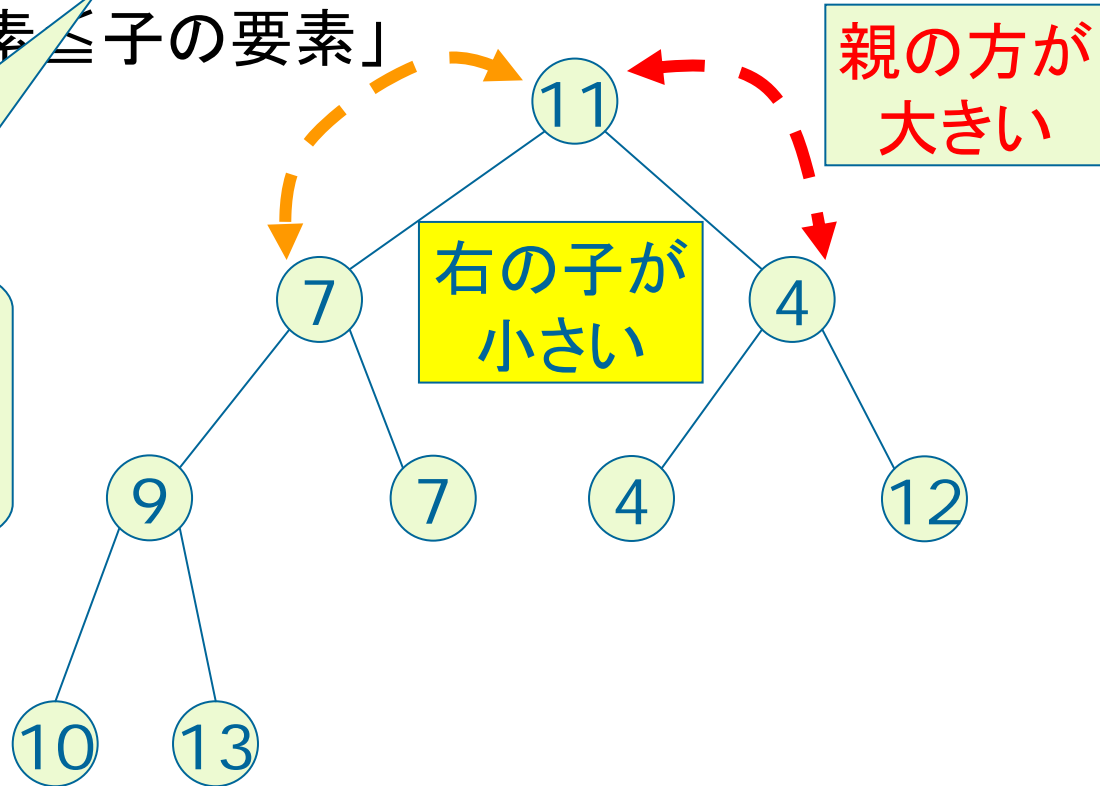
# 最小の要素を削除する

(3) 新しい根の要素を子と比較し、「親の要素 > 子の要素」が成り立つときは、繰り返し親子を交換

- 2つの子のうち、要素の小さい方の子と親を交換

→ 交換後は「親の要素 ≤ 子の要素」が成り立つ

交換する子の選び方を間違えると、条件が成り立たない



