アルゴリズムとデータ構造

第4回優先度付き待ち行列
その2: 優先度付き待ち行列(ヒープ)

優先度つき待ち行列(ヒープ)

- 今日の内容:
  - 抽象データ型:優先度付き待ち行列(Priority queue)
    - 挿入(Insert)と最小値除去(deletemin)をもつリスト(またはキュー)の一種.
  - 実装方法(ヒープ). 応用としてヒープソート.
- ポイント
  - O(log n)最悪計算時間のデータ構造
  - 平衡2分木
  - 抽象データ型とその実装

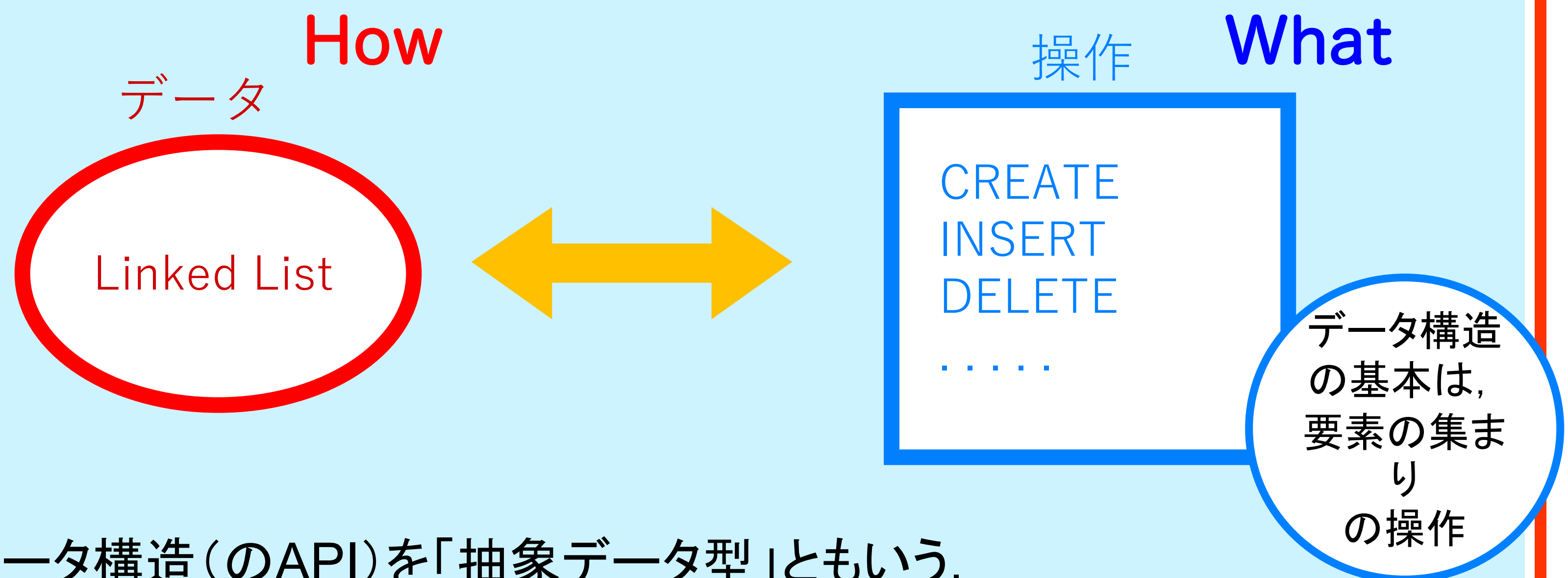
トピック2：優先度付き待ち行列（ヒープ）

- 高度なデータ構造として、「優先度付き待ち行列」（プライオリティー キュー）を学びます。具体的には、その実現方法として「ヒープ」データ構造を学びます。
- 「ヒープ」は、整数の集合を保持するもので、n個の要素を保持しているときに、要素の追加（insert演算）と最小要素の削除 （delete-min）を、O(log n)時間で行える優れたデータ構造です。
- ヒープは後で学ぶ平衡二分探索木（balanced binary tree）にアイディアが似ていますが、もっと単純です。（勉強に良い）。
- ヒープを用いた数の整列方法として、「ヒープソート」(heap sort)を紹介します。これは、与えられたn個の数をO(n log n)最悪時間で整列できる、もっとも高速な高速な整列アルゴリズムの一つです。

