離散構造とアルゴリズム
(2-3) 整列アルゴリズム

高橋篤司
東京工業大学 工学院 情報通信系

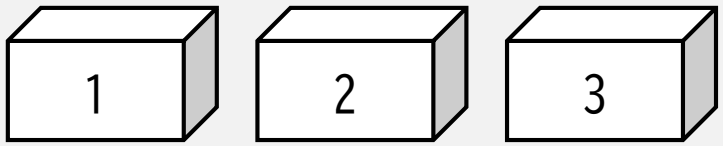
2-3（1）整列問題

■ 整列問題
- 入力：系列 $X = (x_1, x_2, \ldots, x_n)$　但し $(x_1, x_2, \ldots, x_n \in \mathbb{Z})$
- 質問：Xを大きさの昇順に並べかえよ

■ 検討の対象とするアルゴリズム
➢ 二つの数の大きさの比較を繰り返して解く

✓ 対象外の例
- ビンソート(bin sort, bucket sort)
  - 数の入るビン(容器，バケツ)を用意し，数を振り分ける

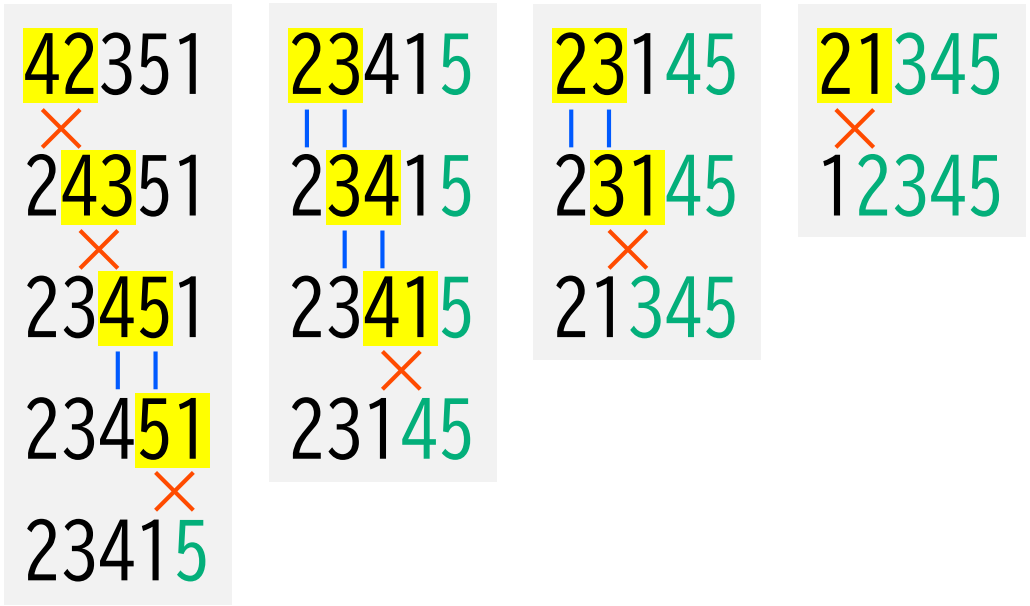
3 1 3 2 1 2 3

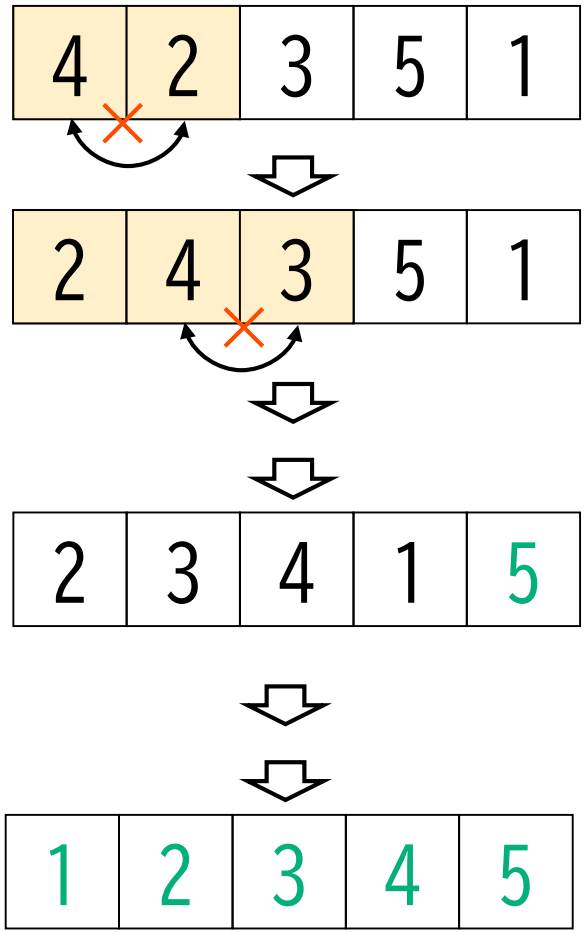


ビンが用意できれば線形時間
大規模データの場合には階層化

バブルソート (bubble sort)

- 配列上の隣接データを比較
  - 整列していない（大小が合致していない）ならば交換
- 比較回数 $\frac{n(n-1)}{2} = O(n^2)$
- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,2,3,5,1)$

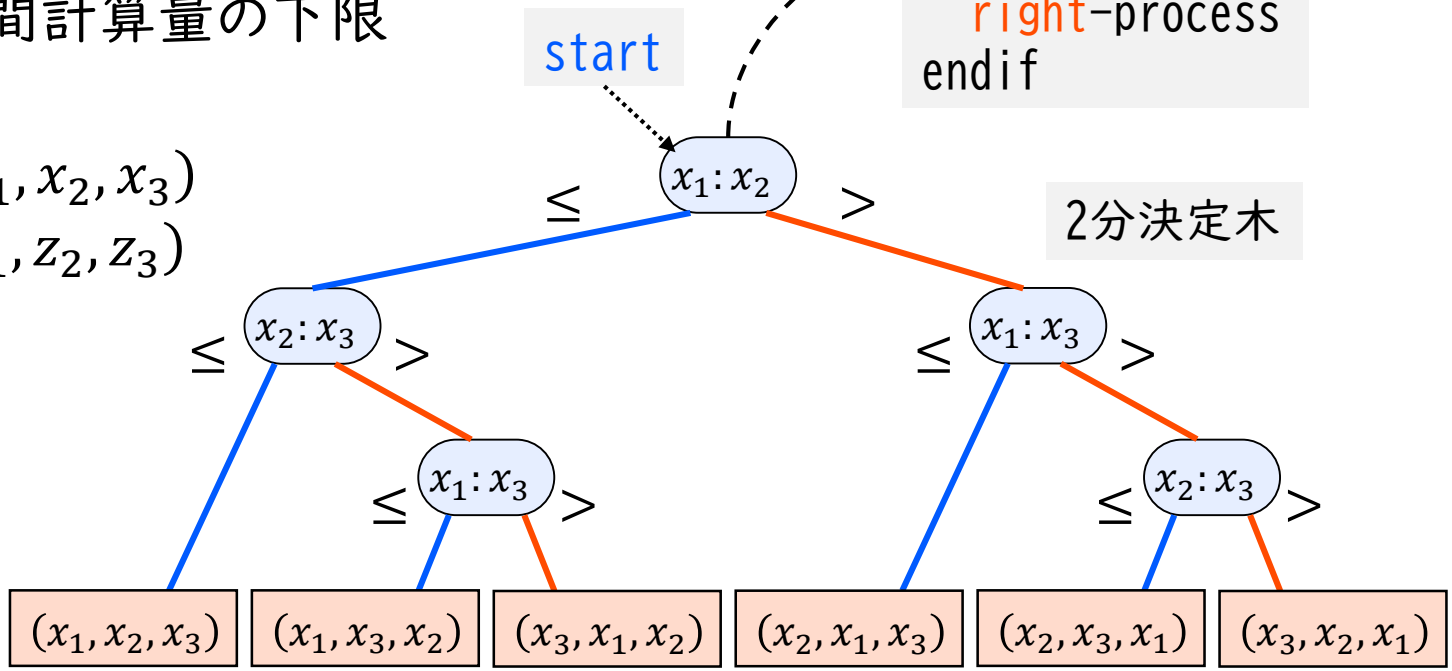




2分決定木 (binary decision tree)

- 2分決定木
  - 条件分岐(if-then-else)の様子を表現
    - アルゴリズムの動作
      - 根から葉に向かう経路
  - 内点＝条件分岐（出力）
  - 葉：出力
  - 高さ：時間計算量の下限
- 例
  - 入力：(x_1, x_2, x_3)
  - 出力：(z_1, z_2, z_3)

