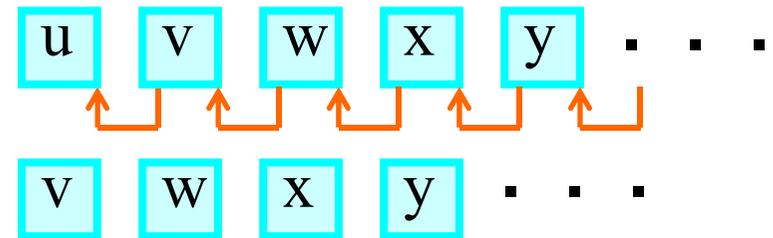


マージソート (merge sort)

前シフト : データ列の 2 番目以降の順番を, 1 つずつ
前進させる.



前シフトの実現法 :

実際にデータを動かす.

先頭要素を指すポインタの値を, 1 つ増やす.

マージ (併合) :

データ列 a の後ろに, データ列 b を連結する.

(1) b が空になるまで繰り返す.

(1-1) a の末尾の直後に, b の先頭要素を加える.

(1-2) b を前シフトする.

`merge_sort (a, sorted_a, n)`

n 個の要素からなる列 (リスト) a の整列結果を,
列 `sorted_a` に作る.

マージソート :

- ・ 要素の列 a を, 前半, 後半の, 2つの列に分け,
それぞれを整列する.
- ・ 2つの列の整列結果から, 整列順に要素を取り出し,
全体で整列した列 `sorted_a` を作る.

- (1) n が 1 であれば,
列 a の先頭の要素を, sorted_a の先頭要素に代入し, 終り. (← 再帰の終了条件)
 - (2) a を, $n/2$ 個の x と, $n-(n/2)$ 個の y に分ける.
 - (3) $\text{merge_sort}(x, \text{sorted_x}, n/2)$
と,
 $\text{merge_sort}(y, \text{sorted_y}, n-(n/2))$
を実行する.
 - (4) 繰り返す.
 - (4-1) sorted_x が空であれば,
 sorted_a に sorted_y をマージし, (4) の終り.
 - (4-2) sorted_y が空であれば,
 sorted_a に sorted_x をマージし, (4) の終り.
- (continue)