では、今から「ヒープ」を学びましょう。。。

抽象データ型 (Abstruct Data Type)とは？

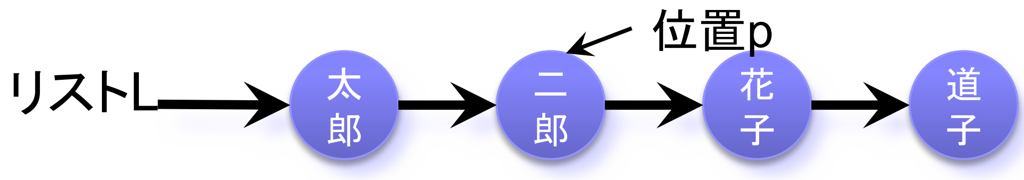
データ型を，それに適用される一組の操作で抽象的に定めたもの．



- データ構造（のAPI）を「抽象データ型」ともいう.
- データ構造には，「それは何か（What）」と「それをどのように実現するか？（How）」の二つの面がある.
- 現代的なプログラム言語やライブラリーはこの考え方に基づく.（例：C++，Java，Ruby, Python などなど）

基本

抽象データ型としての「リスト」に対する操作



- List L = create () : 空のリストを返す.

変更操作

- delete(L, p) : リストLの位置pの要素を**削除**する
- insert(L, p, x) : リストLの位置pの次に要素xを**挿入**する

探索操作

- search (L, x) : リストLに要素xが含まれてるかを1と0で返す

アクセス操作

- find(L, i) : リストLのi番目のセルの内容を返す（ランダムアクセス）
- last(L) : リストLの最後のセルの位置を返す
- next(L, p) : 位置pの１つ次のセルの位置を返す
- previous(L, p) : リストLにおいて、位置pの１つ前のセルの位置を返す