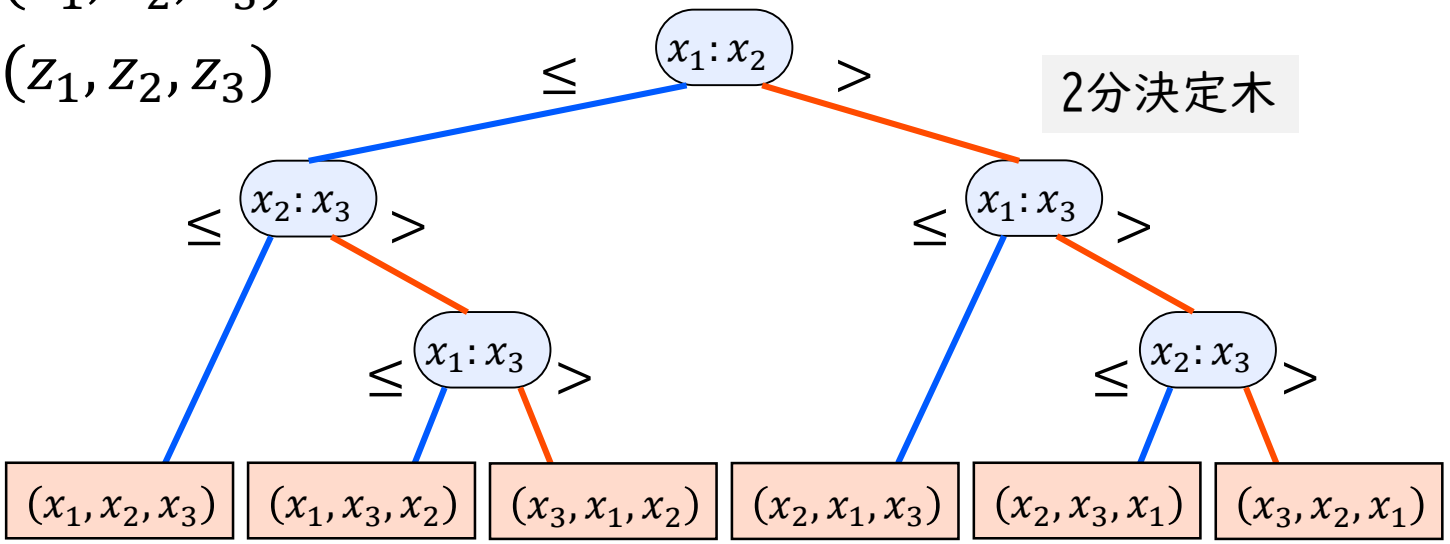
```
if(x1 ≤ x2) then
  left-process
else
  right-process
endif
```



2分決定木 (binary decision tree)

■ 例:3要素整列アルゴリズム

- 入力:(x_1, x_2, x_3)
- 出力:(z_1, z_2, z_3)



2分決定木

入力:$(x_1, x_2, x_3) = (4,9,5)$
$x_1 \leq x_2$ $(4 \leq 9)$
$x_2 > x_3$ $(9 > 5)$
$x_1 \leq x_3$ $(4 \leq 5)$
出力:$(x_1, x_3, x_2) = (4,5,9)$

入力:$(x_1, x_2, x_3) = (6,5,8)$
$x_1 > x_2$ $(6 > 5)$
$x_1 \leq x_3$ $(6 \leq 8)$
出力:$(x_2, x_1, x_3) = (5,6,8)$

バブルソート (bubble sort)

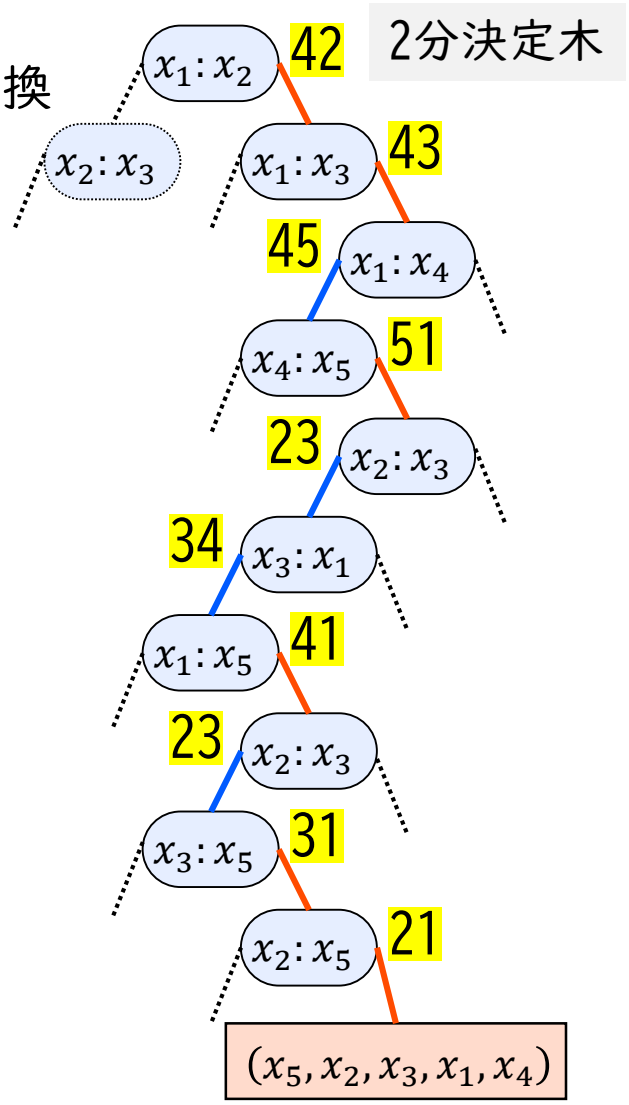
- 配列上の隣接データを比較
  - 整列していない（大小が合致していない）ならば交換
- 比較回数 $\frac{n(n-1)}{2} = O(n^2)$
- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,2,3,5,1)$

42351 → 24351 → 23451 → 23451 → 23415

23415 → 23415 → 23415 → 23145

23145 → 23145 → 21345

21345 → 12345



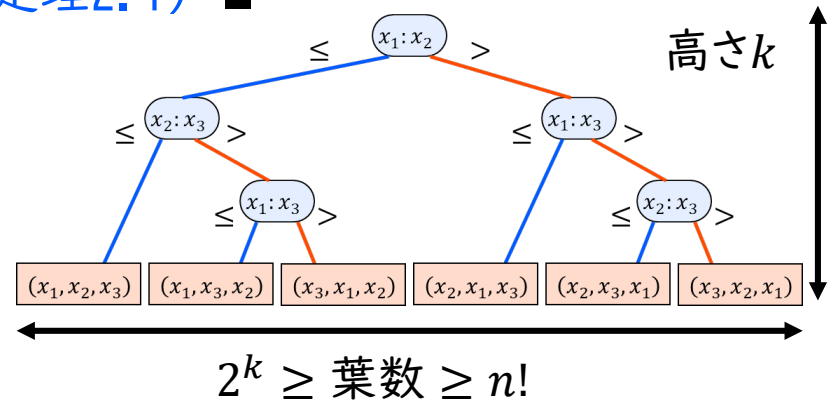
整列：時間計算量の下限

■ 定理（Theorem 2.6）

- 任意の整列アルゴリズム（2数比較）の時間計算量は$\Omega(n \log n)$

□ 証明 ：

- nデータを整列するアルゴリズムの2分決定木
  - 時間計算量＝高さk $\Longrightarrow$ 葉数は高々2^k（∵[1-2]定理1.9）
- 入力列の置換（順列）の総数$n!$
  - 各置換は少なくとも1回は葉に含まれる
  - $2^k \geq n!$
  - $k \geq \log_2(n!) = \Theta(n \log n)$（∵[2-1]定理2.1） ■