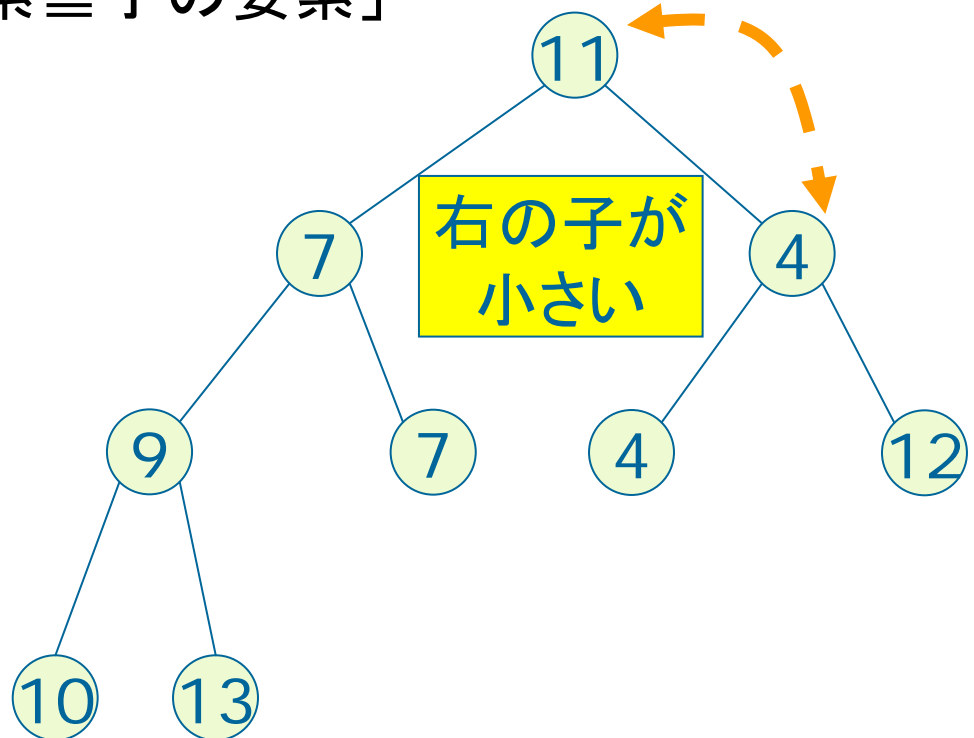


# 最小の要素を削除する

(3) 新しい根の要素を子と比較し、「親の要素  $>$  子の要素」が成り立つときは、繰り返し親子を交換

– 2つの子のうち、要素の小さい方の子と親を交換

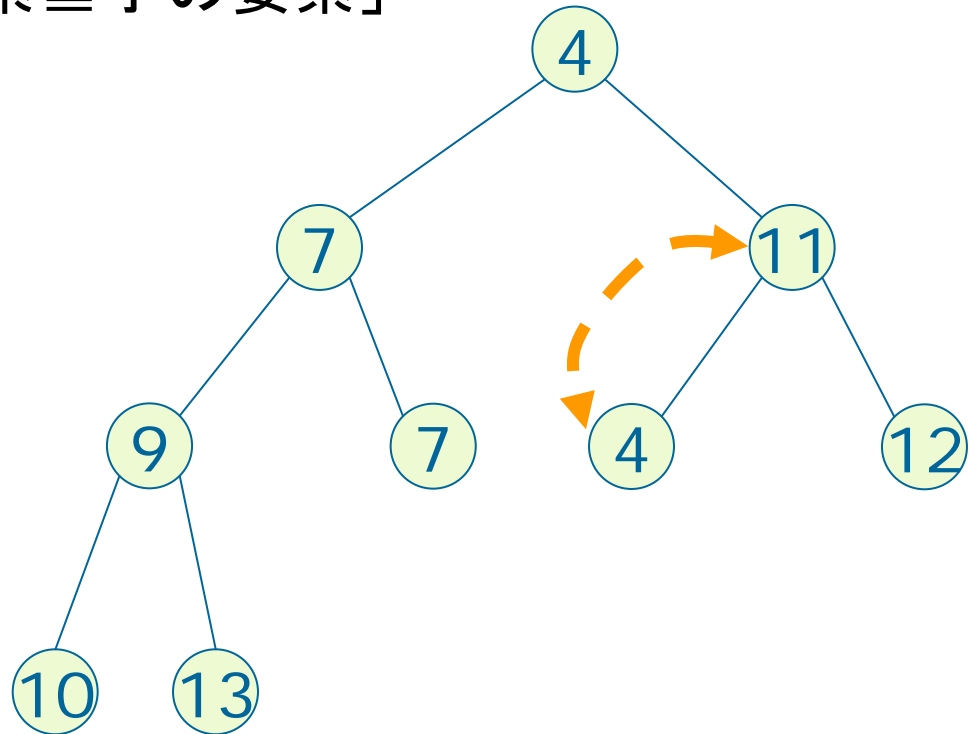
→ 交換後は「親の要素  $\leq$  子の要素」が成り立つ



