

データ構造とアルゴリズム

第 5 回 探索

小池 英樹 (koike@c.titech.ac.jp)

探索

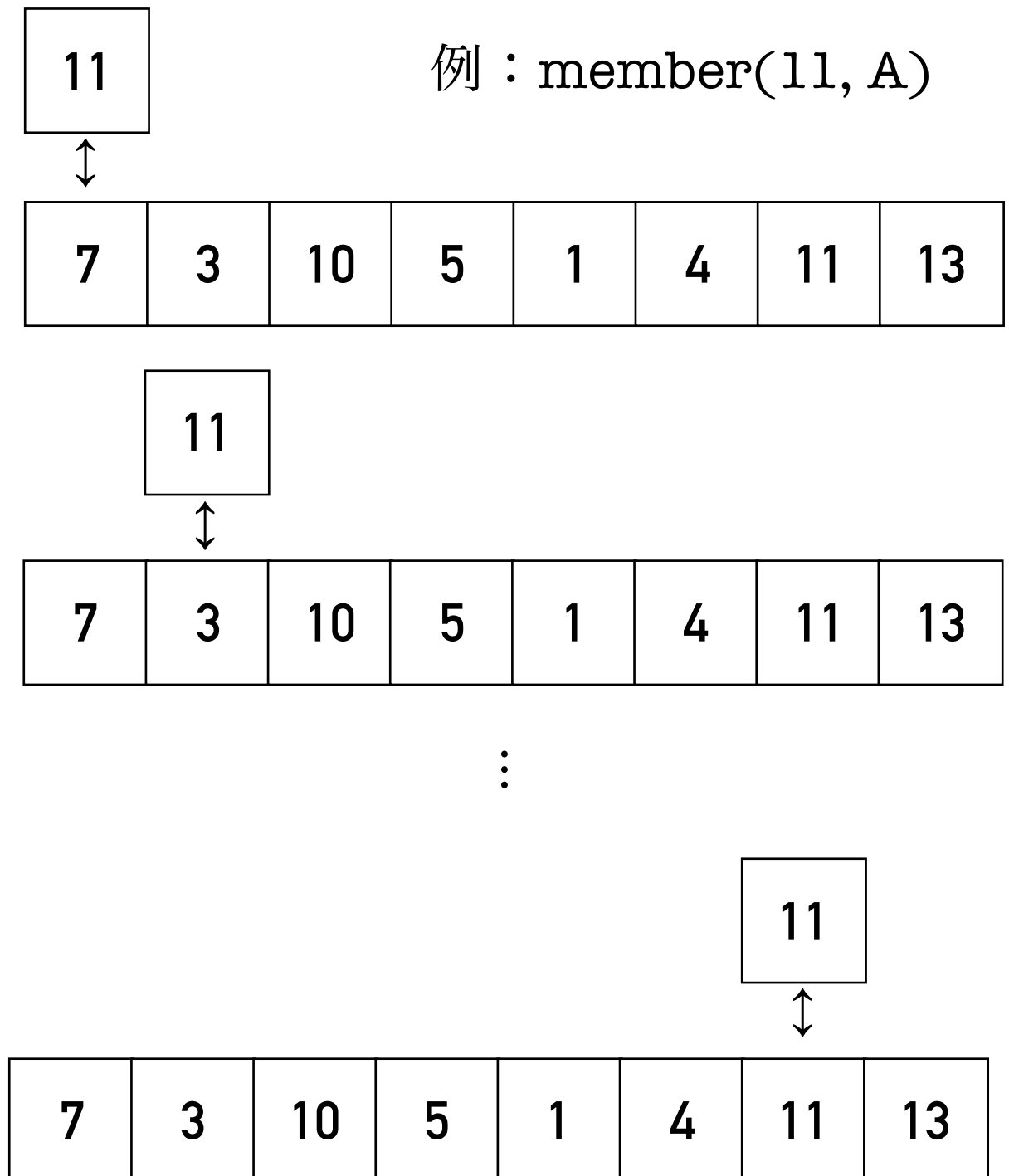
- 線形探索
- 2分探索
- ヒープ
- 2分探索木

線形探索：原理

- MEMBER(x, A)

- 先頭から順に比較

- $O(n)$



線形探索：挿入と削除

➤ INSERT(x , A)

- 先頭に挿入すればよい $\rightarrow O(1)$

➤ DELETE(x , A)

- 先頭から順に調べて、 $\text{element} = x$ があったらそのデータを削除 $\rightarrow O(n)$

順序付き集合

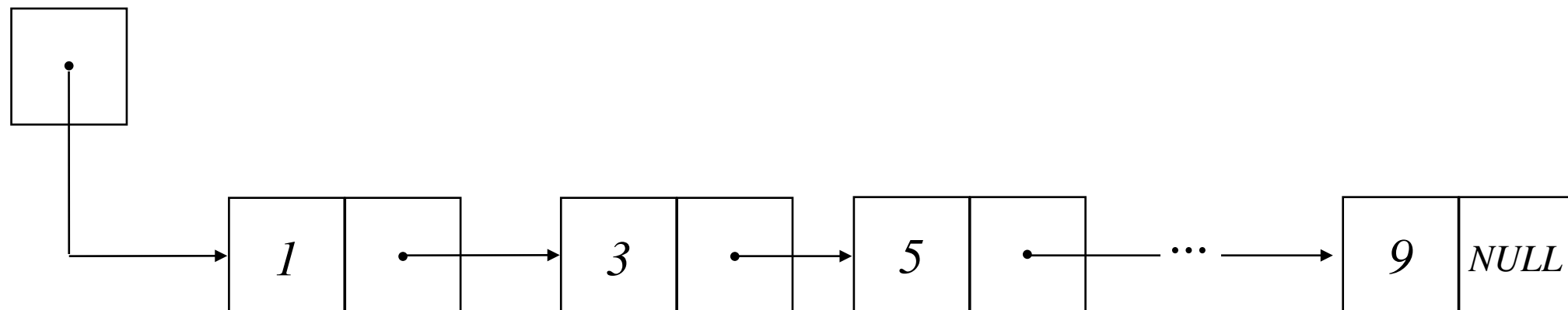
- ▶ 全順序(total order, 線形順序(linear order))
 - ▶ 反射律(reflexivity): すべての x に対し $x \leq x$
 - ▶ 推移律(transitivity): $x \leq y$ かつ $y \leq z$ ならば $x \leq z$
 - ▶ 反対称律(anti-symmetry): $x \leq y$ かつ $y \leq x$ ならば $x = y$
 - ▶ 比較可能性(comparability): 任意の x と y に対し $x \leq y$ あるいは $y \leq x$ が成り立つ
- ▶ 集合 A の要素間に全順序が定義されているとき, A を順序付き集合(ordered set)という.

順序付き集合

- ▶ よく必要となる操作
 - ▶ $\text{MIN}(A)$: $A \neq \Phi$ ならば, \leq に関し最小の要素を返す
 - ▶ $\text{DELETETEMIN}(A)$: $A \neq \Phi$ のとき, 最小の要素 (複数個ある場合はその一つ) を A から除く.

優先度つき待ち行列 (PRIORITY QUEUE)

- 順序つき集合のうち、とくにINSERT(x , A), DELETEMIN(A)を持つもの
- 連結リストによる実装
 - 整列しない場合：
 - DELETEMIN(A): 先頭から走査し最小要素を見つける $\rightarrow O(|A|)$
 - INSERT(x , A): 先頭に挿入するだけ $\rightarrow O(1)$
 - あらかじめ小さなものから整列してある場合：
 - DELETEMIN(A): 先頭要素を削除する. $\rightarrow O(1)$
 - INSERT(x , A): リストを先頭から走査し挿入. $\rightarrow O(|A|)$