優先度付き待ち行列

- 挿入(insert)と最小値取り出し(deletemin)の二つの操作だけをもつデータ構造.
- 集合Aを表わし, その最小値を管理する

リスト,
キュー,
スタック
の仲間

二つの操作

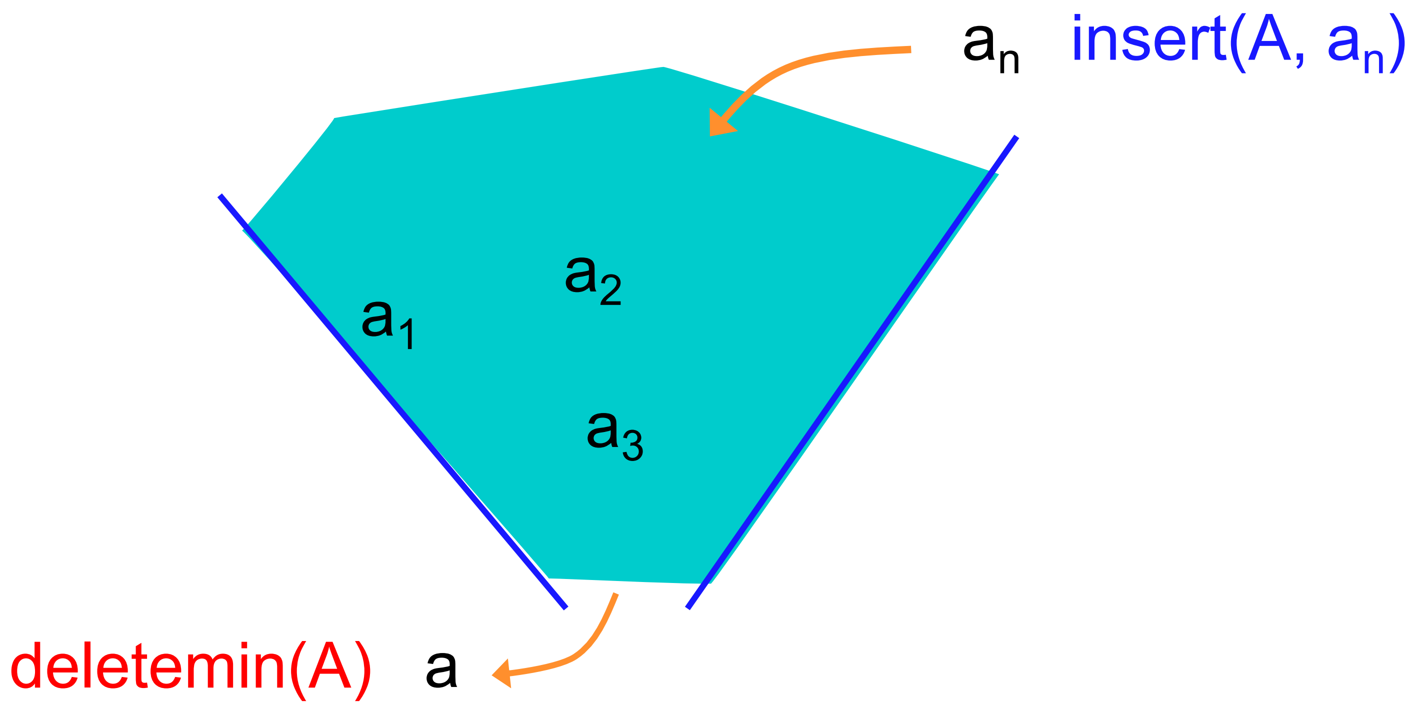
- insert(A, x) :　集合Aに要素xを挿入する
- deletemin(A) : 集合Aが空でないとき, 最小の要素 x を返し, 同時に x をAから削除する.

- 応用1 :「配列のヒープソート」
  - はじめに要素を全て挿入し, その後に最小要素を順に取り出す.
  - O(n log n)時間の最適ソートアルゴリズムの一つ
- 応用2 :「最小木の計算」
  - 重み付きの辺の集合を管理し, プログラムの各ステップで重み最小の辺を取り出す.

基本

優先度付き待ち行列

- 最小値を高速に取り出せる集合



Priority Queue の単純な実現方法

～ 連結リスト（linked list）を用いる方法

整列しない場合

DELETEMIN(A)

- O(n) 時間
- 先頭から操作し，最小の要素をみつける．

INSERT(x, A)

- O(1) 時間
- 先頭に挿入するだけ．

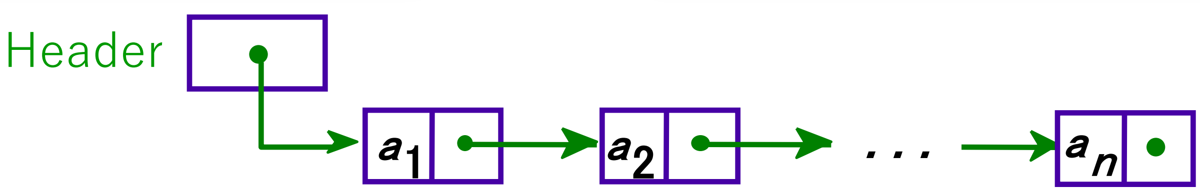
あらかじめ整列した場合

DELETEMIN(A)

- O(1) 時間
- 先頭の要素を除くだけ

INSERT(x, A)

- O(n) 時間
- 先頭から操作し，順序を保って挿入する．



疑問

DELETEMINとINSERTの
両方の演算を
もっと効率良く実現
できないか？

基本

ヒープ（heap）

平衡した２分木をつかって，順序集合を管理

効率よい操作を実現

DELETEMIN, INSERT の両方の操作を
O(log n) 時間で実現.