# 最小の要素を削除する

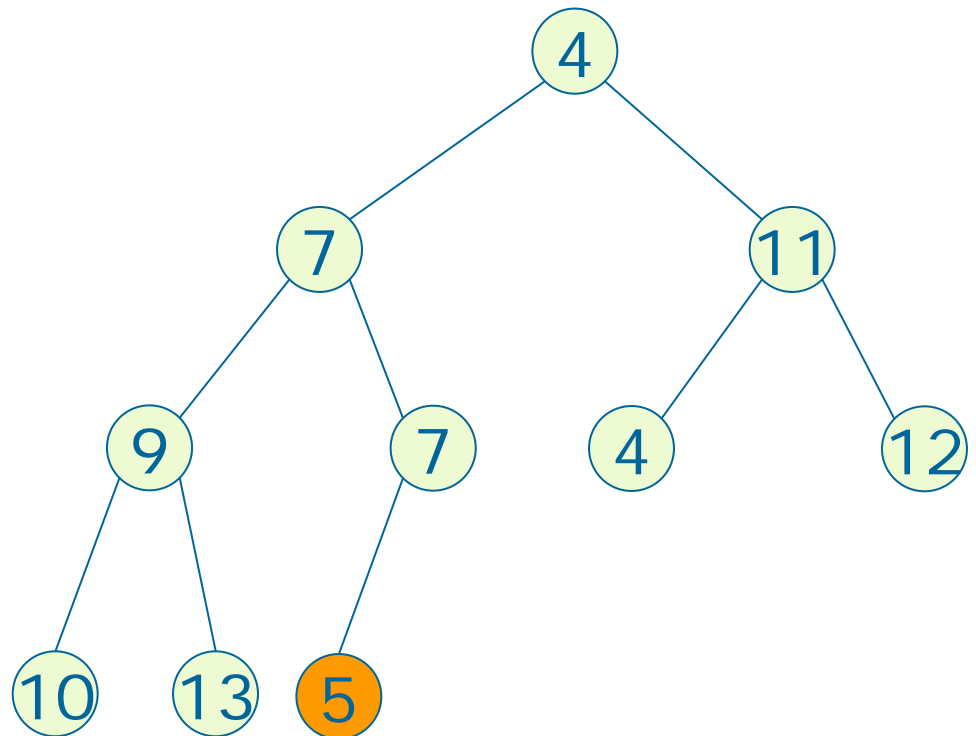
- (3) 新しい根の要素を子と比較し、「親の要素  $>$  子の要素」が成り立つときは、繰り返し**親子を交換**
- 2つの子のうち、**要素の小さい方の子と親を交換**
  - 交換後は「親の要素  $\leq$  子の要素」が成り立つ