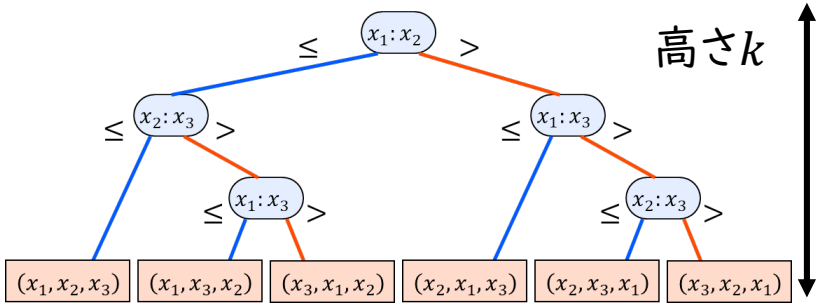


整列:時間計算量の下限

■ 定理 (Theorem 2.6)
- 任意の整列アルゴリズム(2数比較)の時間計算量は$\Omega(n \log n)$

■ バブルソート
- 時間計算量$O(n^2)$
  - ✓ 下限$\Omega(n \log n)$を達成していない

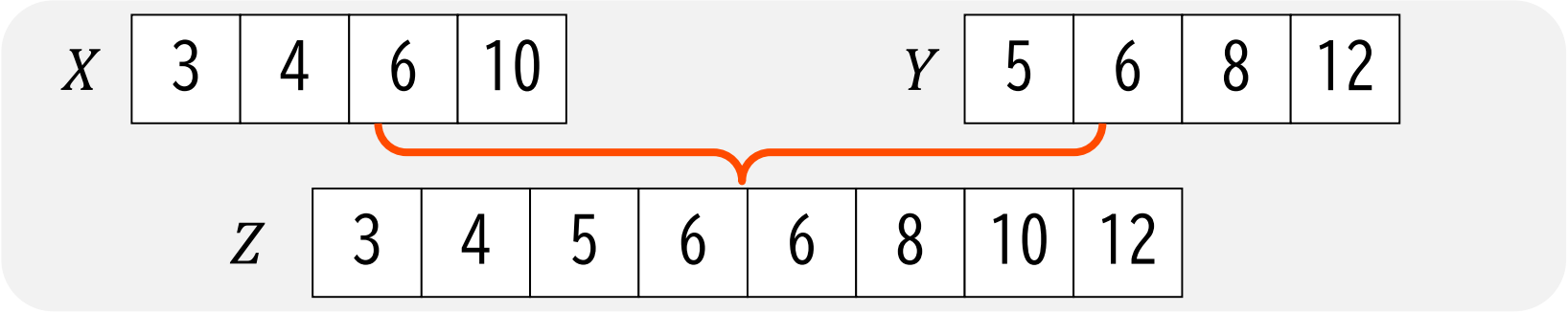
➢ 時間計算量の下限を達成する最適なアルゴリズムは?



2-3（2）併合問題

■ 併合問題

- 入力
  - 整列した系列$X = (x_1, x_2, \ldots, x_p)$
    - $x_1 \leq x_2 \leq \cdots \leq x_p$
  - 整列した系列$Y = (y_1, y_2, \ldots, y_q)$
    - $y_1 \leq y_2 \leq \cdots \leq y_q$
- 質問
  - XとYを併合した整列した系列$Z = (z_1, z_2, \ldots, z_n)$を構築せよ
    - $\{x | x \in X\} + \{y | y \in Y\} = \{z | z \in Z\}$
    - $z_1 \leq z_2 \leq \cdots \leq z_n$
    - 但し $n = p + q$



併合アルゴリズム

■ Algorithm 2.2（併合）

- 入力：整列した$X = (x_1, x_2, \ldots, x_p)$, $Y = (y_1, y_2, \ldots, y_q)$
- 出力：整列した$Z = (z_1, z_2, \ldots, z_n)$

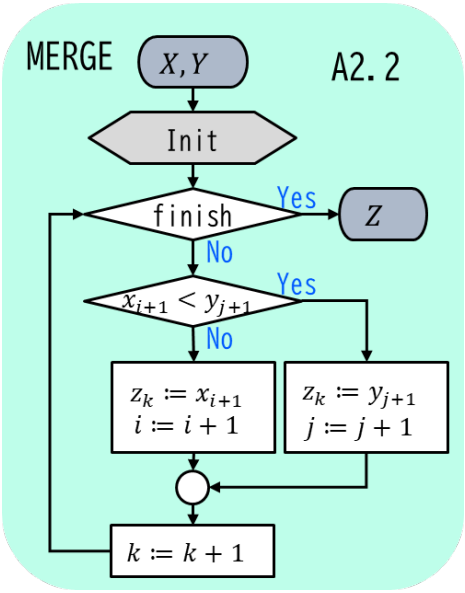
Step 0: Set $i = 0, j = 0, k = 1, x_{p+1} = \infty, y_{q+1} = \infty$

Step 1: If $i + j = n$ then output $(z_1, z_2, \ldots, z_n)$, and halt

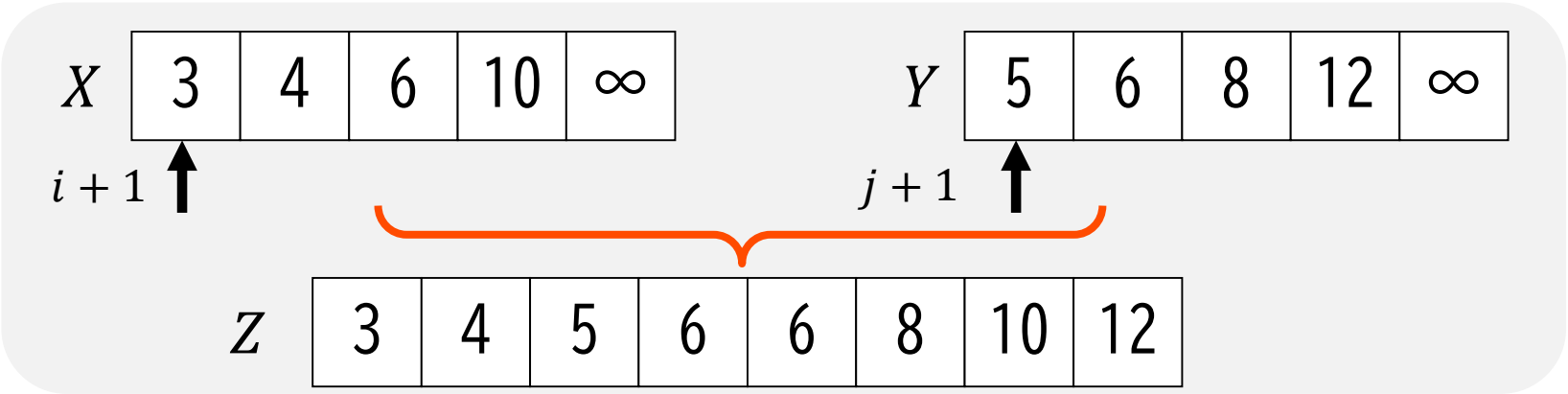
Step 2: If $x_{i+1} < y_{j+1}$ then set $z_k = x_{i+1}$ and $i = i + 1$, and go to Step 4

Step 3: Set $z_k = y_{j+1}$ and $j = j + 1$

Step 4: Set $k = k + 1$ and return to Step 1



■ Example:

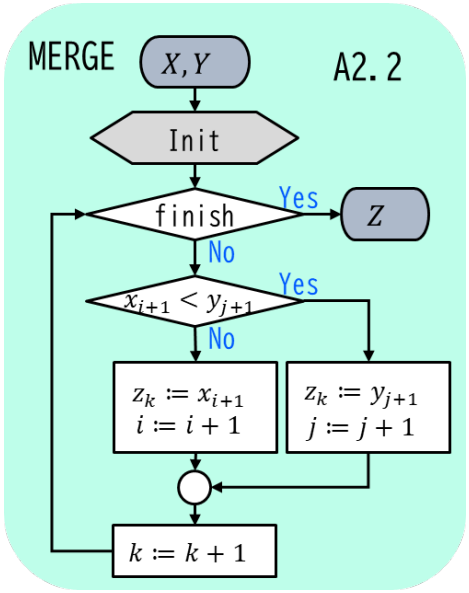


併合(Merging)

■ 補題（Lemma 2.1）
- 併合(Algorithm 2.2)は併合問題をn回の比較で解く

□ 証明
- Zにk番目に加えられる要素$z_k \in Z$
  - XまたはYの要素で，Z中で重複しないことは明らか
- Zの$k+1$番目以降にz_kより小さい要素が存在しないことを示す
  - z_kは$i=\alpha, j=\beta$のときにZに加えられたとする
    - $k=\alpha+\beta+1,\ z_k \le x_{\alpha+1} \le x_{\alpha+2} \le \cdots \le x_p,\ z_k \le y_{\beta+1} \le y_{\beta+2} \le \cdots \le y_q$
  - $z_l \in (x_{\alpha+1}, x_{\alpha+2}, \ldots, x_p, y_{\beta+1}, y_{\beta+2}, \ldots, y_q)\ (\forall l > k+1)$
  - $z_k \le z_l\ (\forall l > k+1)$
- 任意のkで成り立つ
  - $z_1 \le z_2 \le \cdots \le z_n$（$Z$は整列）
- 各z_kを選択するときx_{i+1}とy_{j+1}を比較（それ以外に比較しない）