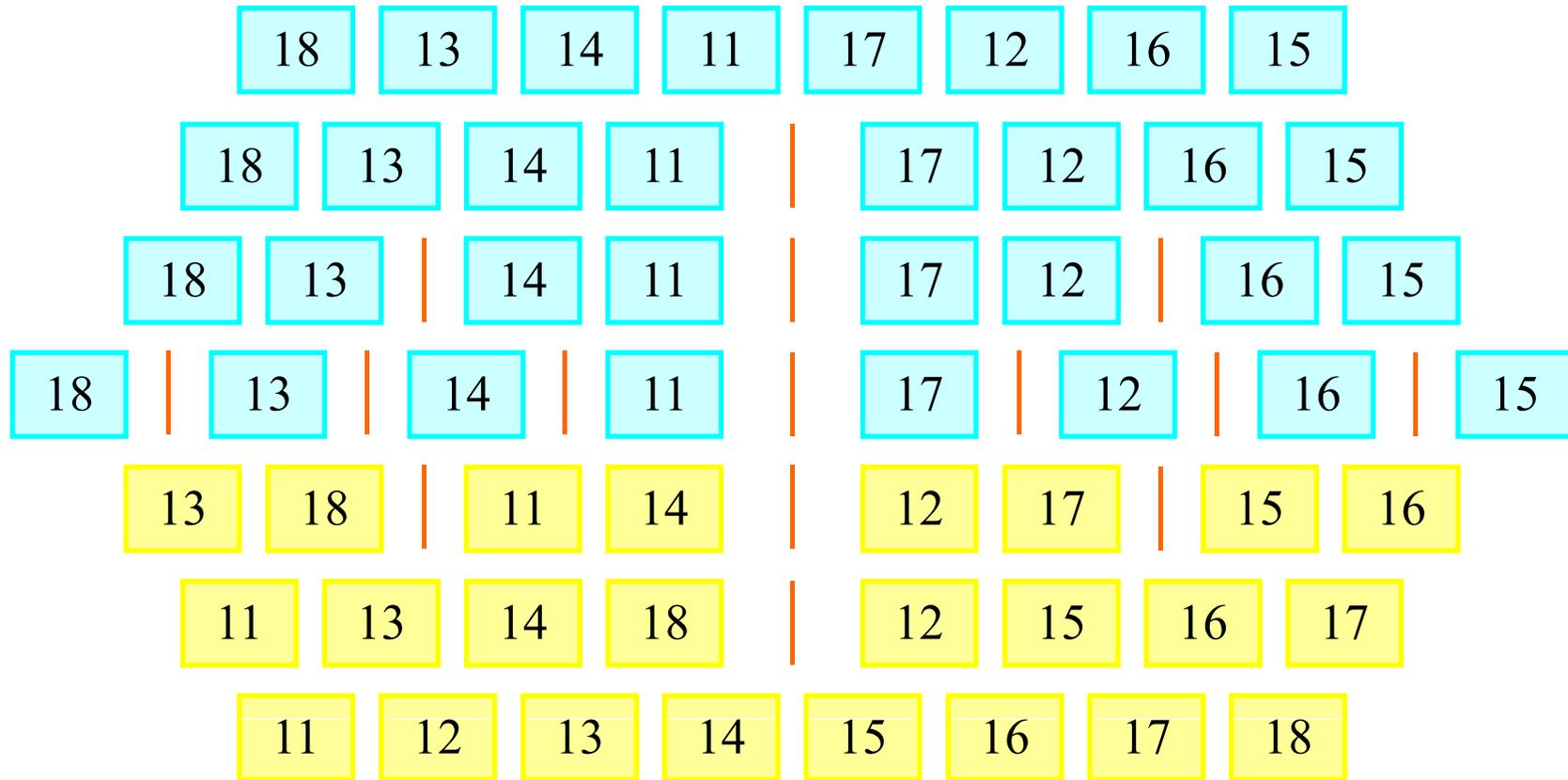
(4-3) (sorted_x の先頭要素キー)
≤ (sorted_y の先頭要素キー)
であれば,

sorted_x_a の末尾直後に,
sorted_x の先頭要素を加えて,
sorted_x を前シフトする.

さもなければ,

sorted_a の末尾直後に,
sorted_y の先頭要素を加えて,
sorted_y を前シフトする.

マージソートの実行



クイックソート (quick sort)

分割： ある要素（ピボット（軸）要素）のキー値をピボットとして、ピボットに対する \leq , $>$ で、要素の列を、前側、後側の2つの列に分割する。

クイックソート：

- ・ 前側の列において、ピボット要素を末尾に置く。
- ・ 前側の列、後側の列を、再帰的に分割・整列する。

`quick_sort (a, n)`

要素の列 `a` の、0番から `n-1` 番を整列する、

`quick_sort0 (a, 0, n-1)`

を実行する。

quick_sort0 (a, i, j)

要素の列 a の, i 番から j 番を整列する.

- (1) i 番が j 番以降であれば, 終り.
- (2) i 番から j 番の要素の列の中から, ピボット要素を選ぶ.
- (3) a のうち, i 番から j 番の要素値を,
 (要素キー \leq ピボット) となる前側,
 (要素キー $>$ ピボット) となる後側
 に分割し, かつ, 前側の末尾要素の位置 k を決定し,
 そこに, ピボット要素を設置する.
- (4) quick_sort0 (a, i, k-1)
 および
 quick_sort0 (a, k+1, j)
 を実行する.

ピボットの選び方：

要素列中の， 整列対象部分に含まれる要素において，

- ・ 先頭の要素のキーとする.
- ・ 数個の要素のキーの，
平均に近い値とする.
中間値とする.
- ・ 要素列の 1 番と 2 番の， 大きい方を選ぶ.