親子の入れ替えの回数  
 $\leq$  木の高さ  
→ 時間計算量は  
 $O(\log n)$



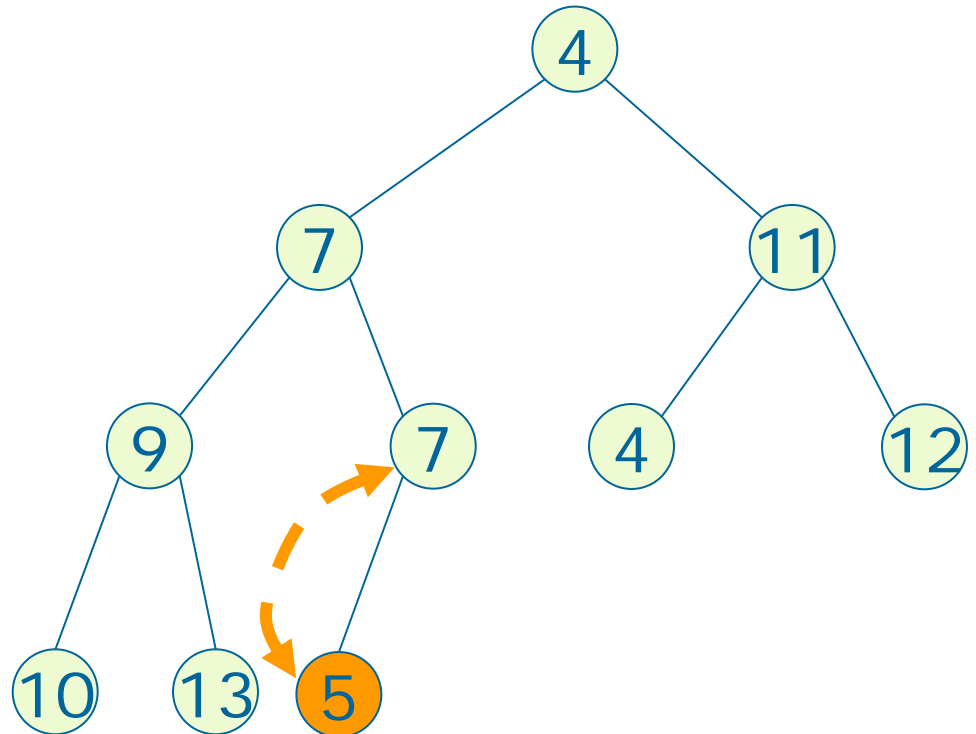
# 新しい要素を追加する

(1) 一番深いレベルの最も右側の葉の隣(空かない場合は次のレベルの一番左)に新しい要素を挿入



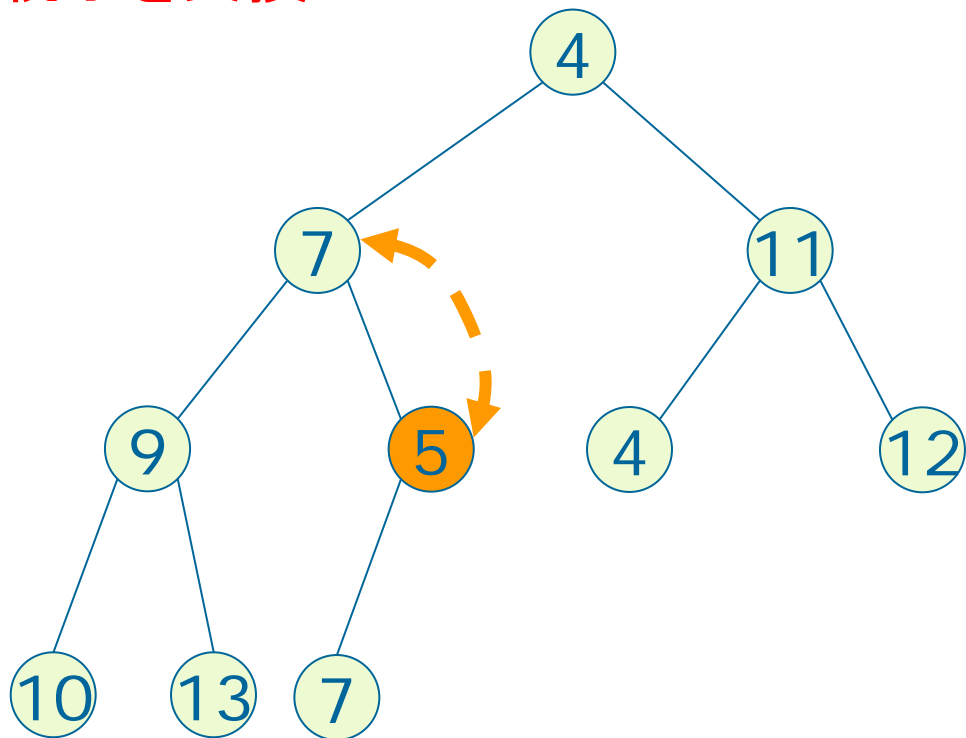
# 新しい要素を追加する

- (1) 一番深いレベルの最も右側の葉の隣(空がない場合は次のレベルの一番左)に新しい要素を挿入
- (2) 新しい要素を親と比較し、「親の要素 > 子の要素」が成り立つときは、繰り返し**親子を交換**