2分探索：原理

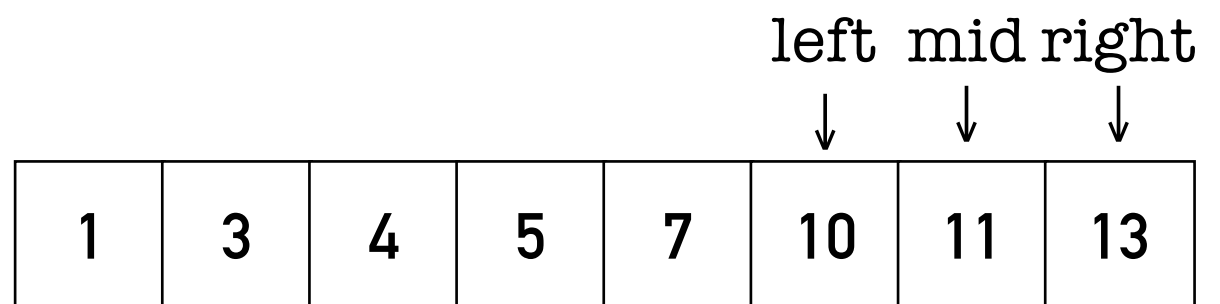
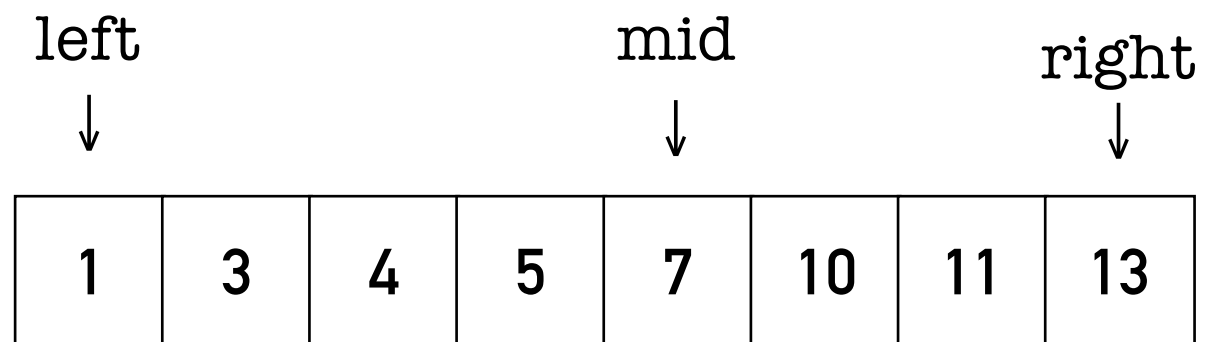
- データがあらかじめ順番に並べられていることを仮定

例：member(11, A)

- MEMBER(x, A)

- 探索範囲の中央を比較

- $O(\log n)$



2分探索：アルゴリズム

```
int member(x, A) {  
    left = 0, right = n  
    while left < right  
        mid = [(left+right)/2]  
        if x = A[mid]  
            探索成功して終了.  
  
        else if x < A[mid]  
            right = mid  
        else if x > A[mid]  
            left = mid+1  
        探索失敗して終了.  
}
```

2分探索：挿入と削除

- ▶ 2分探索では，線形探索と同様にデータを大きさ順に並べなければならないので，挿入と削除が面倒 ($O(n)$).
- ▶ 本講義（第2回）のリストの実現における挿入，削除の手続きに相当する.


優先度つき待ち行列

- ▶ INSERTとDELETEMINの両方をもっと効率よくできないか？

ヒープ(HEAP)

研究社 新英和中辞典での「heap」の意味

►heap

音節 heap 発音記号 / hɪːp / 音声を聞く 

名詞 可算名詞

1 積み重ね, かたまり, 山.

- **in a heap [heaps]** 山をなして.

2 《口語》

a [通例 a heap of... または heaps of... で] たくさん, どっさり 《★【比較】 a lot of, lots of のほうが一般的》.

⊕ **You do know a heap of things, don't you?** ずいぶんいろんなことをご存じですね.

b [heaps; 副詞的に] 大いに, ずっと.

⊕ **The patient is heaps better.** 患者はずっとよくなった.

3 《俗語》 ぼんこつ車; 荒れ果てた建物.

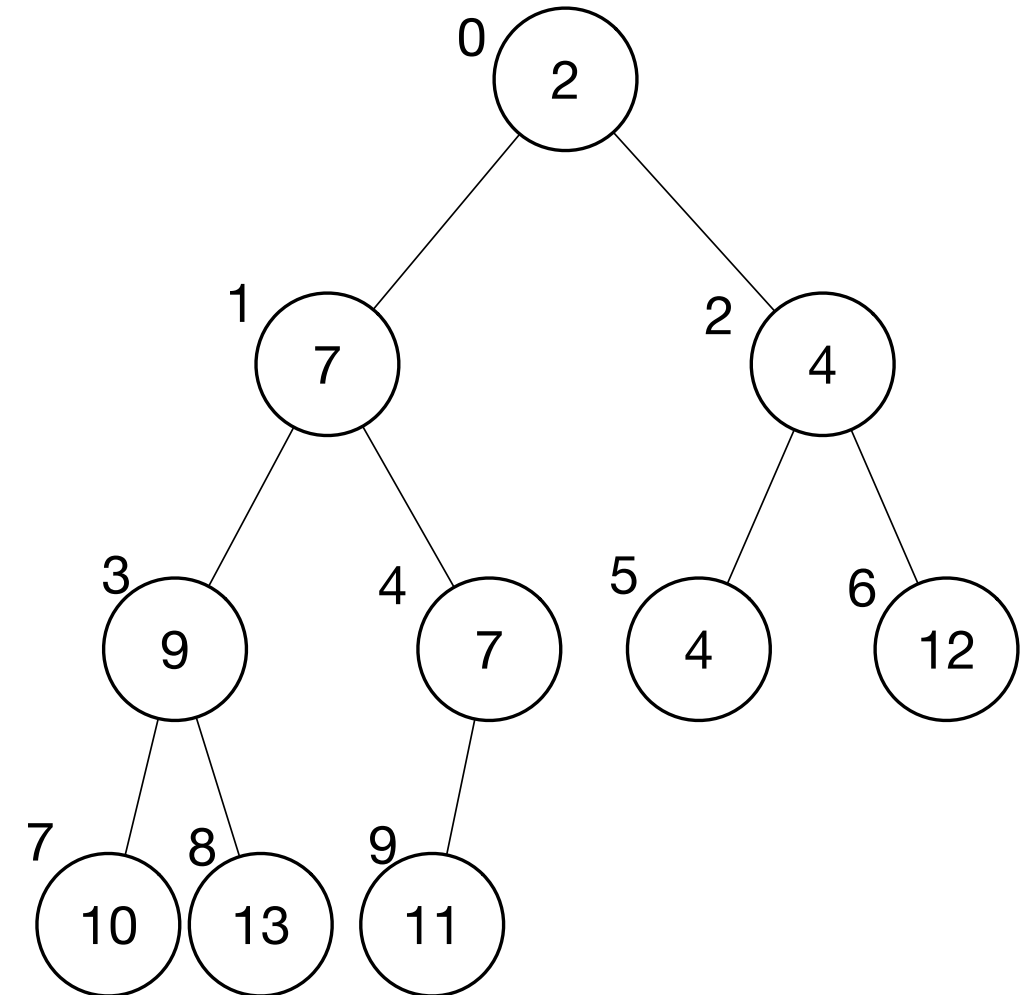


ヒープ(HEAP)

➤ ヒープ条件 (重要)

- (1) 木の高さをhとすると深さh-1までは完全2分木で，深さhの葉は木の左に詰められている
- (2) 節点vの親をuとすると，それぞれの要素 x_v ， x_u は次の条件を満たす

$$x_u \leq x_v$$



ヒープ(HEAP)

- 配列による実現

- 節点番号 $i=0, 1, \dots, n-1$ を上から下へ, 同一深さでは左から右に走査して決める.