整列対象部分の分割と、ピボット要素の前側列末尾への設置：

(1) 整列対象部分で、ピボット要素と先頭要素を入れ替える。

(2) 整列対象部分で、

先頭要素の次の要素からは、末尾向きに走査、

末尾要素からは、先頭向きに走査

をはじめ、繰り返す。

(2-1) 末尾向き走査で、

(要素キー) > (ピボット)

となる要素の位置 dn を得る。

対象部分の末尾に達したら、末尾要素の位置を

dn とする。

(continue)

(2-2) 先頭向き走査で,

$$(\text{要素値キー}) \leq (\text{ピボット})$$

となる要素の位置 up を得る.

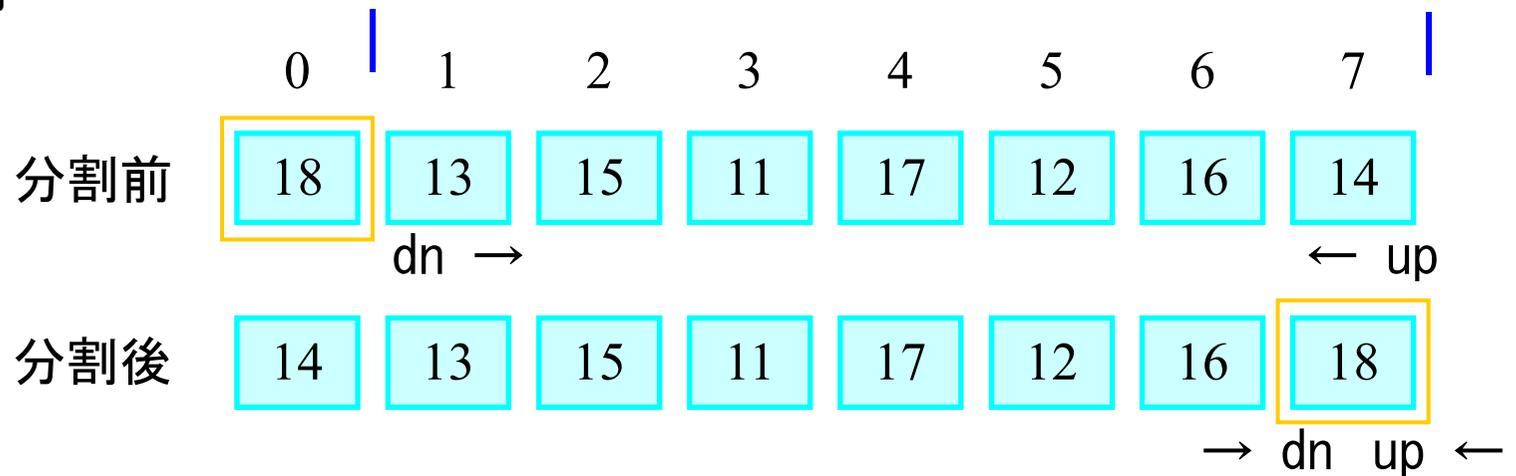
対象部分の先頭に達したら, 先頭要素の位置を up とする.

(2-3) $up \leq dn$ となり, 両側からの走査が交差したら, 先頭要素 (ピボット要素) と位置 up の要素を入れ替え, up を前側の末尾位置として, (2) の終り.