✓ n回の比較 ■



X	3	4	6	10	∞	Y	5	6	8	12	∞

Z	3	4	5	6	6	8	10	12

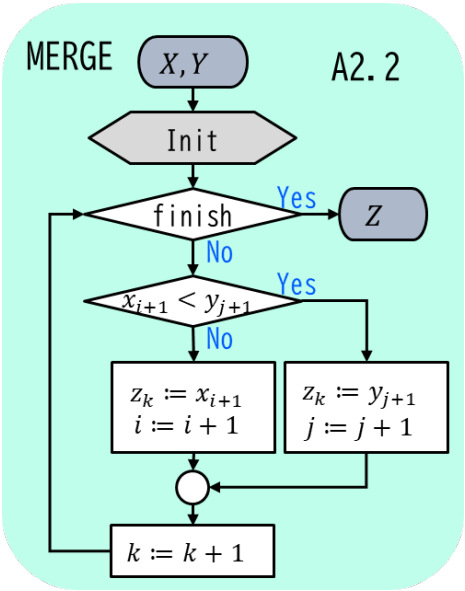
併合(Merging)

■ 定理 (Theorem 2.7)

- 併合(Algorithm 2.2)は併合問題を線形時間で解く

□ 証明

- 正当性 ： 補題2.1より
- 時間計算量： $O(n)$



■ Algorithm 2.2 (併合)

- 入力：整列した$X = (x_1, x_2, \ldots, x_p)$, $Y = (y_1, y_2, \ldots, y_q)$
- 出力：整列した$Z = (z_1, z_2, \ldots, z_n)$

Step 0: Set $i = 0, j = 0, k = 1, x_{p+1} = \infty, y_{q+1} = \infty$ — $O(1)$

Step 1: If $i + j = n$ then output $(z_1, z_2, \ldots, z_n)$, and halt — $O(1)$

Step 2: If $x_{i+1} < y_{j+1}$ then set $z_k = x_{i+1}$ and $i = i + 1$, and go to Step 4 — $O(1)$

Step 3: Set $z_k = y_{j+1}$ and $j = j + 1$ — $O(1)$

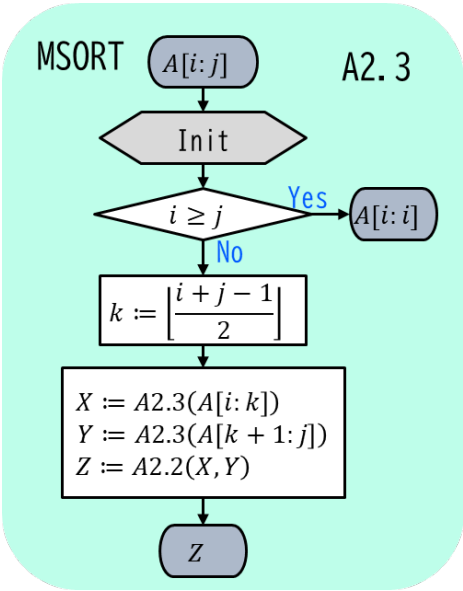
Step 4: Set $k = k + 1$ and return to Step 1 — $O(1)$

出力：$O(n)$

n回

2-3（3）併合整列アルゴリズム

■ 整列問題
- 入力：系列$X = (x_1, x_2, \ldots, x_n)$　但し $(x_1, x_2, \ldots, x_n \in \mathbb{Z})$
- 質問：Xを大きさの昇順に並べかえよ



■ Algorithm 2.3（併合整列）
- 入力：系列$A[i:j] = (a_i, a_{i+1}, \ldots, a_j)$
- 出力：$A[i:j]$を整列した系列$Z[i:j] = (z_i, z_{i+1}, \ldots, z_j)$

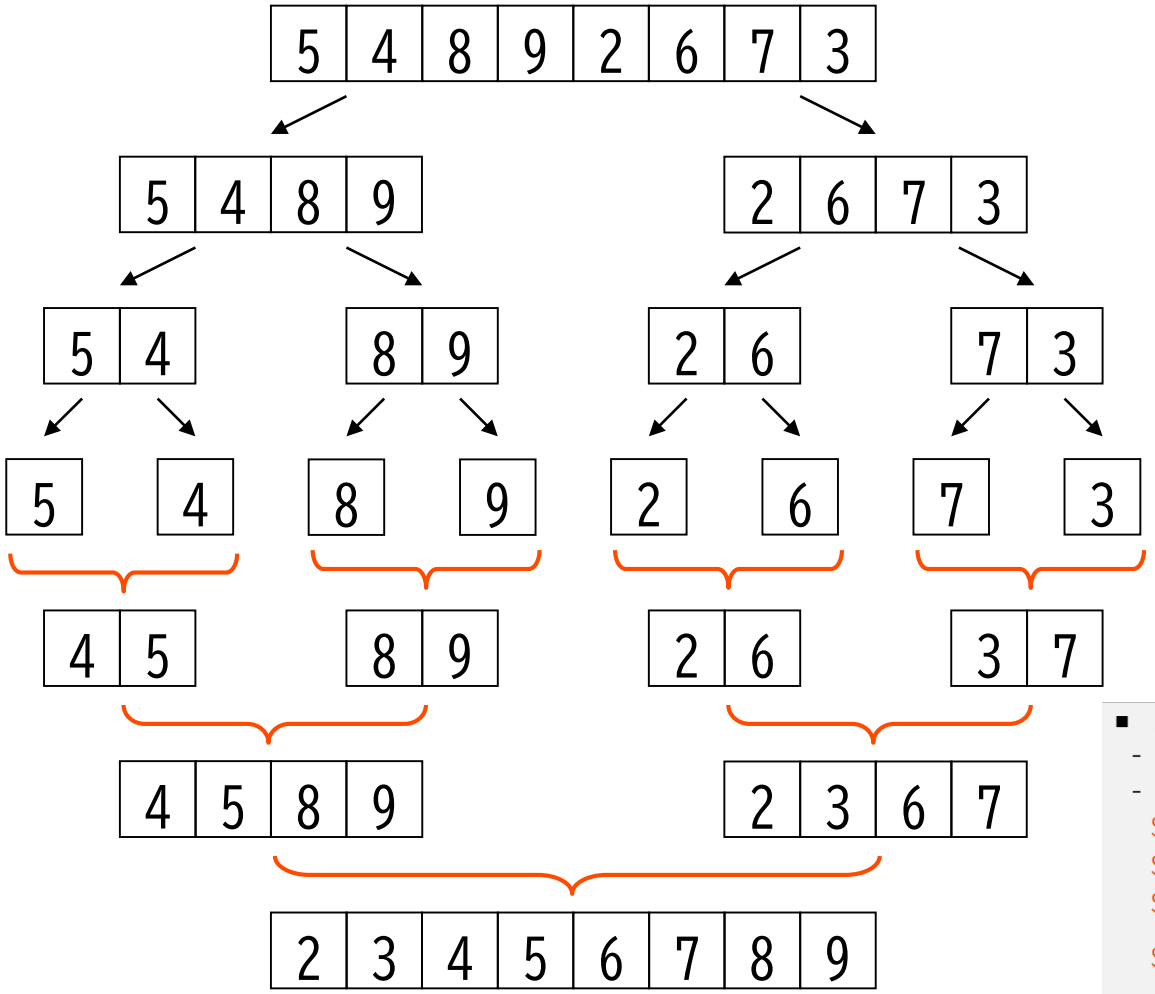
Step 0: If $i \geq j$, then output $A[i:i]$, and halt
Step 1: set $k = \left\lfloor \frac{i+j-1}{2} \right\rfloor$
Step 2: Apply 併合整列(2.3) to $A[i:k]$
and store output in $X[i:k]$
Step 3: Apply 併合整列(2.3) to $A[k+1:j]$
and store output in $Y[k+1:j]$
Step 4: Apply 併合(2.2) to $X = X[i:k], Y = Y[k+1:j]$
and store output in $Z[i:j]$, and halt

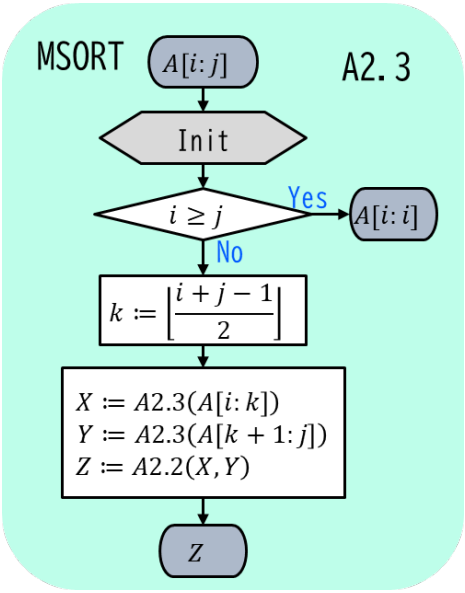
➢ 再帰的に系列の2分割を繰り返す
➢ 整列した2系列を併合

■ 系列Xは配列に格納
- $A = A[1:n] = \langle A[1], A[2], \ldots, A[n] \rangle$

$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	$A[7]$	$A[8]$
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8

