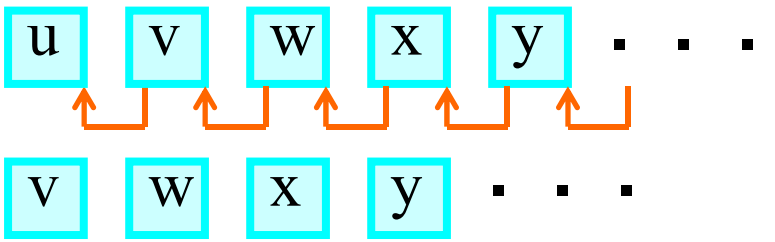
# マージソート　（merge sort）

前シフト：　データ列の２番目以降の順番を，１つづつ
　　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 個の前半 x と, n-(n/2) 個の後半 y に分る.
(3) merge_sort (x, sorted_x, n/2)
　　と,
　merge_sort (y, 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) sortex_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 の先頭要素の値）

≦（sorted_y の先頭要素の値）

であれば,

sortex_a の末尾直後に,

sorted_x の先頭要素を加えて,

soretd_x を前シフトする.

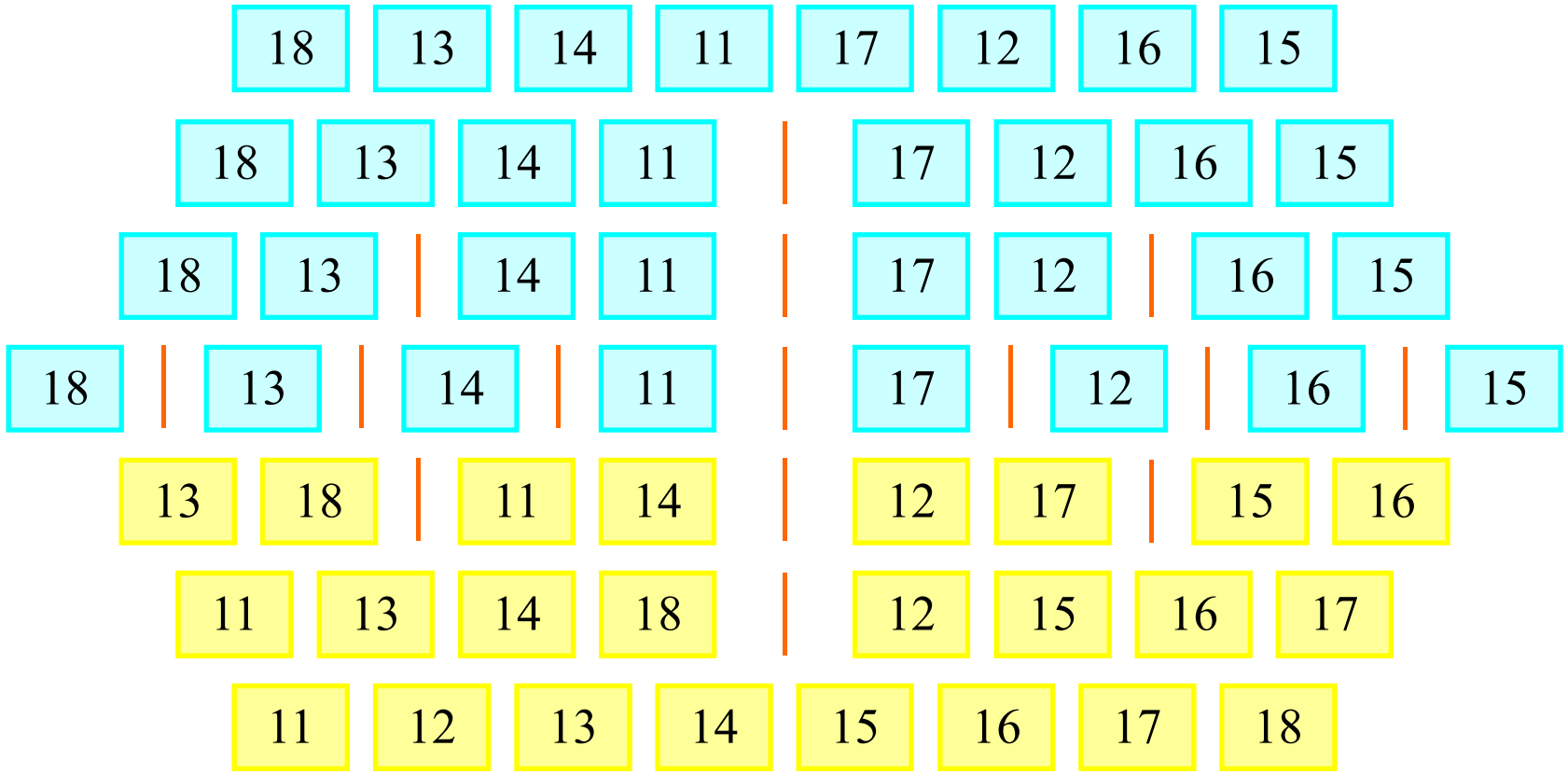
さもなければ,

sorted_a の末尾直後に,

sorted_y の先頭要素を加えて,

soretd_y を前シフトする.

# マージソートの実行


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