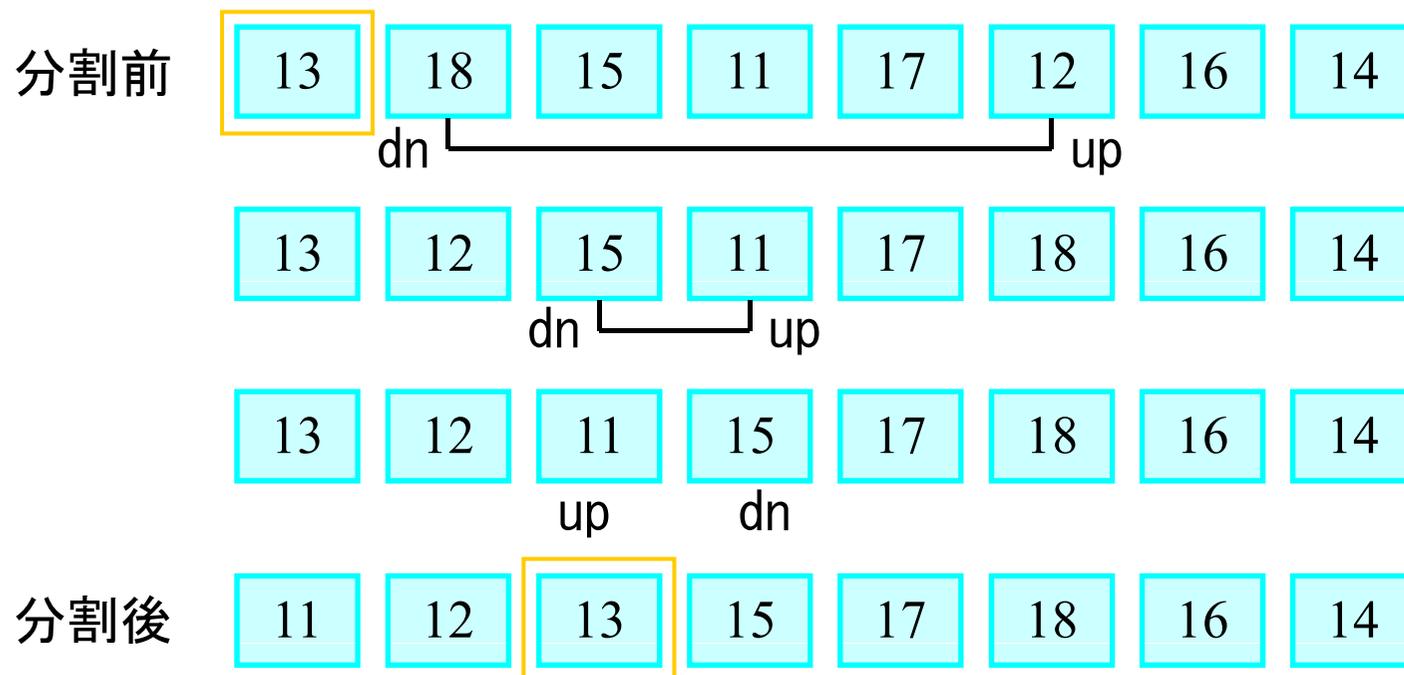
(2-4) 位置 dn の要素と, 位置 up の要素を入れ替える.