併合整列アルゴリズム：例





■ Algorithm 2.3（併合整列）

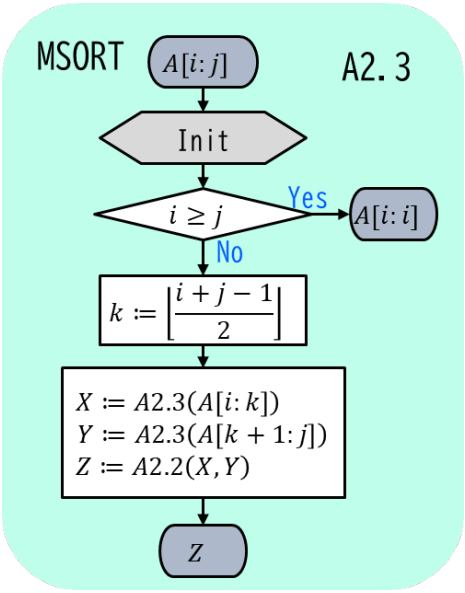
- 入力：系列 $A[i:j] = (a_i, a_{i+1}, \ldots, a_j)$
- 出力：$A[i:j]$を整列した系列 $Z[i:j] = (z_i, z_{i+1}, \ldots, z_j)$

Step 0: If $i \geq j$, then output $A[i:i]$, and halt
Step 1: set $k = \lfloor \frac{i+j-1}{2} \rfloor$
Step 2: Apply 併合整列(2.3) to $A[i:k]$
and store output in $X[i:k]$
Step 3: Apply 併合整列(2.3) to $A[k+1:j]$
and store output in $Y[k+1:j]$
Step 4: Apply 併合(2.2) to $X = X[i:k], Y = Y[k+1:j]$
and store output in $Z[i:j]$, and halt

併合整列アルゴリズム (Merge Sort)

■ 定理 (Theorem 2.8)
- 併合整列(Algorithm 2.3)は整列問題を$O(n \log n)$で解く

□ 証明
- 正当性 : 定理2.7より
  - 再帰的に2分割し得られる長さ1の系列は整列している
  - 併合(Algorithm 2.2)は正しく併合し整列した系列を得る
- 時間計算量:
  - $T(n)$: 併合整列アルゴリズム2.3の時間計算量
  - ✓ $T(n) = O(1) + O(1) + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
  - ✓ $T(1) = O(1)$



■ Algorithm 2.3 (併合整列)
- 入力:系列$A[i:j] = (a_i, a_{i+1}, \ldots, a_j)$
- 出力:$A[i:j]$を整列した系列$Z[i:j] = (z_i, z_{i+1}, \ldots, z_j)$

Step 0: If $i \geq j$, then output $A[i:i]$, and halt — $O(1)$

Step 1: set $k = \lfloor \frac{i+j-1}{2} \rfloor$ — $O(1)$

Step 2: Apply 併合整列(2.3) to $A[i:k]$ and store output in $X[i:k]$ — $T(\lfloor n/2 \rfloor)$

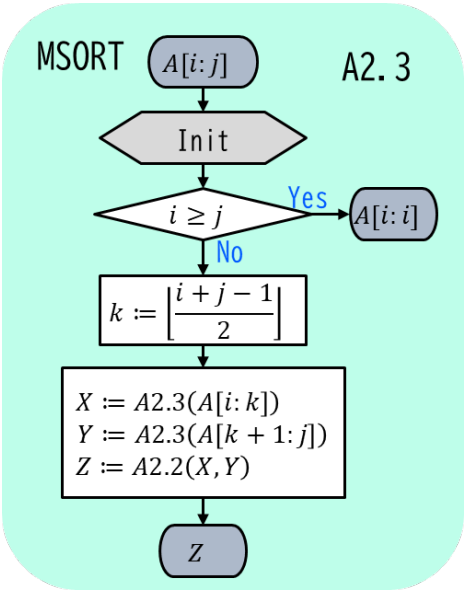
Step 3: Apply 併合整列(2.3) to $A[k+1:j]$ and store output in $Y[k+1:j]$ — $T(\lceil n/2 \rceil)$

Step 4: Apply 併合(2.2) to $X = X[i:k], Y = Y[k+1:j]$ and store output in $Z[i:j]$, and halt — $O(n)$

併合整列アルゴリズム (Merge Sort)

■ 定理 (Theorem 2.8)
- 併合整列(Algorithm 2.3)は整列問題を$O(n \log n)$で解く

□ 証明(つづき)
- $T(n) = O(1) + O(1) + T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
  $= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
- $p = 2^{\lceil \log_2 n \rceil}$
  - $n \le p \le 2n, p = \Theta(n)$
- $T(n)$
  $\le T(\lfloor p/2 \rfloor) + T(\lceil p/2 \rceil) + O(p)$
  $\le 2T\left(\frac{p}{2}\right) + cp \le 2\left(2T\left(\frac{p}{4}\right) + \frac{cp}{2}\right) + cp$
  $= 2 \times 2 \times T\left(\frac{p}{4}\right) + cp + cp$
  $\le 2 \times 2 \times \cdots \times 2 \times T(1) + cp + cp + \cdots + cp$
  $= 2^{\log_2 p} T(1) + cp \log_2 p$
  $= pT(1) + cp \log_2 p$
  $= O(p) + O(p \log p)$
  $= O(n \log n)$ ■



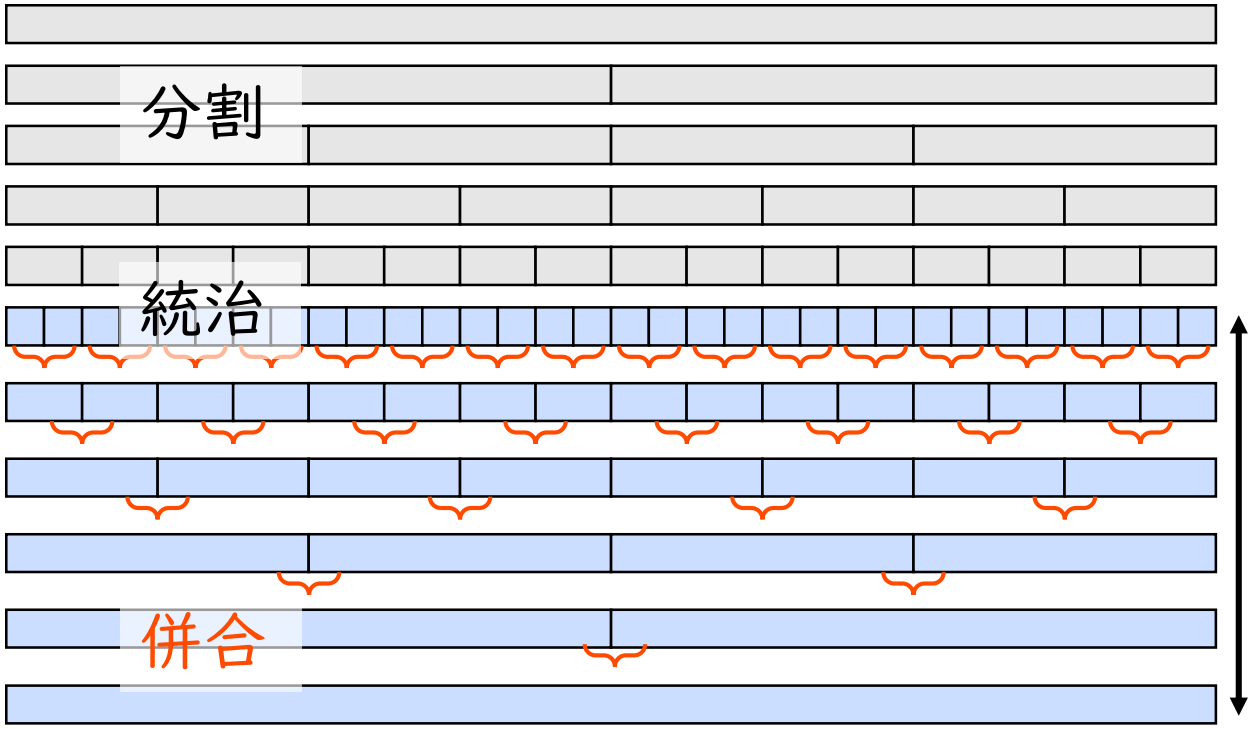
■ Algorithm 2.3 (併合整列)
- 入力:系列$A[i:j] = (a_i, a_{i+1}, \ldots, a_j)$
- 出力:$A[i:j]$を整列した系列$Z[i:j] = (z_i, z_{i+1}, \ldots, z_j)$