ヒープを用いて，Priority Queue を実現

ヒープ(heap)

英語:「山積み」という意味.

* 初心者注意

優先度付き待ち行列（ヒープ, heap）

集合X上の全順序（total order, 線形順序（linear order））とは
X上の要素間の2項関係`≦'で、次の性質をもつものをいう。

(1) x≦x for all x∊X (反射律, reflexivity)

(2) x≦y, y≦z ⇒ x≦z (推移律, transitivity)

(3) x≦y, y≦x ⇒ x=y (反対称律, anti-symmetry)

(4) x≦y or y≦x for all x,y∊X (比較可能性, comparability)

基本

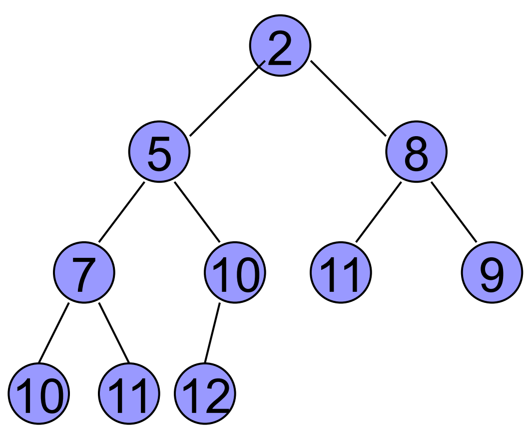
全順序`≦'が定義されている集合の要素を節点にもつ木で、
次のような「ヒープ条件」を満たす2分木（各節点の子の数が高々2つの木）を考える

ヒープ条件

任意の節点uに対して

uの親の要素≦uの要素

が成り立つ。



ヒープの定義：基本アイディア

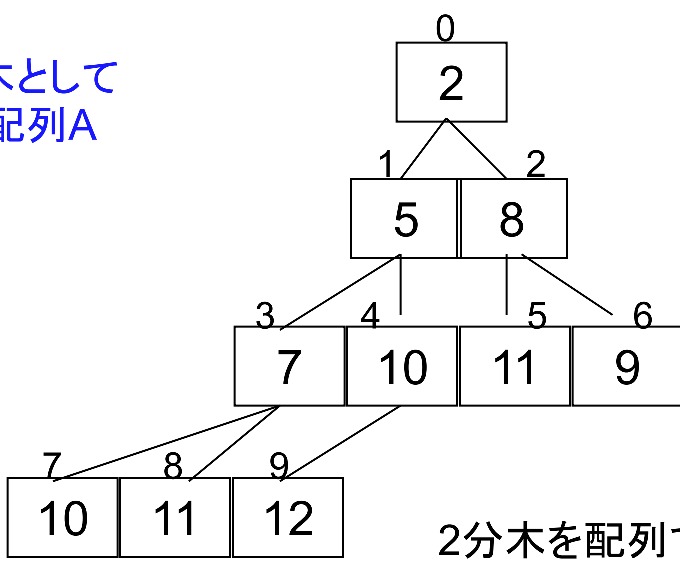
ヒープ（heap, 順位付きキュー（priority queue））とは

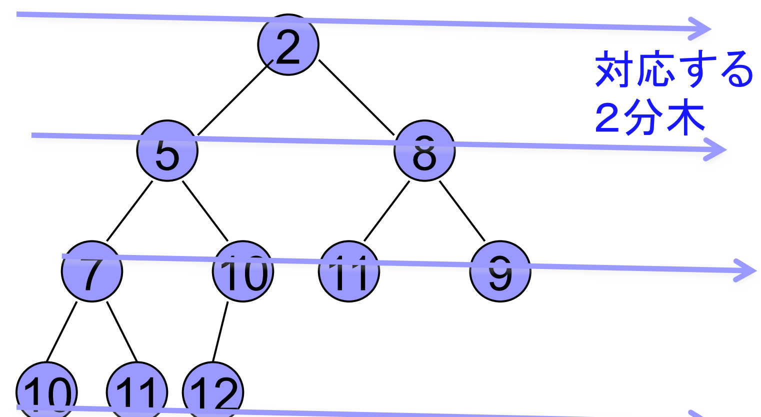
$A[i]$の親を$A[\lfloor (i-1)/2 \rfloor]$として定義される2分木がヒープ条件を満たす配列A