分割の実行

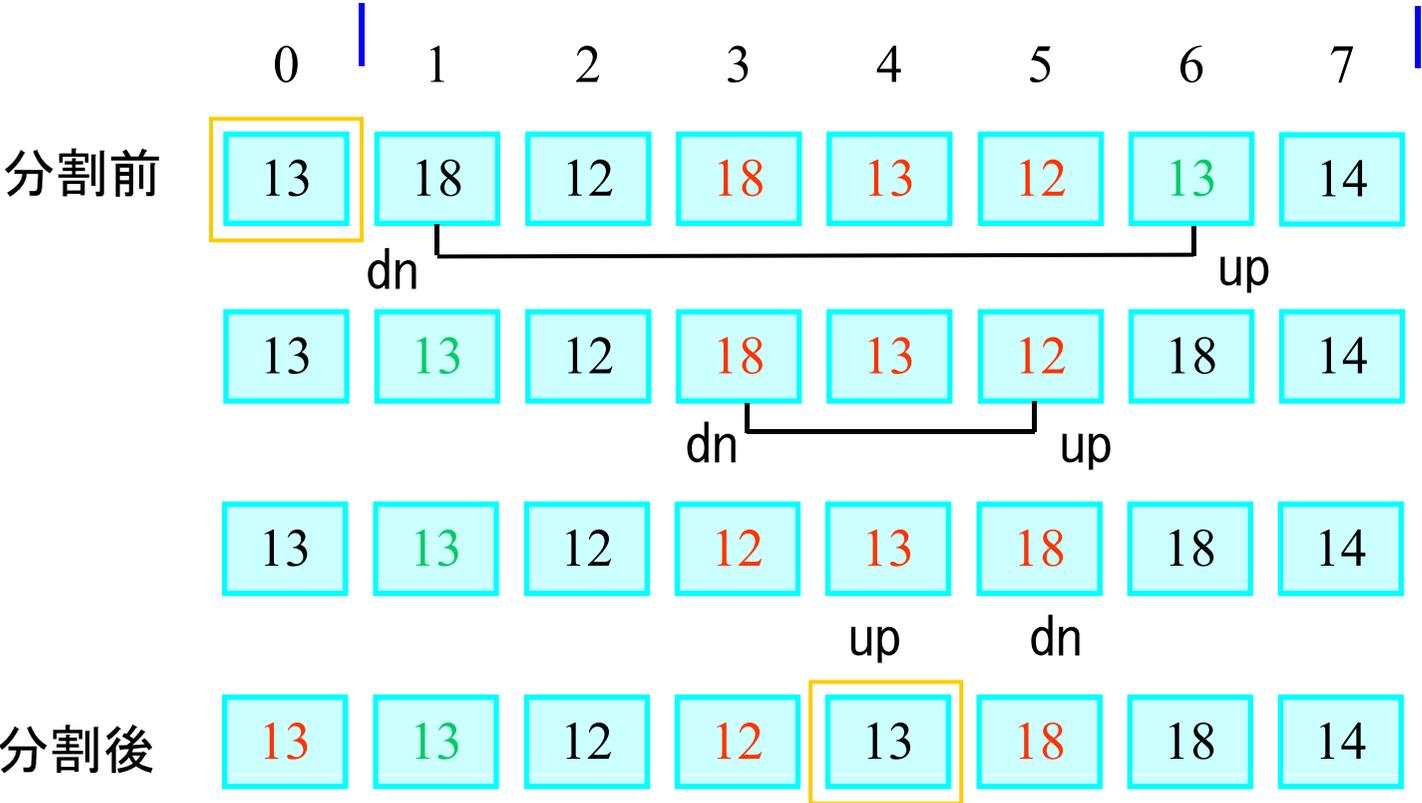
例 (1)



例 (2)



例（3） 重複したキーが存在する場合



クイックソートの実行

13 18 15 11 17 12 16 14

11 12 | 13 | 15 17 18 16 14

11 | 12 | 13 | 14 | 15 | 18 16 17

11 | 12 | 13 | 14 | 15 | 17 16 | 18

11 | 12 | 13 | 14 | 15 | 16 | 17 | 18



前側末尾位置に設定された、ピボット要素

ヒープソート (heap Sort)

要素列を, ヒープで表された完全2分木とする.

(ヒープは, 節点0からでなく1から要素を入れた)

完全2分木を, 半順序木に変換できる.

最小要素を取り出すことができる.

heap_sort (a, n)

0番から $n-1$ 番の, n 個の要素をもつ要素列 a を
整列する.

- (1) 要素列 a の各要素を, 1つ後ろにずらす.
(番号 1 からの要素列とする)
- (2) 要素数 n の完全2分木 a を, 半順序木に変換する.
- (3) 節点数 n の, 完全半順序2分木 a を整列する,
heap_sort0 (a, n)
を実行する.
- (4) 要素列 a の各データを, 1つ前にずらす.

heap_sort0 (a, i)