Step 0: If $i \ge j$, then output $A[i:i]$, and halt
Step 1: set $k = \lfloor \frac{i+j-1}{2} \rfloor$
Step 2: Apply 併合整列(2.3) to $A[i:k]$
and store output in $X[i:k]$
Step 3: Apply 併合整列(2.3) to $A[k+1:j]$
and store output in $Y[k+1:j]$
Step 4: Apply 併合(2.2) to $X = X[i:k], Y = Y[k+1:j]$
and store output in $Z[i:j]$, and halt

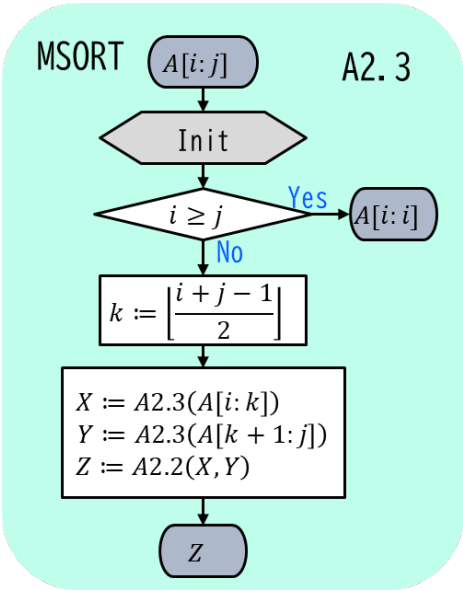
併合整列アルゴリズムの計算量

- $T(n) \approx 2^k \cdot O\left(\frac{n}{2^k}\right) + \cdots + 2 \cdot O\left(\frac{n}{2}\right) + 1 \cdot O(n)$

$$\approx \underbrace{O(n) + \cdots + O(n) + O(n)}_{\log_2 n} = O(n \log n)$$



段数$k \approx \log_2 n$



各段の併合操作

$$\begin{aligned} 2^k \times O(n/2^k) &= O(n) \\ \vdots \qquad\qquad &\quad \vdots \\ 4 \times O(n/4) &= O(n) \\ 2 \times O(n/2) &= O(n) \\ 1 \times \quad O(n) &= O(n) \end{aligned}$$

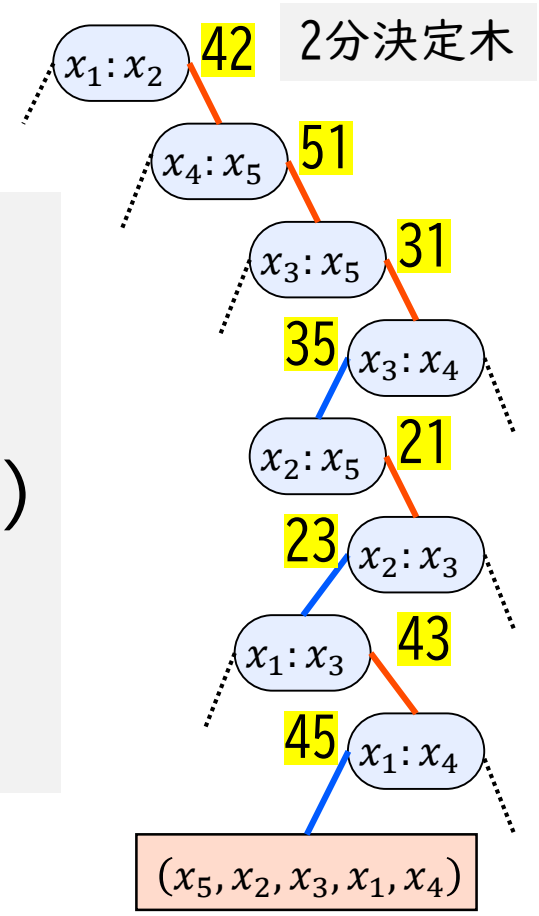
マージソート (merge sort)

- 整列した列を併合
- 時間計算量$O(n \log n)$

- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,2,3,5,1)$

(4)(2)(3)(5)(1)
(24) (3)(5)(1)
(24) (3) (15)
(24) (1) (3)(5)
(24) (135)

(24)(135)
(1)(24)(35)
(12) (4) (35)
(123) (4)(5)
(12345)



バブルソート (bubble sort)

- 配列上の隣接データを比較
- 時間計算量$O(n^2)$

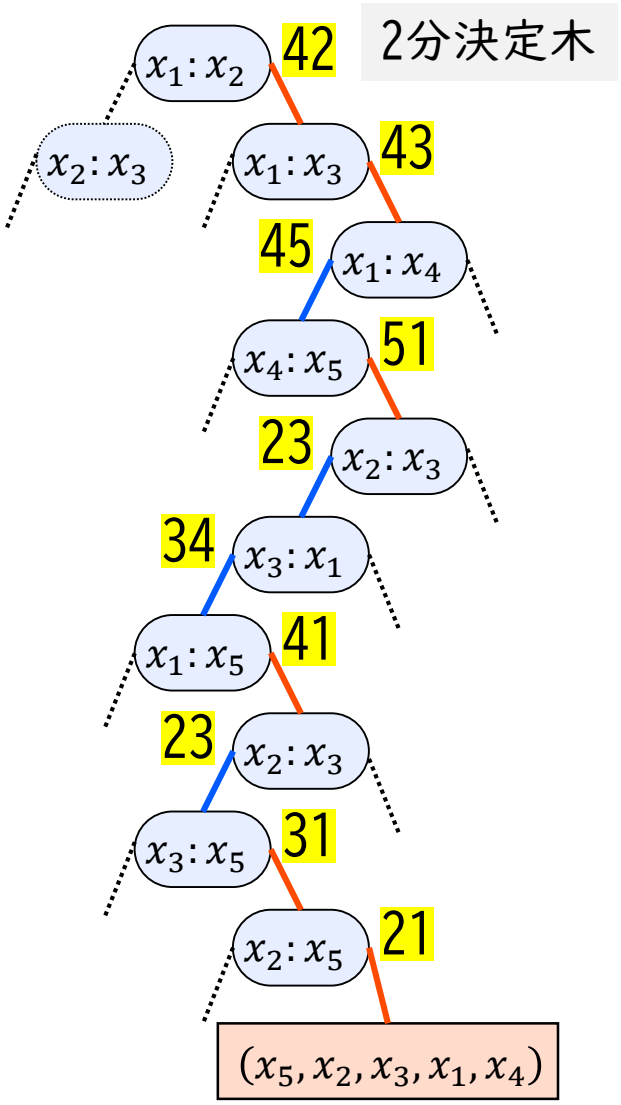
- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,2,3,5,1)$

42351 → 24351 → 23451 → 23451 → 23415

23415 → 23415 → 23415 → 23145

23145 → 23145 → 21345

21345 → 12345



バブルソート (bubble sort)