# クイックソート（quick sort）

分割：　ある要素（ピボット（軸）要素）の値をピボットとして，ピボットに対する ≦，＞ で，要素の列を，前側，後側の２つの列に分割し，前側の列において，ピボット要素を末尾に置く．

クイックソート：

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 番の要素を,
（要素の値 ≦ ピボット）となる前側,
（要素の値 ＞ 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，

- 先頭の要素の値とする．
- 数個の要素の値の，
  - 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

を始め，繰り返す.

(2-1) 末尾向き走査で，

（要素の値）＞（ピボット）

となる要素の位置 dn を得る.

対象部分の末尾 j 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,

（要素の値）≦（ピボット）

となる要素の位置 up を得る.

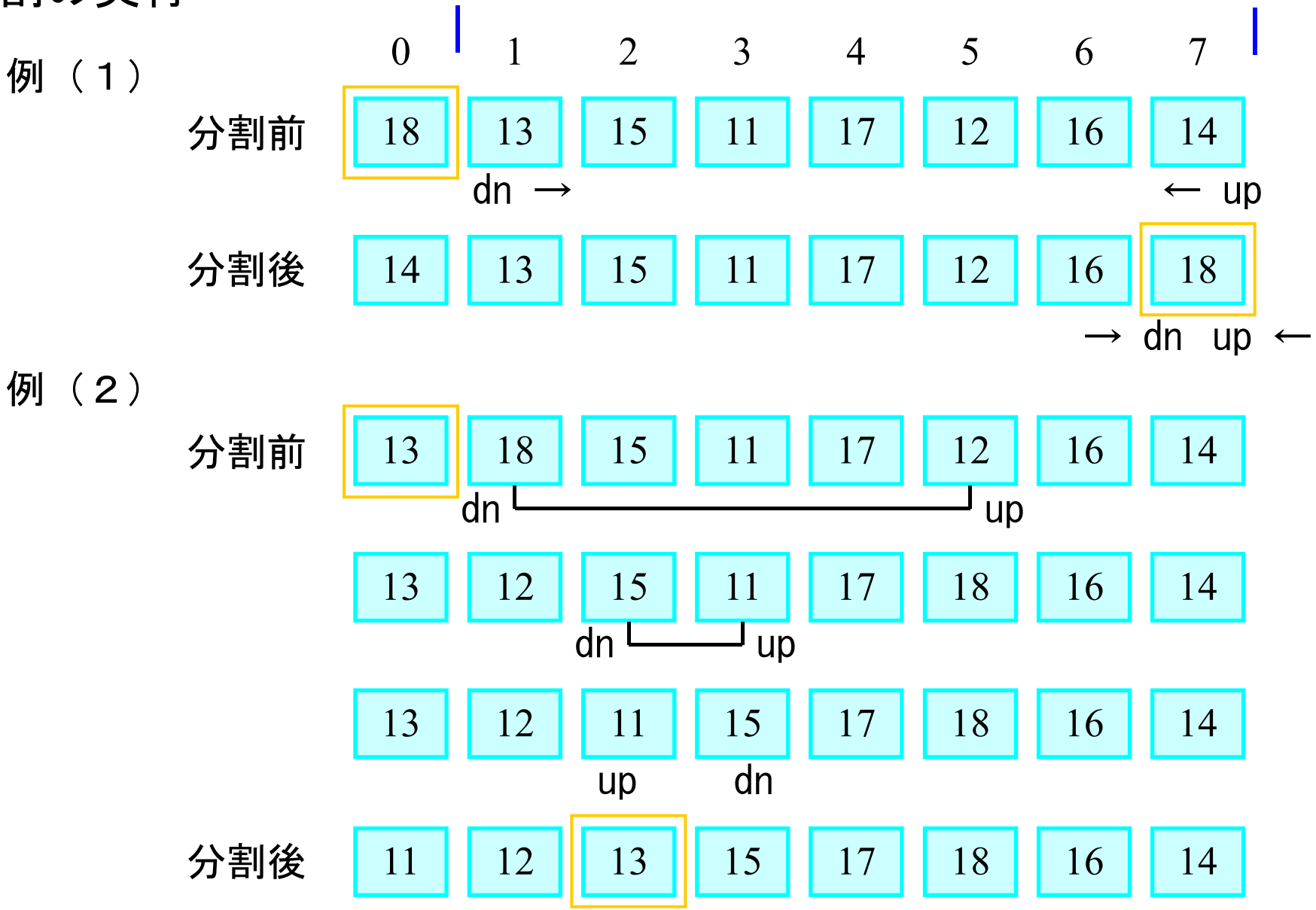
対象部分の先頭 i に達したら，先頭要素の位置を up とする.