- 節点 i の要素を $A[i]$ に入れる

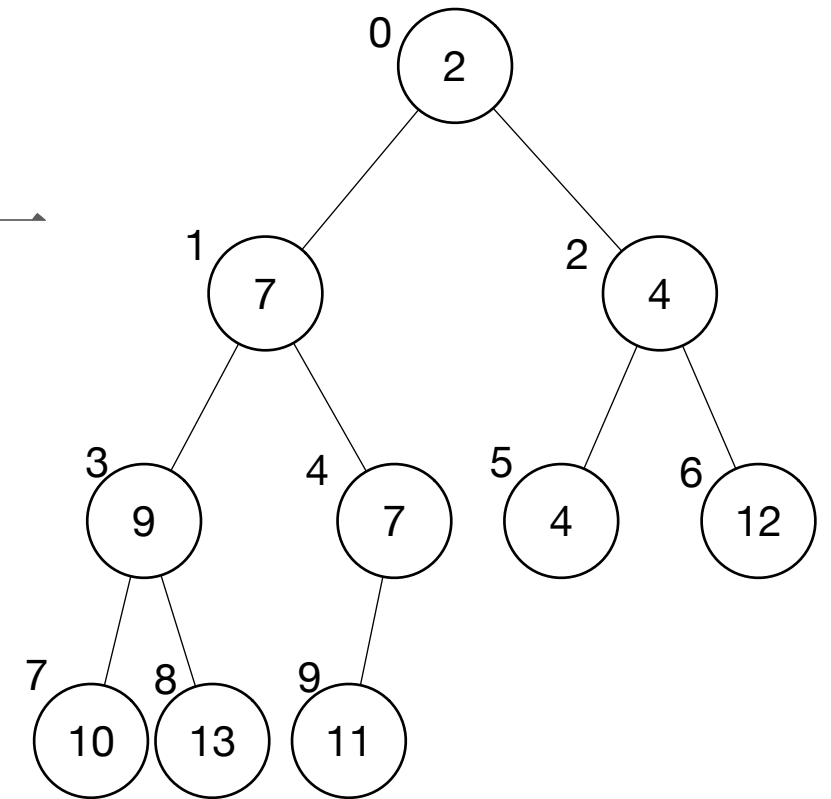
- 特徴: (重要)

- $A[0]$ は根

- $A[n-1]$ は最後の要素

- $A[i]$ の左の子は $A[2i+1]$, 右の子は $A[2i+2]$

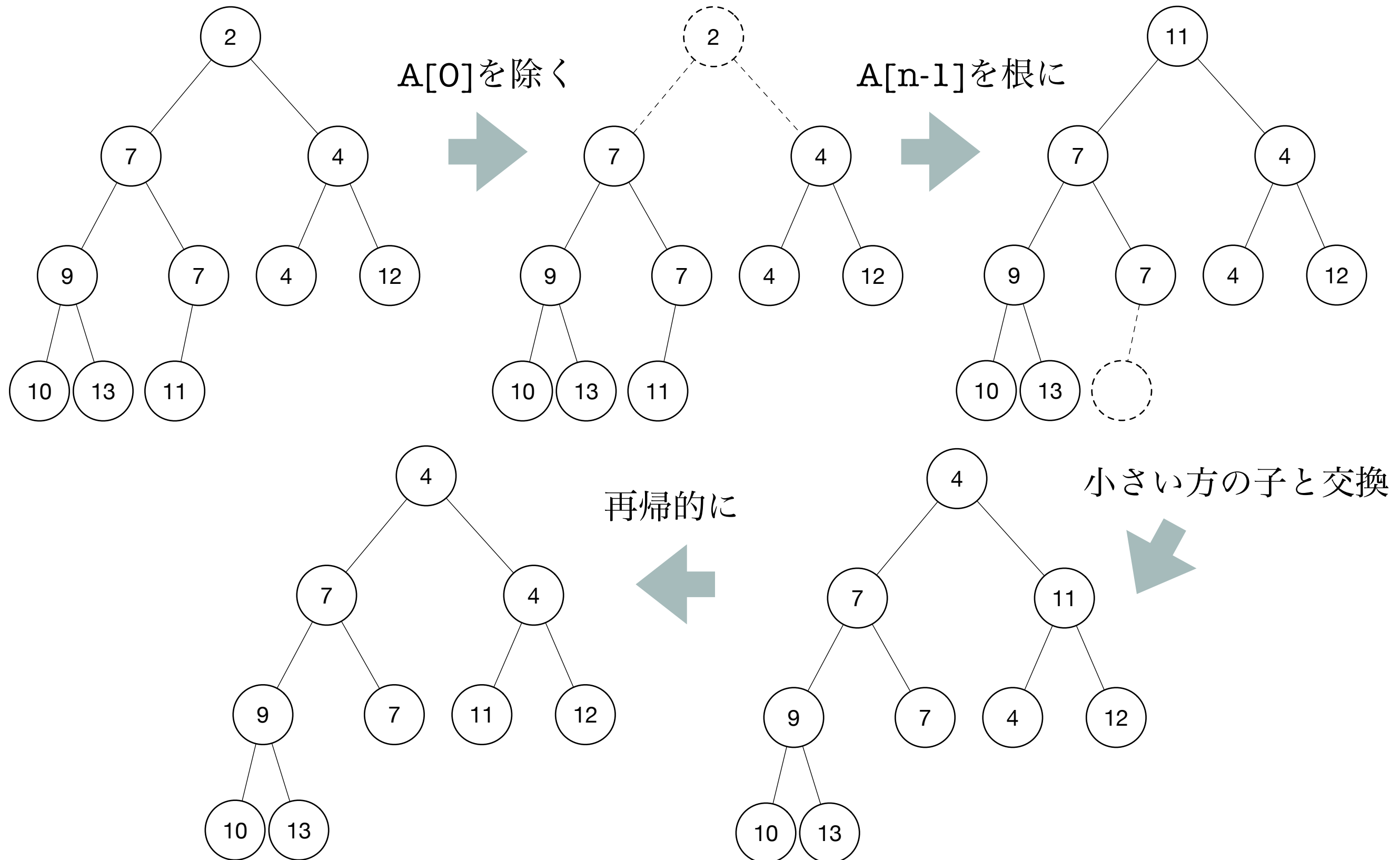
- $A[i]$ の親は $A[\lfloor (i-1)/2 \rfloor]$



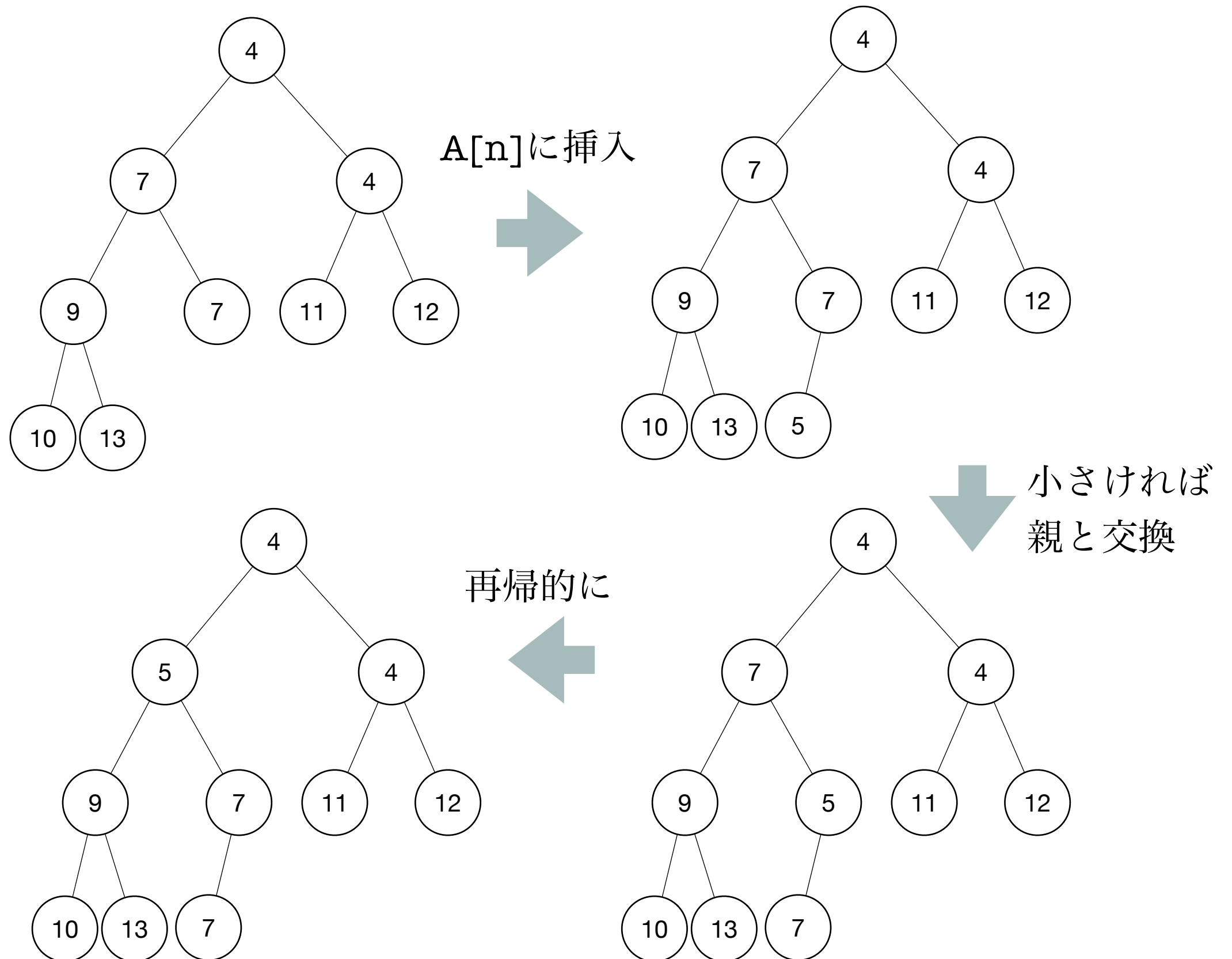
0	1	2	3	4	5	6	7	8	9	10
2	7	4	9	7	4	12	10	13	11	-