- 配列上の隣接データを比較
- 時間計算量$O(n^2)$

- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,1,3,5,2)$

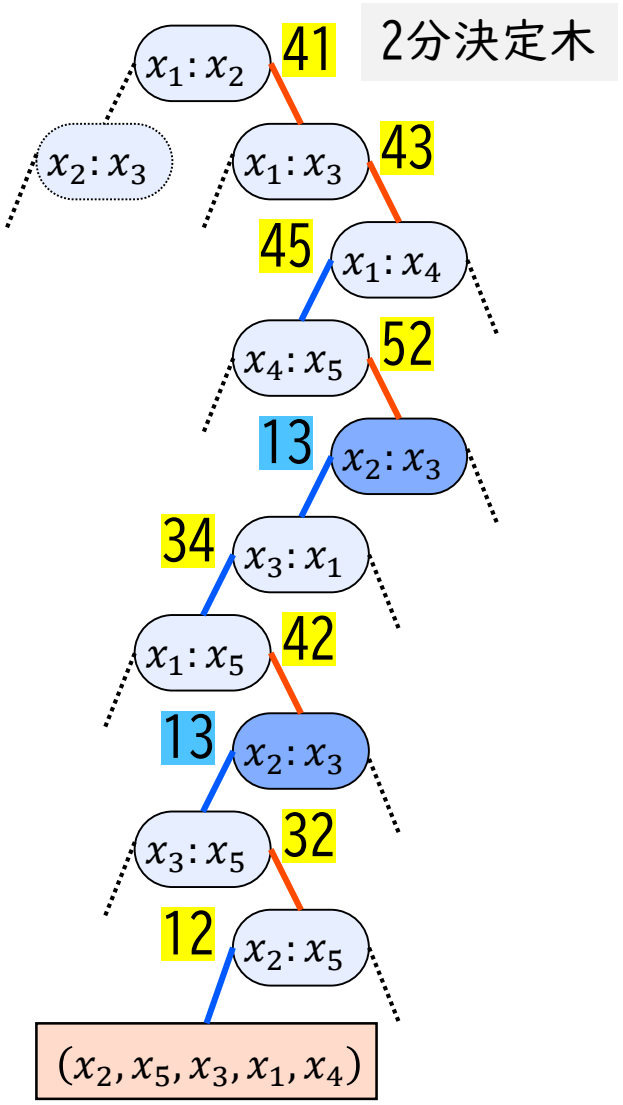
41352 → 14352 → 13452 → 13452 → 13425

13425 → 13425 → 13425 → 13425 → 13245

13245 → 13245 → 13245 → 12345

12345 → 12345

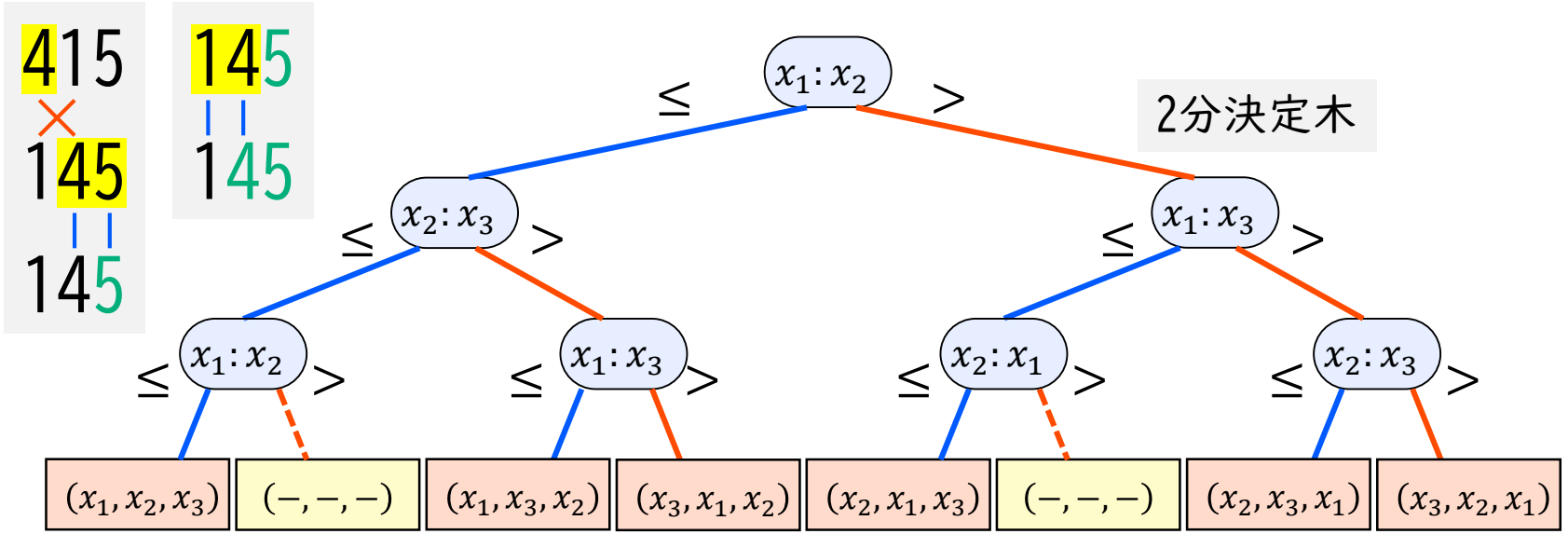
同じ比較を繰り返すことあり=冗長
並列処理と相性はよい



バブルソート (bubble sort)

- 配列上の隣接データを比較
- 時間計算量$O(n^2)$

- 例
  - $(x_1, x_2, x_3) = (4,1,5)$



いかなるデータでも到達しない葉もある
無駄な比較も計算量の評価では考慮する必要あり

シンプルソート（simple sort）（馬鹿ソート）

- 未整列の最後と比較
- 時間計算量$O(n^2)$

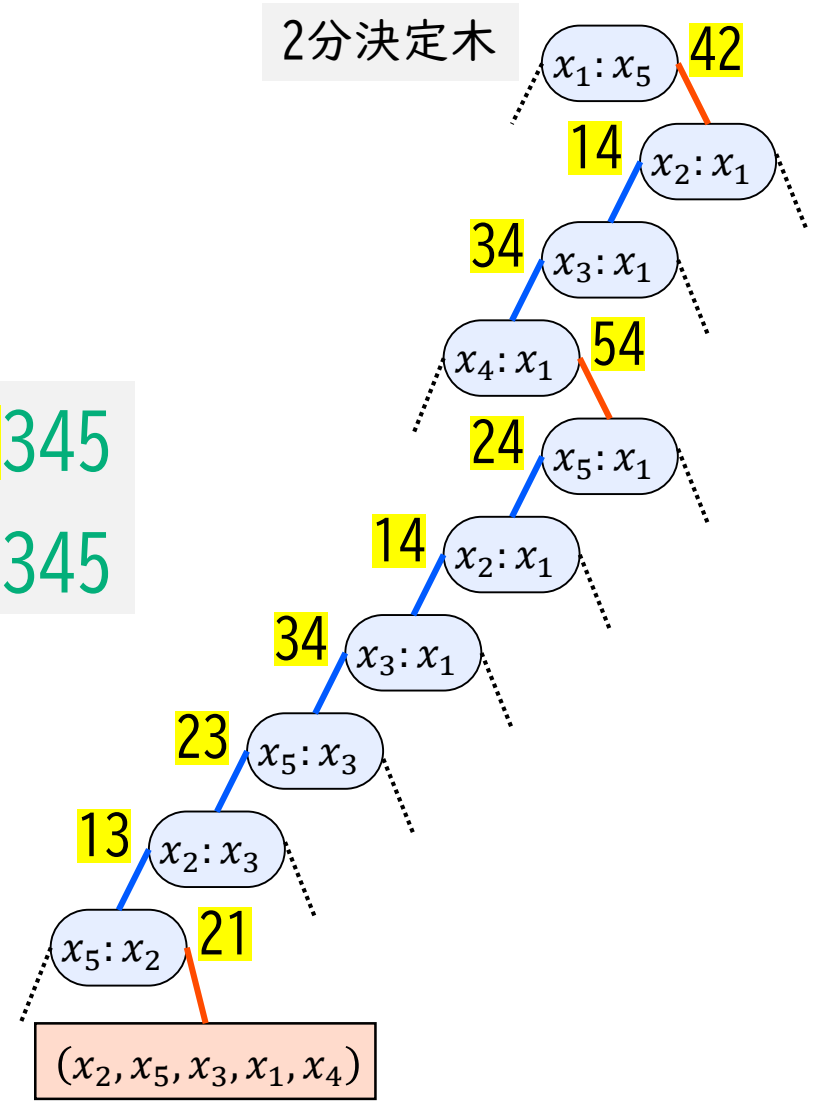
- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (4,1,3,5,2)$