(2-3) up ≦ dn となり，両側からの走査が交差したら,

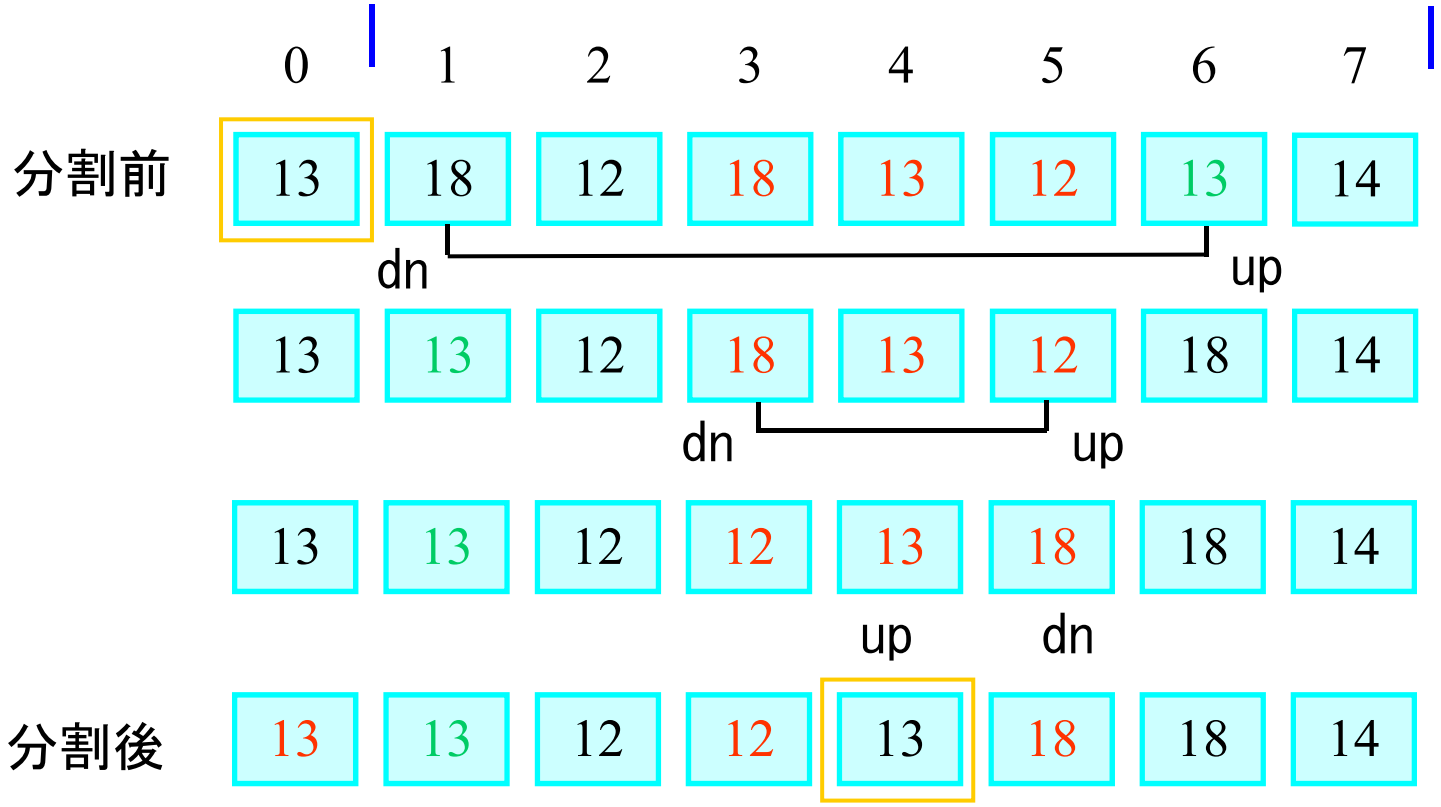
- up を前側の末尾位置とする,
- 先頭要素（ピボット要素）と位置 up の要素を入れ替える,
- (2) の終り.