配列による実現($n=10, N=11$)

HEAPからの最小要素の削除 (DELETEMIN(A))

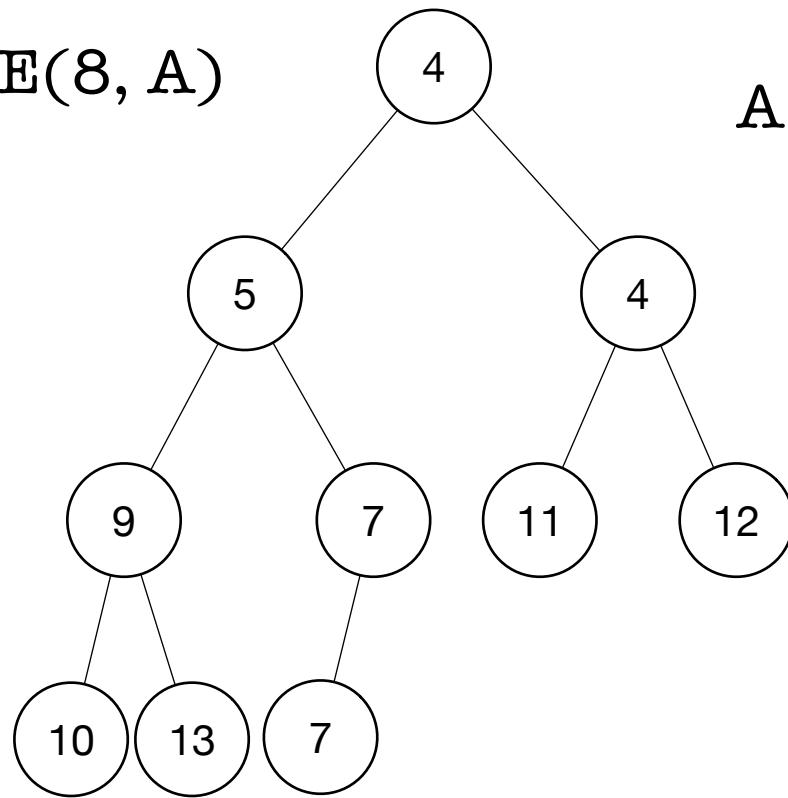


HEAPへの挿入(INSERT(X, A))

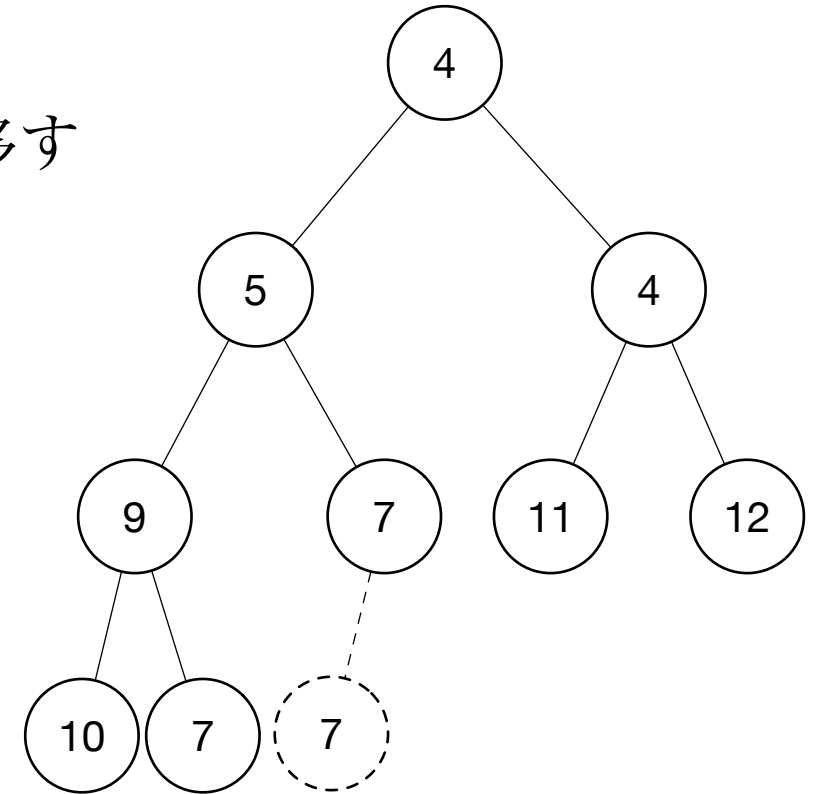


HEAPからの値の削除(DELETE(I, A))

例：DELETE(8, A)