(例)

	0	1	2	3	4	5	6	7	8	9
配列A	2	5	8	7	10	11	9	10	11	12

2分木として
みた配列A



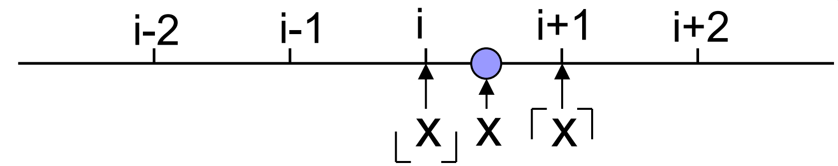


対応する
2分木

2分木を配列で表現する基本アイディア：2分木のノードを，同じ高さ毎（幅優先探索順）で切って，左から配列に詰め込む

基本

ヒープの定義(詳細)



実数xの切り捨て⌊x⌋と切り上げ⌈x⌉

ヒープ(heap, 順位付きキュー(priority queue))とは

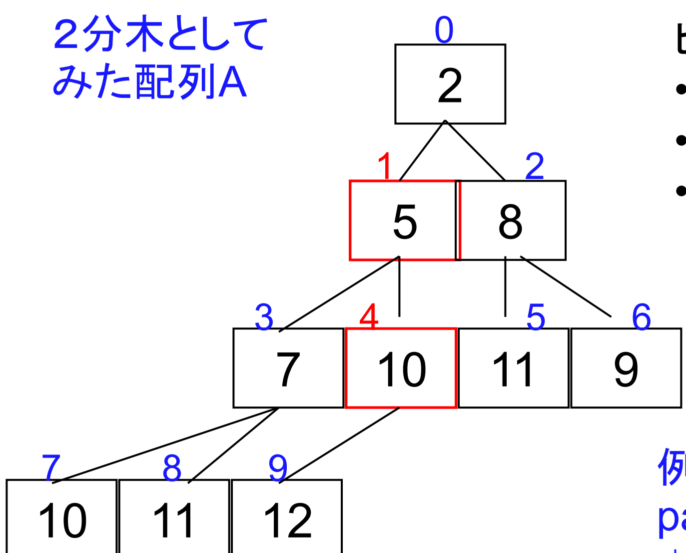
A[i]の親をA[⌊(i-1)/2⌋]として定義される2分木がヒープ条件を満たす配列A

(例)

	0	1	2	3	4	5	6	7	8	9
配列A	2	5	8	7	10	11	9	10	11	12

ヒント:二分木を、ポインタを使わずに、うまく配列で表すところがちょっとむずかしいかもしれない。でも、子供iから親を表す式を見て、下の木の例の上、よく考えればわかります。

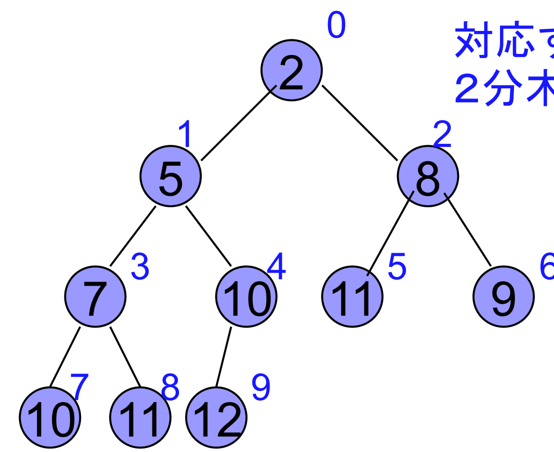
2分木としてみた配列A



ヒント:配列を2分木とみるには

- 配列の添字 =>ノード
- 配列の値=> ノードの値
- parent(i) = ⌊(i-1)/2⌋

対応する2分木



例)ノード4の親はノード1:
parent(4) = ⌊(4-1)/2⌋ =
⌊3/2⌋ = ⌊1.5⌋ = 1

ヒープの基本操作(最小要素の削除)