(2-4) 位置 dn の要素と，位置 up の要素を入れ替える.

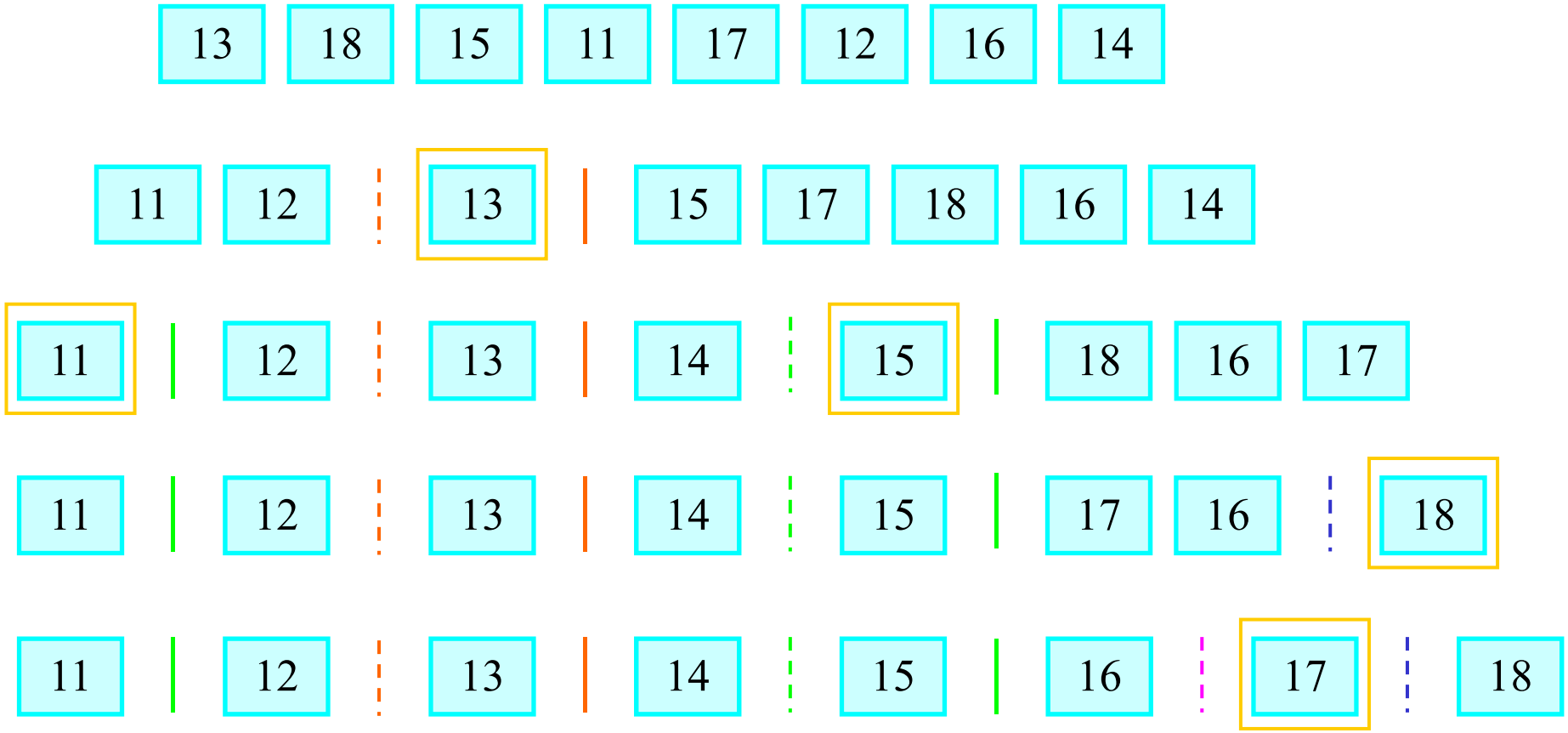
# 分割の実行


例（１）
0 1 2 3 4 5 6 7
分割前
18 13 15 11 17 12 16 14
dn →
← up
分割後
14 13 15 11 17 12 16 18
→ dn up ←
例（２）
分割前
13 18 15 11 17 12 16 14
dn
up
13 12 15 11 17 18 16 14
dn
up
13 12 11 15 17 18 16 14
up
dn
分割後
11 12 13 15 17 18 16 14