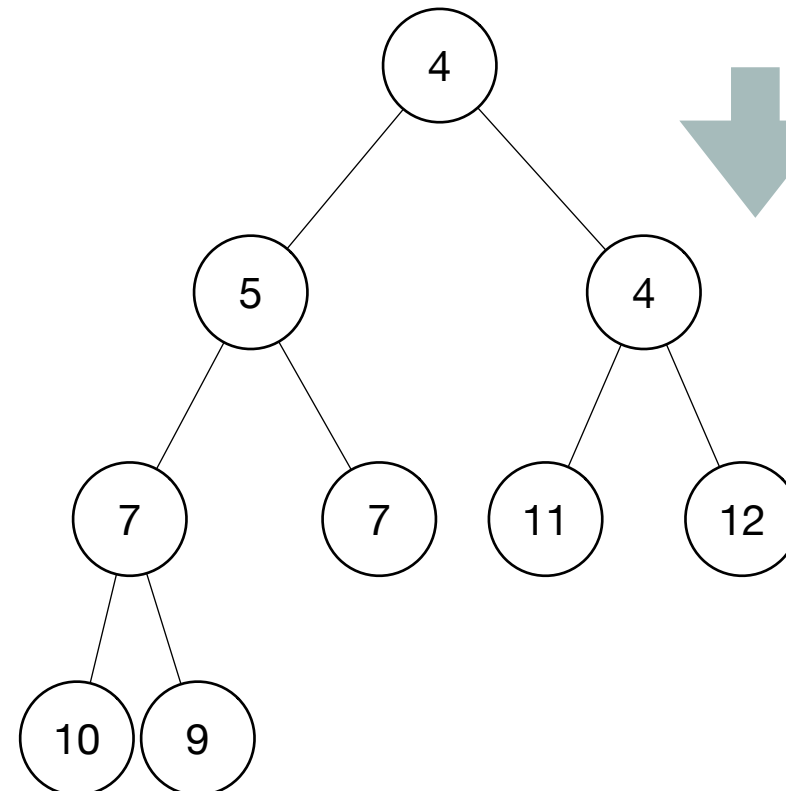
A[n-1]をA[i]に移す



A[i]が親より小さかったら交換



再帰的に

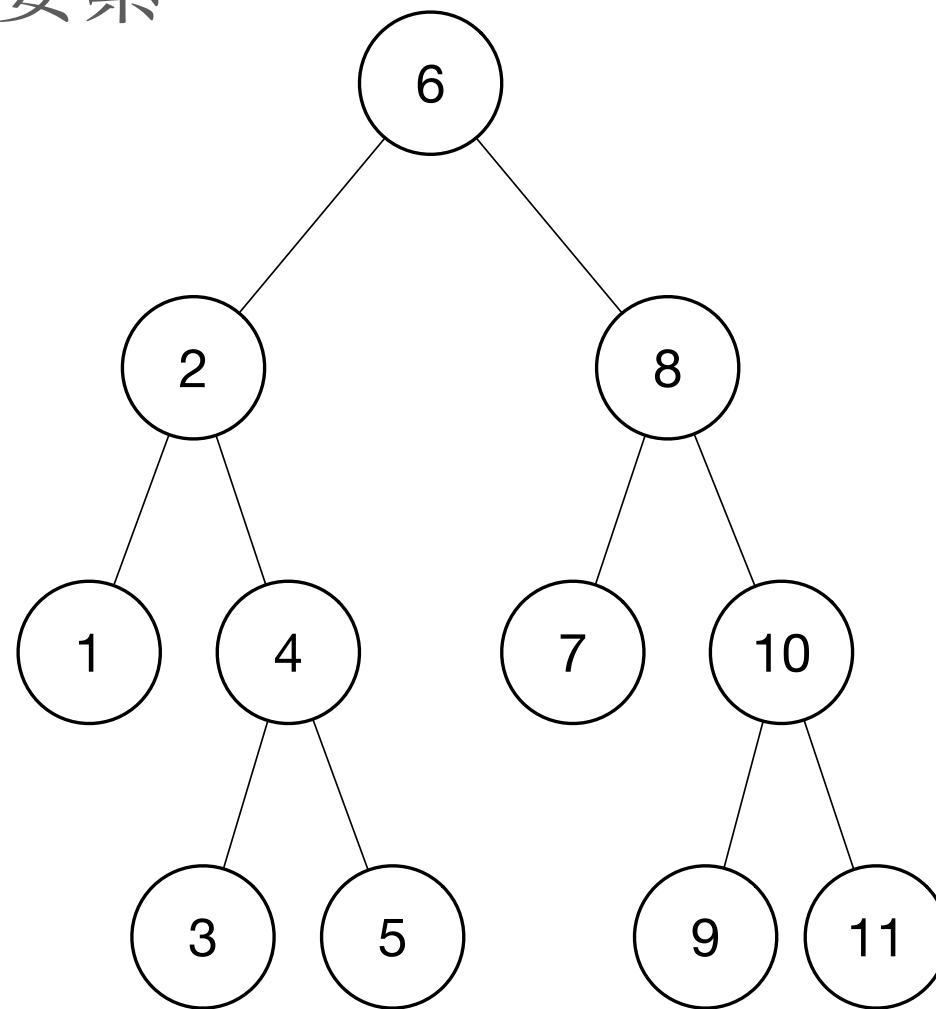


HEAPの計算量

- $|A| = n$ として：
- $\text{DELETMIN}(A) \rightarrow O(\log n)$
- $\text{INSERT}(x, A) \rightarrow O(\log n)$
- $\text{DELETE}(i, A) \rightarrow O(\log n)$

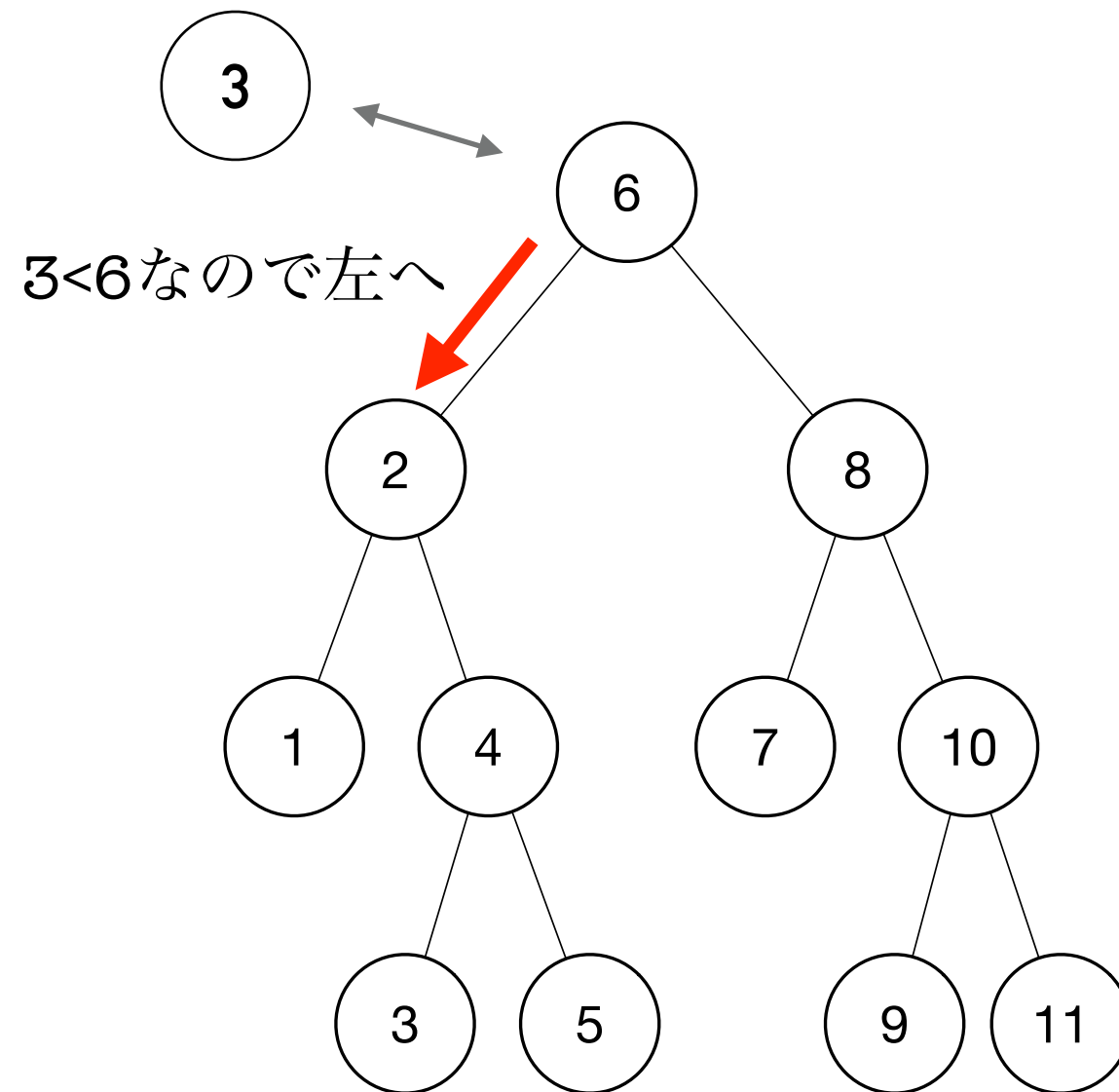
2分探索木

- ▶ 任意の節点 u に対して次の条件が成り立つ2分木（重要）
 - ▶ u の左部分木の任意の節点の要素 $< u$ の要素 $< u$ の右部分木の任意の節点の要素