# 新しい要素を追加する

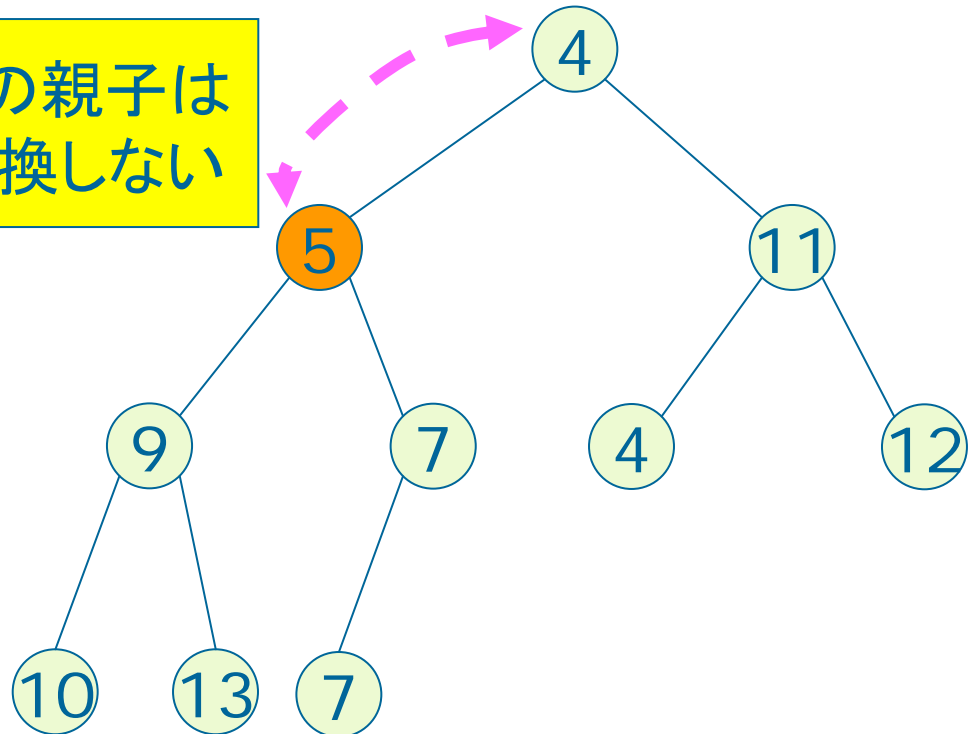
- (1) 一番深いレベルの最も右側の葉の隣(空がない場合は次のレベルの一番左)に新しい要素を挿入
- (2) 新しい要素を親と比較し、「親の要素  $>$  子の要素」が成り立つときは、繰り返し**親子を交換**



# 新しい要素を追加する

- (1) 一番深いレベルの最も右側の葉の隣(空がない場合は次のレベルの一番左)に新しい要素を挿入
- (2) 新しい要素を親と比較し、「親の要素 > 子の要素」が成り立つときは、繰り返し**親子を交換**