例（３）　重複した値が存在する場合



# 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）

要素列を完全２分木で表す
（木は，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．

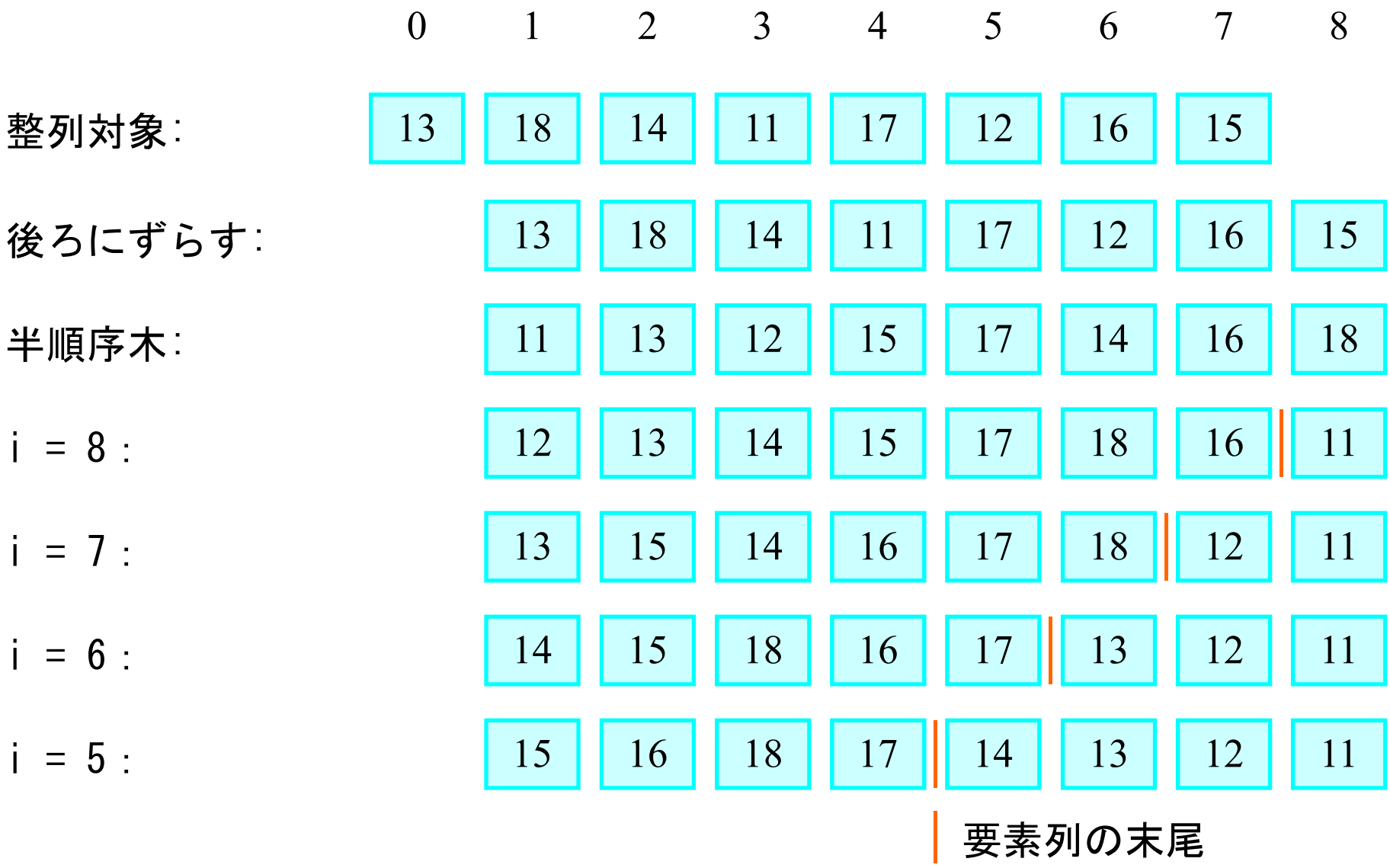
heap_sort0 (a, i)

完全半順序２分木 a の，i 番から１番の要素を整列する.