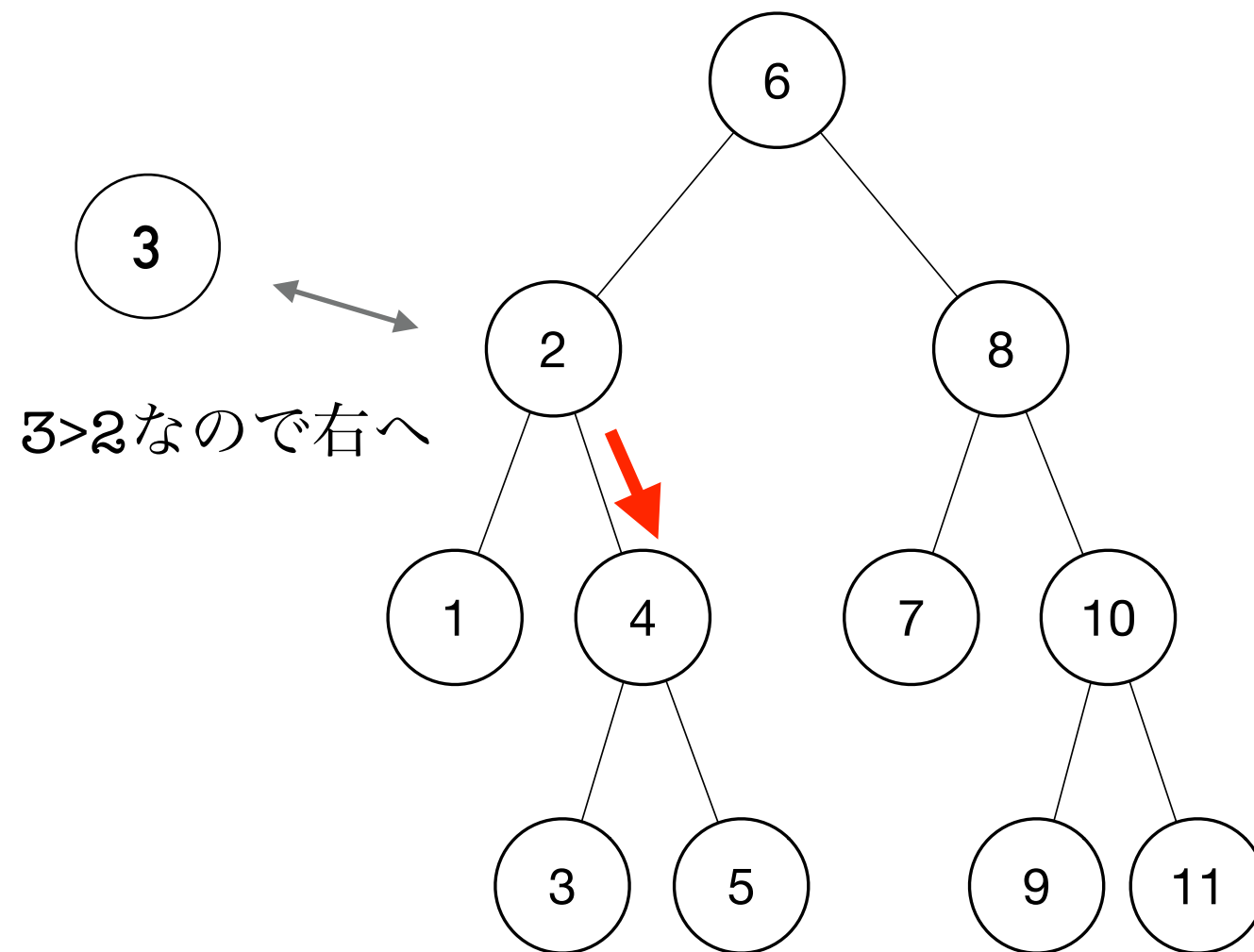
2分探索木：探索

例：MEMBER(3, A)



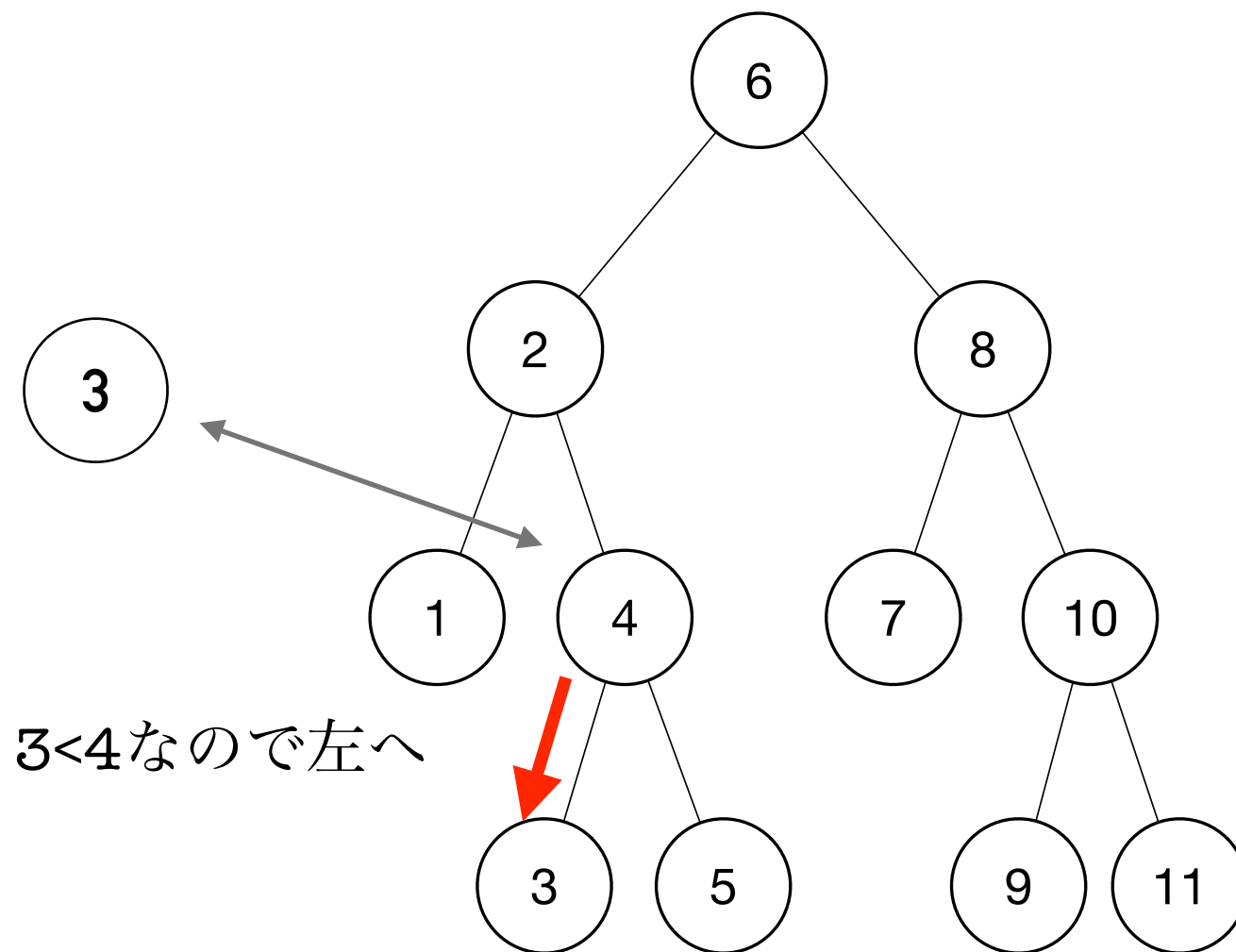
2分探索木：探索

例：MEMBER(3, A)