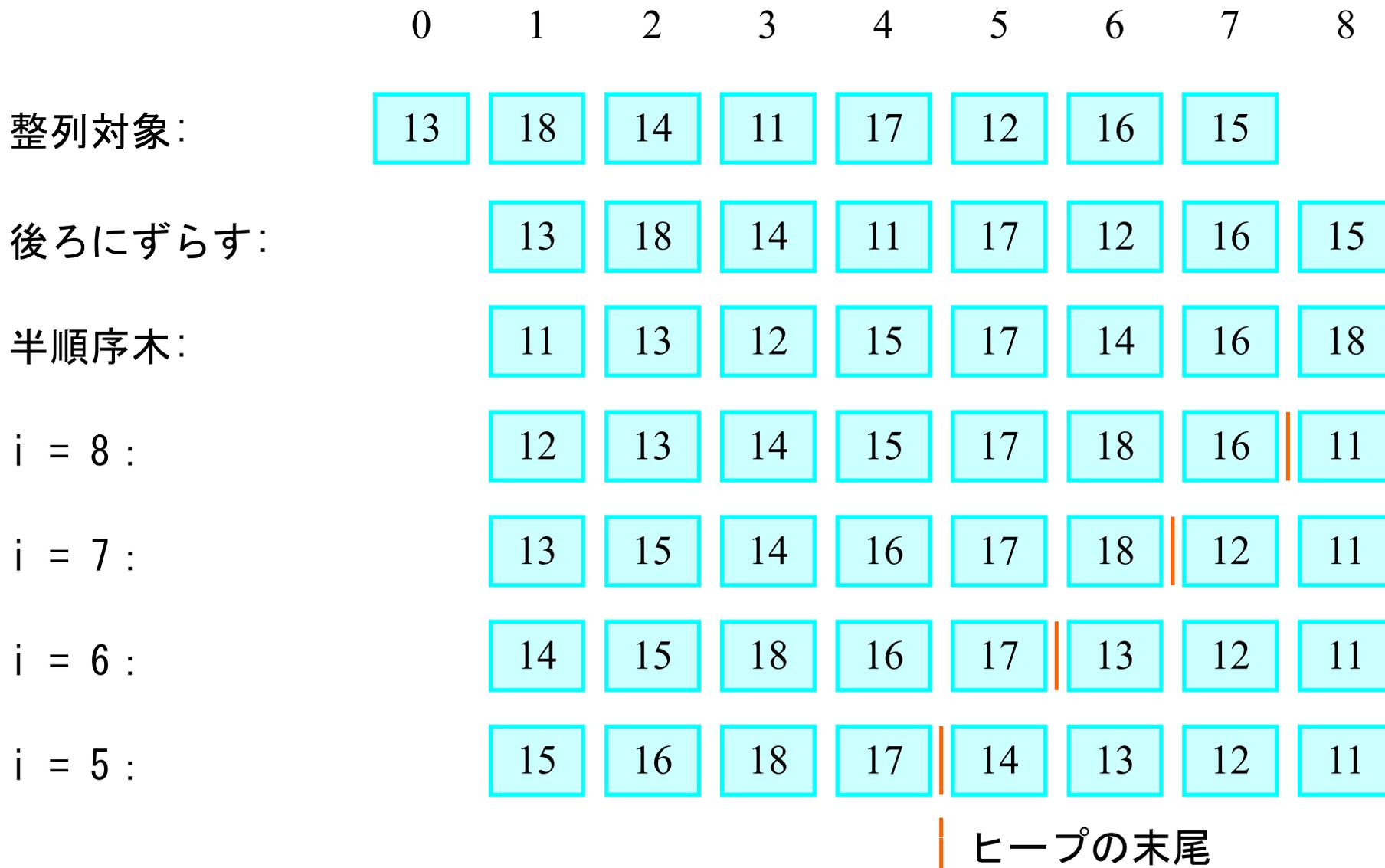
完全半順序 2 分木 a の, i 番から 1 番の要素を整列する.