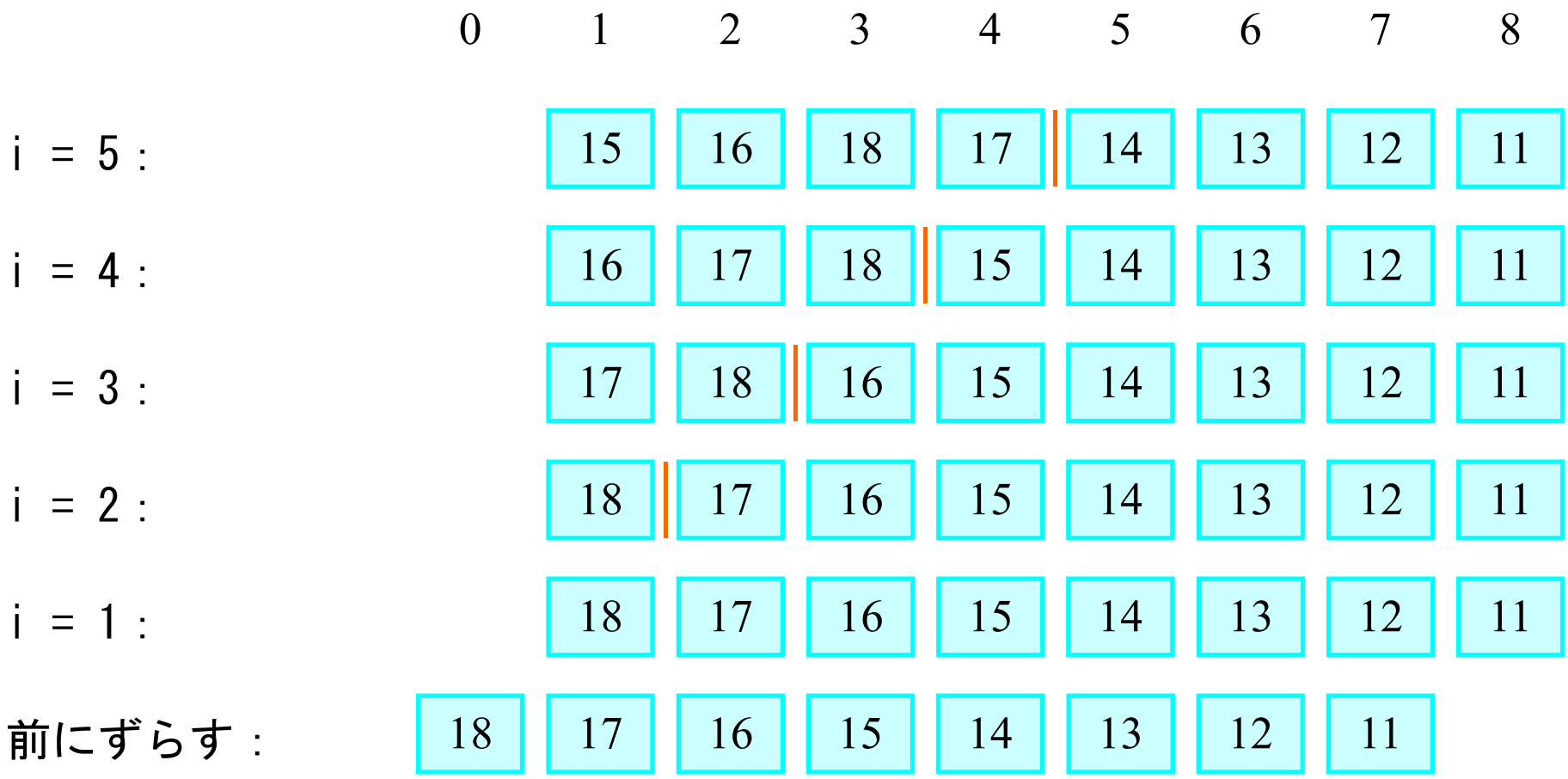
整列したデータを蓄える場所　→
i 番以降である，要素列の末尾より後ろを使う.

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




出来るのは ≦ の逆順 → 木の半順序関係 ≦ を再定義する.
得られた結果を逆順にする.