この親子は  
交換しない



ステップ(2)が  
行なわれる度に  
新しい要素は  
一つ上に上がる

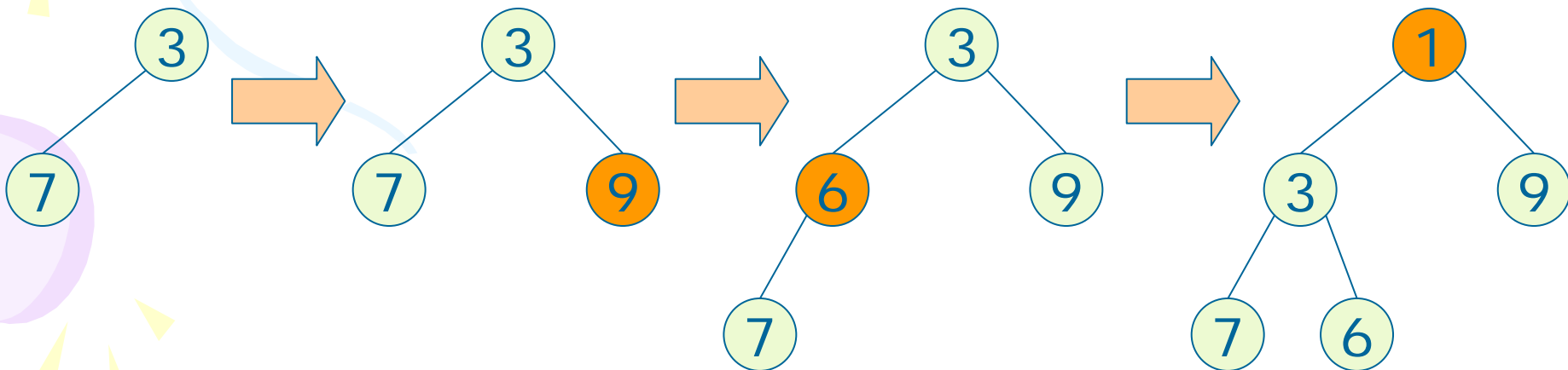
→ 反復回数 ≤ 木の高さ

→ 時間計算量は  $O(\log n)$

# ヒープソート

- ヒープを使った整列アルゴリズム
  - アルゴリズムの手順
    - (1) 与えられた数の集合を順番にヒープに追加 (n 回)
    - (2) ヒープから最小要素を次々に削除, 並べる (n 回)
- $O(n \log n)$  時間

7, 3, 9, 6, 1 を順に追加



# ヒープソート

- ヒープを使った整列アルゴリズム

- アルゴリズムの手順

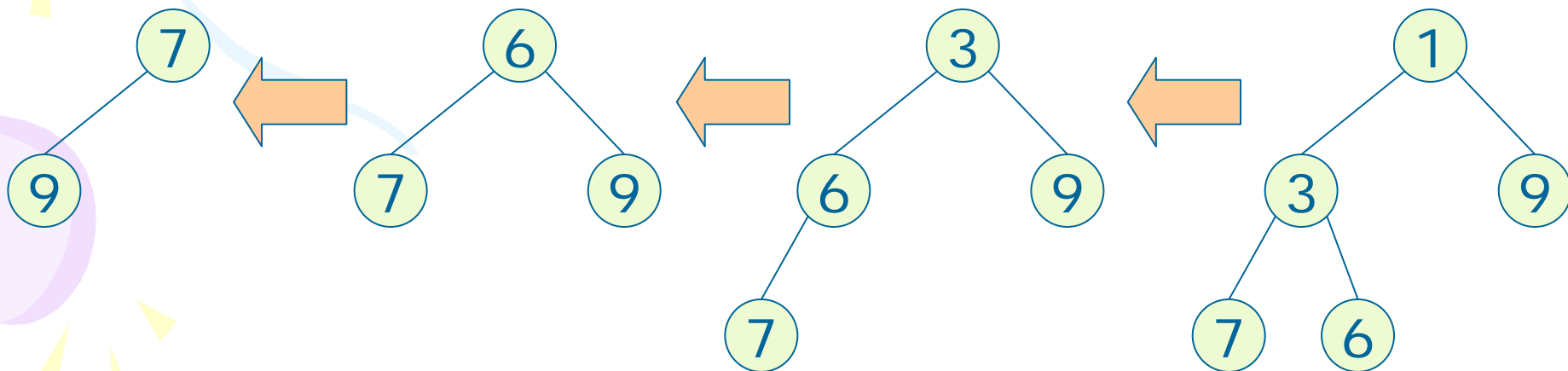
(1) 与えられた数の集合を順番にヒープに追加 (n 回)

(2) ヒープから最小要素を次々に削除, 並べる (n 回)

→  $O(n \log n)$  時間

最小要素を順に削除

① ③ ⑥ ⑦ ⑨





# レポート(紙で提出)

## 問1

- (1) 数の集合 $\{15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1\}$ に対するヒープを二分木の形で書きなさい
- (2) (1)で求めたヒープに対して, 最小要素を削除するときの動作を, 図を使って詳しく説明せよ.
- (3) (2)で求めたヒープに対して, 要素10を追加するときの動作を, 図を使って詳しく説明せよ.

## 問2

スタックは連結リスト(双方向ではないリスト)を使っても実現できる. 連結リストを用いた場合, 追加と削除が $O(1)$ 時間で行えることを(図を使って)説明せよ.

締切: 5月27日(木)午前9時頃