整列したデータを蓄える場所 →

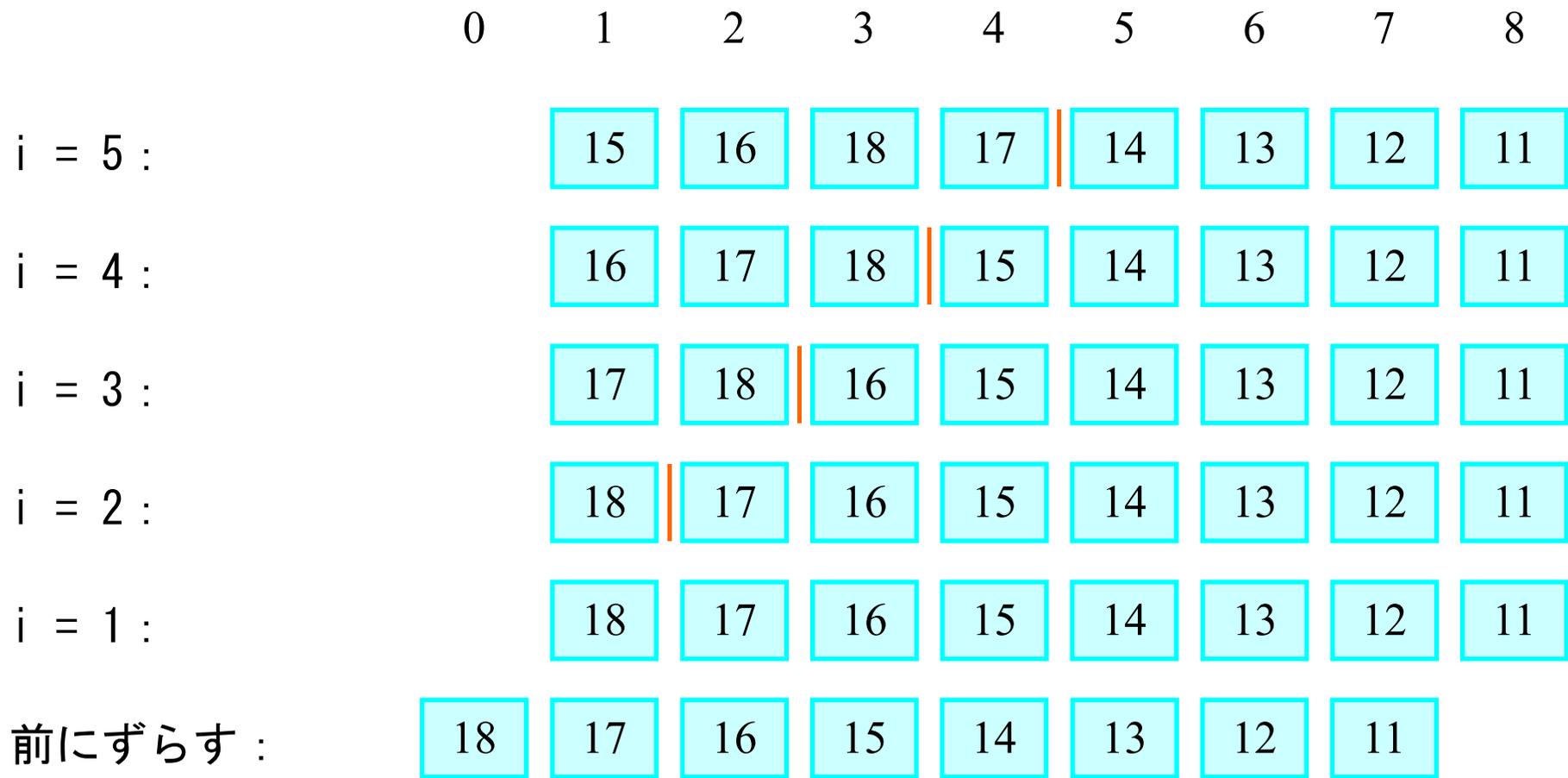
i 番以降である, ヒープの末尾より後ろを使う.

- (1) i が 1 であれば, 終り.
- (2) a の 1 番の要素を, 変数 min に代入する.
- (3) 節点数 i の完全半順序 2 分木 a の最小要素を削除する (完全半順序 2 分木からの最小要素削除のアルゴリズムを適用) .
- (4) min の値を, i 番の要素に代入する.
- (5) heap_sort0 (a, i-1) を実行する.

ヒープソートの実行 (1 / 2)



ヒープソートの実行 (2 / 2)



出来上がるのは \leq の逆順 \rightarrow 半順序木を定義しなおす.
得られた結果を逆順にする.