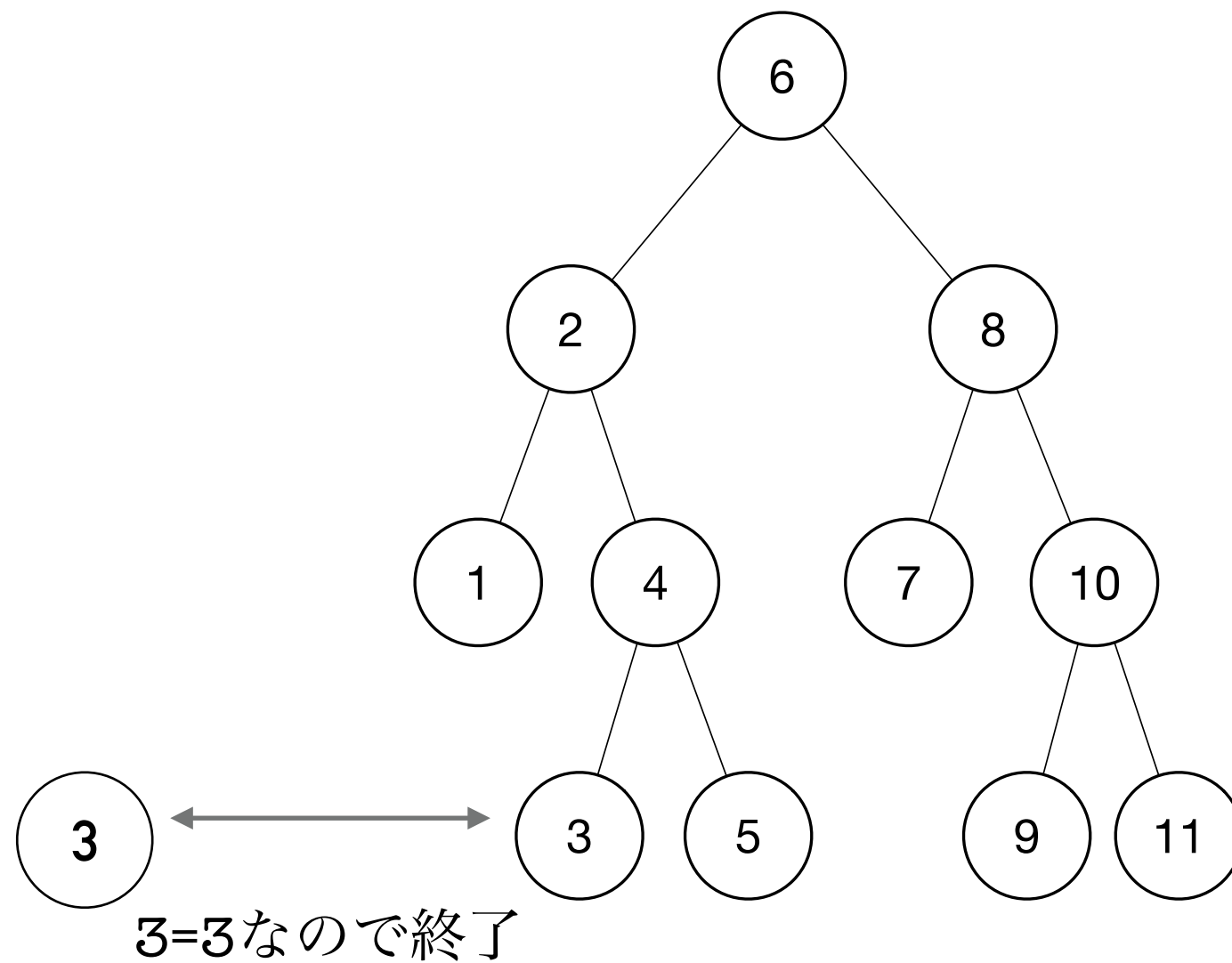
2分探索木：探索

例：MEMBER(3, A)



2分探索木：探索

例：MEMBER(3, A)



2分探索木：ループ版

```
int member(x, A) {  
    q = Aの根ノード  
    while (q != NULL) {  
        if qの要素 == x  
            探索成功して終了  
        else if qの要素 < x  
            q = qの右の子供  
        else  
            q = qの左の子供  
    }  
    探索失敗して終了  
}
```


2分探索木：プログラム例

```
enum yn {yes, no};

enum yn member(int x, struct node *p) {
    struct node *q;

    q = p;
    while (q != NULL) {
        if (q->element == x)
            return(yes);
        if (q->element < x)
            q = q->right;
        else
            q = q->left;
    }
    return(no);
}
```

2分探索木：再帰版

```
int member(x, A) {  
    q = Aの根ノード  
    if q == NULL  
        探索失敗して終了  
    else if qの要素 == x  
        探索成功して終了  
    else if qの要素 < x  
        qの右部分木を再帰的に調べる  
    else  
        qの左部分木を再帰的に調べる  
}
```