基本

DELETEMIN(A)　最小(根にある)要素の削除 (n:削除前の要素数)

Step 1 最小要素A[0]を出力. A[0]←A[n-1], i←0

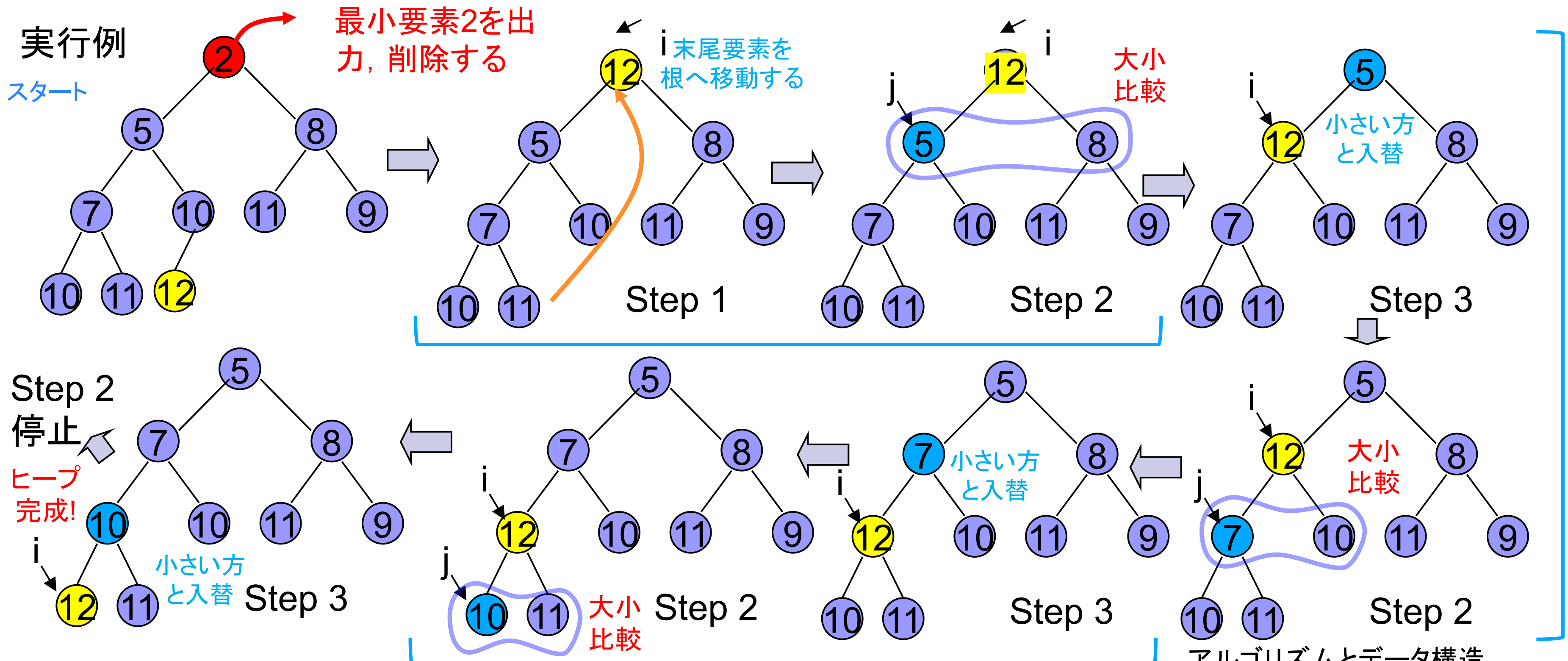
Step 2 2i+1≧n-1ならば停止。

そうでなければ$j \leftarrow \mathrm{argmin}_{k \in \{2i+1, 2i+2\}, k<n} A[k]$とする。

解説:A[2i+1]とA[2i+2]の小さい方をjとして選ぶ

Step 3 A[i]≦A[j]ならば停止。

そうでなければA[i]とA[j]の中身を入れ替え、i←jとしてStep 2へ



ヒープの基本操作（要素の追加）

INSERT(x,A)　要素の追加 (n:追加前の要素数)