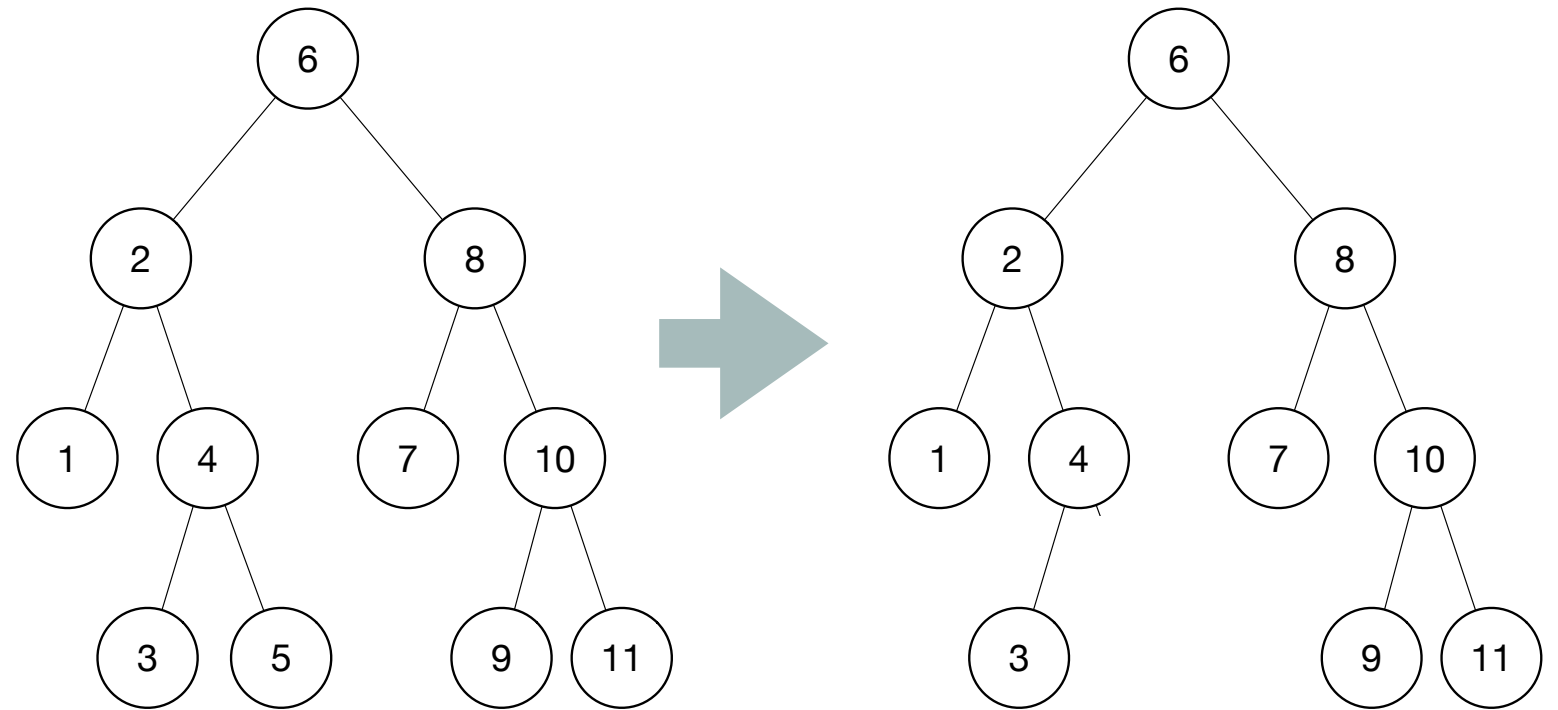
2分探索木：挿入

- ▶ 挿入は、根から要素を比較していき、葉までたどりついたらそこに新しい葉ノードを作成する

2分探索木：削除（1）

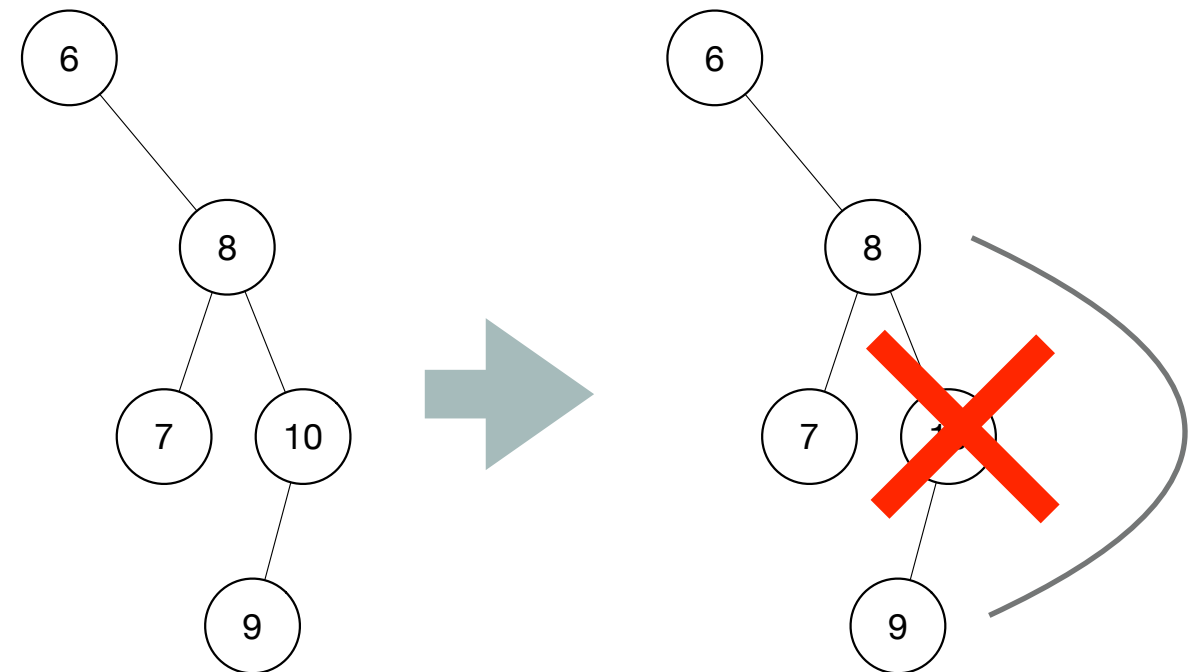
➤ ノードが葉の場合：

➤ そのまま削除



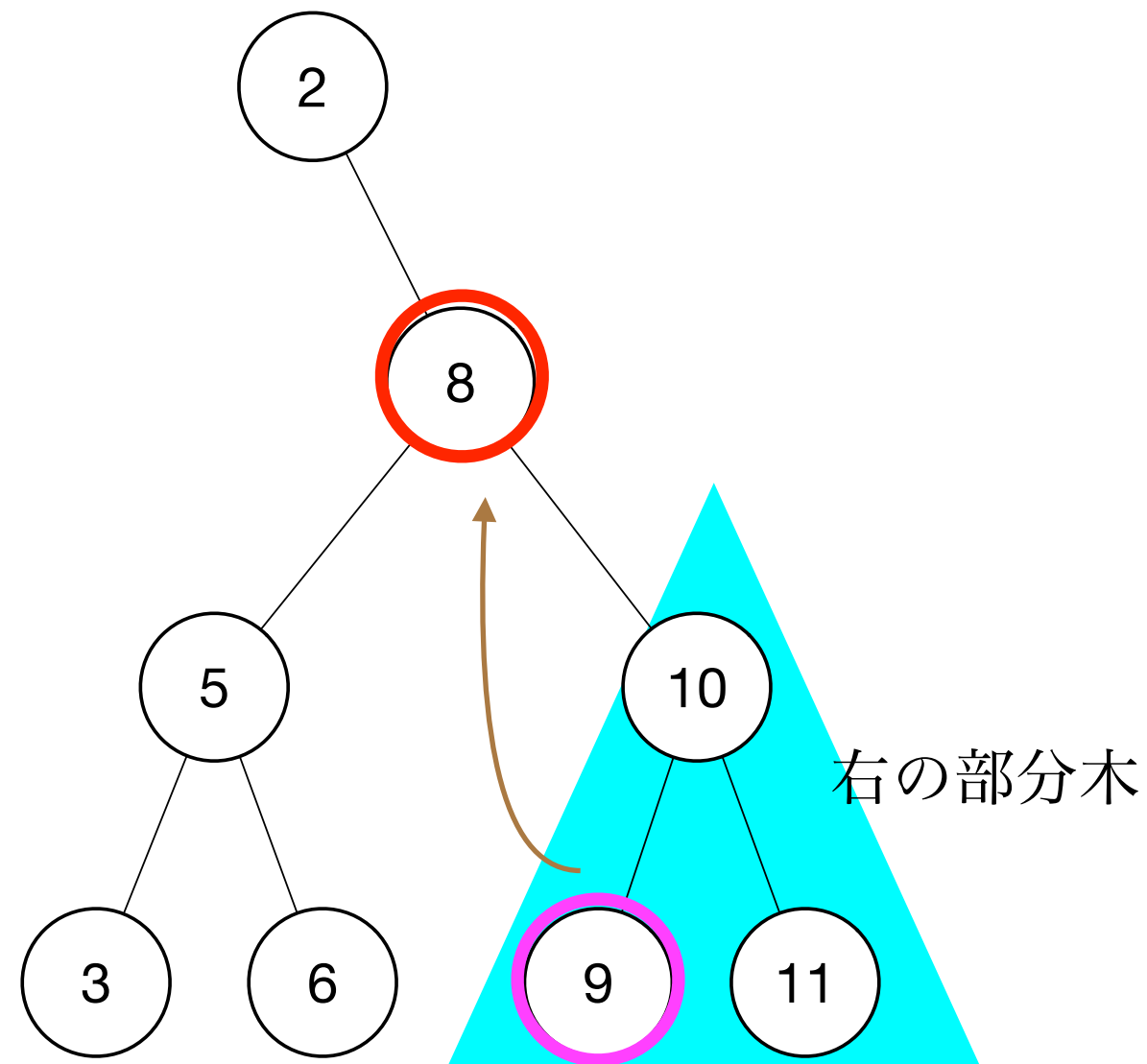
➤ ノードに子が1つの場合：

➤ 子を親とつなげる



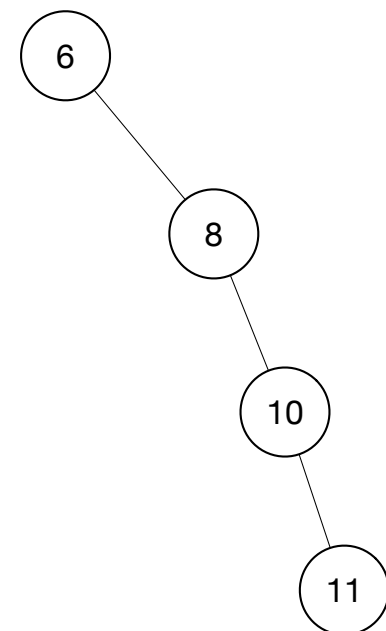
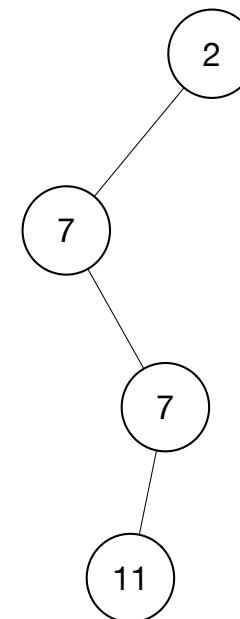
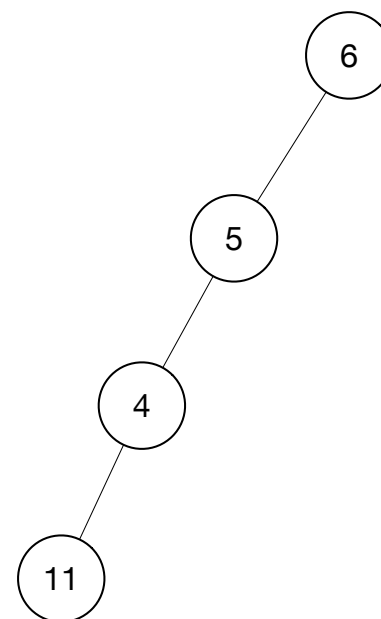
2分探索木：削除（2）

- ノードに子が2つある場合：
 - 右部分木の最小値ノードと交換する（左部分木の最大値ノードと交換しても良い）。



2分探索木の計算量

- 2分探索木に対する $\text{MEMBER}(x, A)$, $\text{INSERT}(x, A)$, $\text{DELETE}(x, A)$ の計算時間
- 最悪時間計算量 $O(n)$
- 平均時間計算量 $O(\log n)$



最悪のケース

比較

.....

	線形探索	2 分探索	2 分探索木
探索	$O(n)$	$O(\log n)$	$O(\log n)$ (ただし最悪 $O(n)$)
挿入	$O(1)$	$O(n)$	$O(\log n)$
削除	$O(n)$	$O(n)$	$O(\log n)$

レポート課題

- ▶ 配列を用いてヒープを実現し，最小要素の削除(`deletemin(x, A)`), 挿入(`insert(x, A)`)を実現しなさい.
- ▶ ポインタを用いて2分探索木を実現し，探索(`member(x, A)`), 挿入(`insert(x, A)`), 削除(`delete(x, A)`)を実現しなさい.
- ▶ 締切：12/26 17:00
- ▶ 提出先：`algo2017@vogue.cs.titech.ac.jp`