基本

Step 1 A[n]←x, i←n

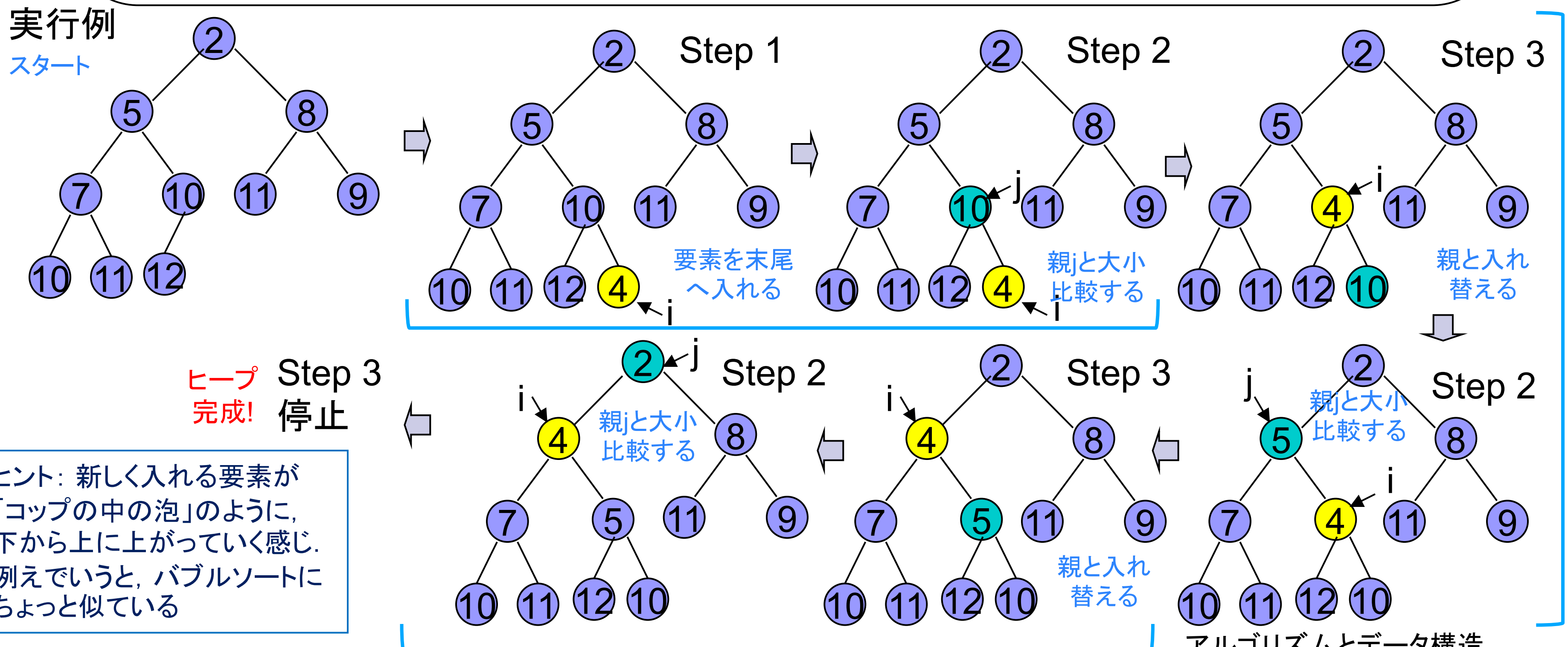
Step 2 i=0ならば停止。

そうでなければj← ⌊(i-1)/2⌋とする。

Step 3 A[i]≧A[j]ならば停止。

そうでなければA[i]とA[j]の中身を入れ替え、i←jとしてStep 2へ

実行例



ヒント：新しく入れる要素が「コップの中の泡」のように，下から上に上がっていく感じ．例えでいうと，バブルソートにちょっと似ている

＊初心者注意

ヒープの基本操作の時間計算量

基本

DELETEMIN(A), INSERT(x,A)ともに、各Stepは定数時間で実行可能

⇩

Step 2とStep 3の間のループの回数のオーダーで実行可能

1回の繰り返し毎にDELETEMINはiの位置が1つずつ深くなっていく
INSERTは　　iの位置が1つずつ浅くなっていく

⇩

最悪、木の高さの回数だけループする

証明してみよう！

要素数nのヒープを2分木で表現した場合、木の高さは $\lfloor \log_2 n \rfloor$ である。

（証明のあらすじ） ヒープの配列での表現方法から，平衡係数（前回の平衡探索木）が常に1以下なので，木の高さは$O(\log_2 n)$

DELETEMIN(A)とINSERT(x,A)の最悪時間計算量はO(log n)

応用：ヒープソート

- 整列問題のO(n log n)時間の最適解法の一つ

整列問題

入力：長さ n の配列 A

6	17	3	1	8	2

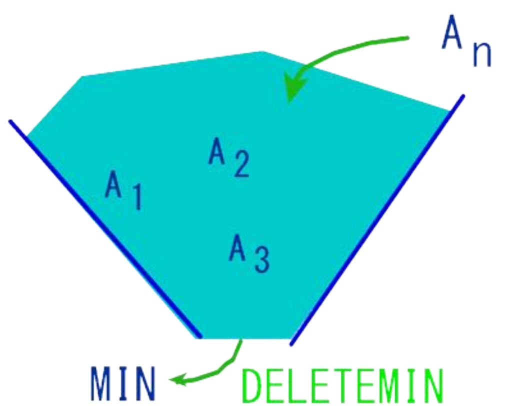
出力：A の要素を昇順にならべかえた配列

1	2	3	6	8	17

コメント：ヒープソートは，授業後半（整列アルゴリズム）でもう一度詳しく学びjます．

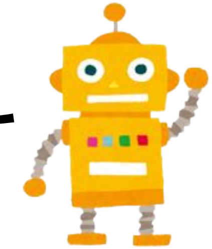
応用:ヒープソート

- 考え方
  - ヒープ(優先度付き待ち行列)を使って，長さ配列Aの要素をO(n log n)時間で整列する
- 手順
  - 空のヒープへ要素A[0], …, A[n-1] を順にinsertする.
  - Deleteminをn回適用し，最小要素を順に取り出して，出力配列へ左詰めで入れる



コメント: ヒープソートは，授業後半(整列アルゴリズム)でもう一度詳しく学びjます.

アルゴリズム道場:今日のおまけ



次ができたら、学んだアルゴリズムが理解できた証拠です

1. 友達に、「それは、だいたいこんなアルゴリズムだよ」と、(図をかいたりして)説明できる

2. 自分でプログラムが書けそうな気がする(実際にかくのは時間がかかるが、時間さえあればできると思える)

逆に言うと、上のどちらかできるような気がすれば、アルゴリズムの細かなところは忘れても大丈夫！

どうかな？

アルゴリズムの説明は、同じ本で理解しても、
人によって全然違うし、違っていて良いのです。
それが新しいアルゴリズムの発明の第一歩です。

(今までになくて良い)

優先度つき待ち行列(ヒープ)

- 今日の内容:
  - 抽象データ型:優先度付き待ち行列(Priority queue)
    - 挿入(Insert)と最小値除去(deletemin)をもつキューの一種.
  - 実装方法(ヒープ). 応用としてヒープソート.
- ポイント
  - O(log n)最悪計算時間のデータ構造
  - 平衡2分木
  - 抽象データ型とその実装