41352 → 21354 → 21354 → 21354 → 21345

21345 → 21345 → 21345 → 21345

21345 → 21345 → 21345

21345 → 12345

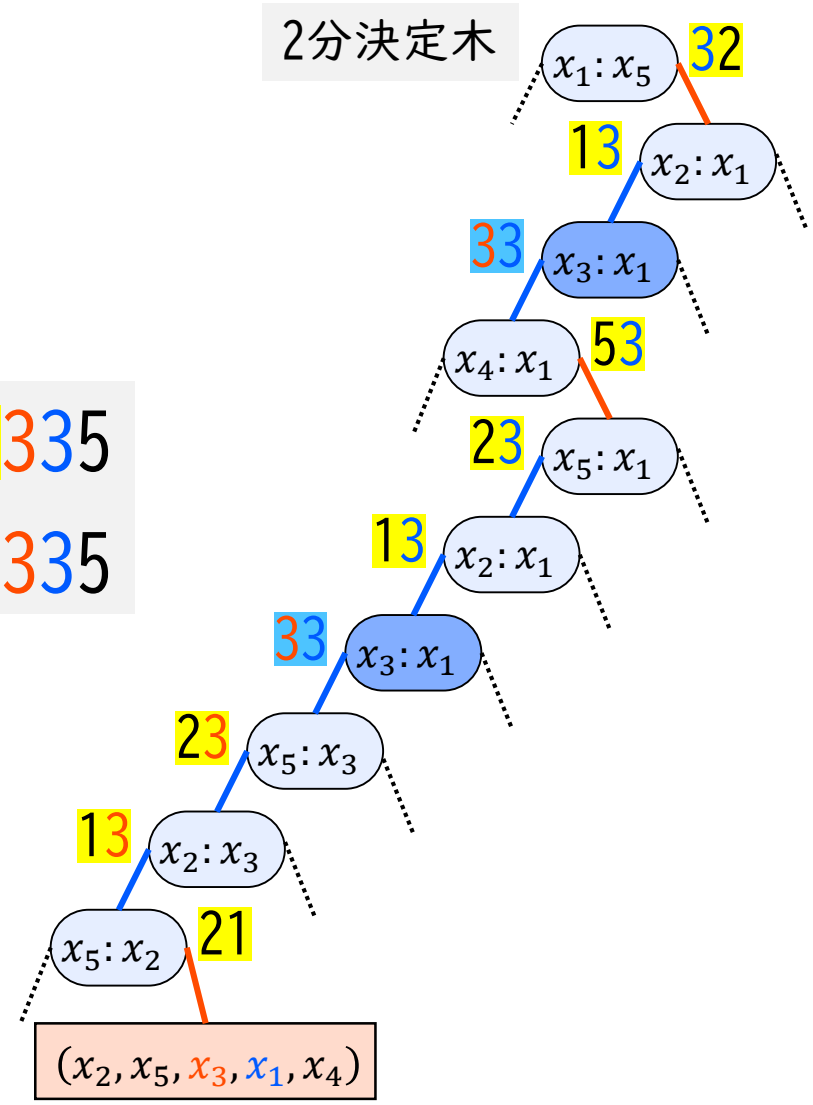


シンプルソート（simple sort）（馬鹿ソート）

- 未整列の最後と比較
- 時間計算量$O(n^2)$

- 例
  - $(x_1, x_2, x_3, x_4, x_5) = (3,1,3,5,2)$

安定性なし
同等データの順序が
保存されない



整列アルゴリズム(sorting)

- 様々なアルゴリズム
  - 時間計算量
  - 空間計算量
  - 安定性
  - 並列処理との相性
  - データ量とデータの特徴
- 最適アルゴリズム$O(n \log n)$
  - 併合整列(Merge Sort)
  - ヒープソート(Heap Sort)
- 非最適アルゴリズム
  - クイックソート(Quick Sort)
    - $O(n^2)$
    - 平均性能は$O(n \log n)$
  - バブルソート(Bubble Sort)
    - $O(n^2)$

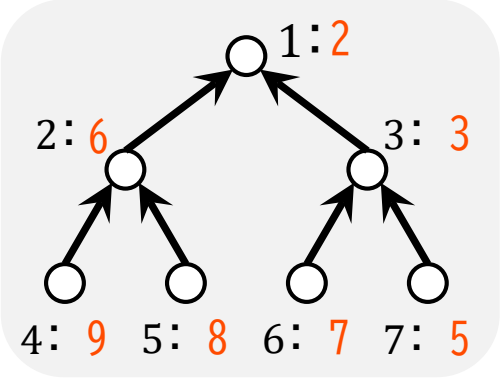
おまけ

ヒープ(heap)構造

■ 二分木

- 「親の値」は「子の値」より小さいか等しい
  - ✓ 根の値は最小値
- 配列で実現可能
  - ✓ 親アドレスa ⇒ 子アドレス$2a$, $2a+1$

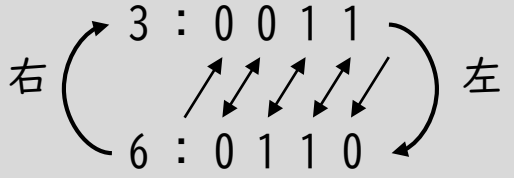
点数=データ数n
⇒ 木の高さ$h = \lfloor \log_2 n \rfloor$

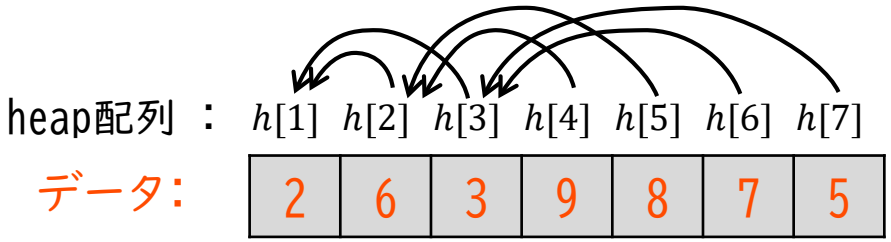


親アドレスから子アドレス計算
- 2倍 ⇒ 左1ビットシフト
- +1 ⇒ 加算 ⇒ 最下位ビットを1に

子アドレスから親アドレス計算
- 除算(½倍)
  ⇒ 右1ビットシフト

シフト演算





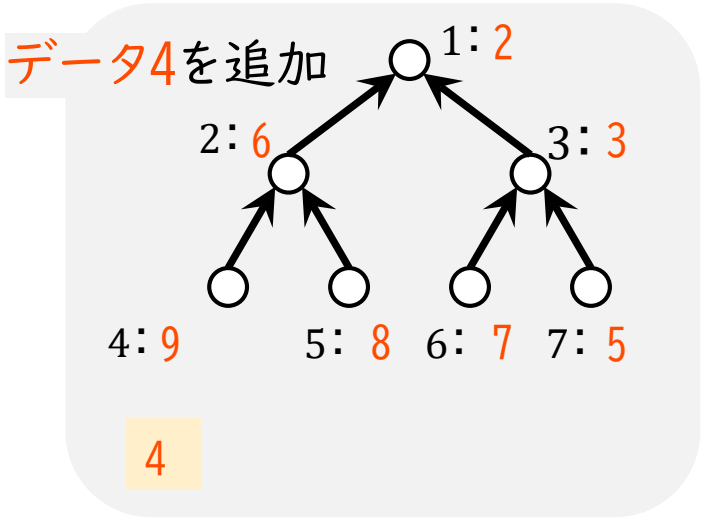
ヒープ(heap)構造の構築・維持

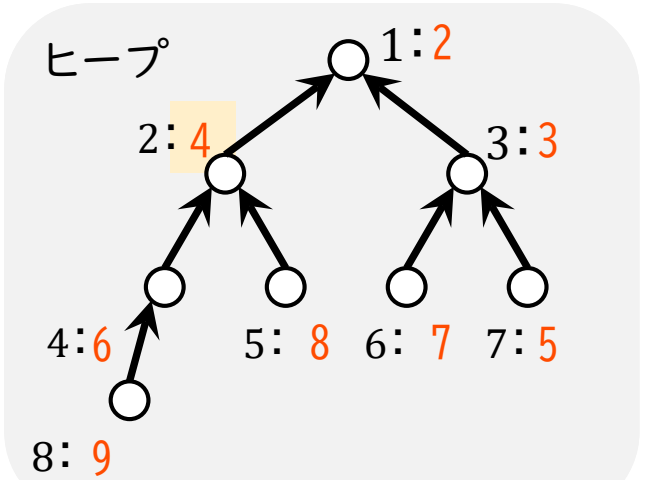
■ データの追加（up-heap）： $O(\log n)$

- $h[n+1]$にデータ追加
- 親の値より小さければ交換 ⇒ 根に向かって繰り返す

$n = |V|, m = |E|$

点数=データ数n
⇒ 木の高さ$h = \lfloor \log_2 n \rfloor$





$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$	$h[8]$
2	6	3	9	8	7	5	

$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$	$h[8]$
2	4	3	6	8	7	5	9

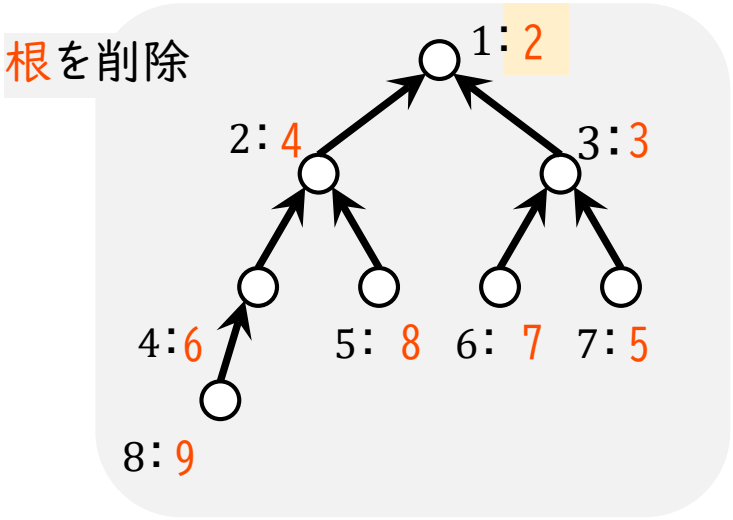
ヒープ(heap)構造の構築・維持

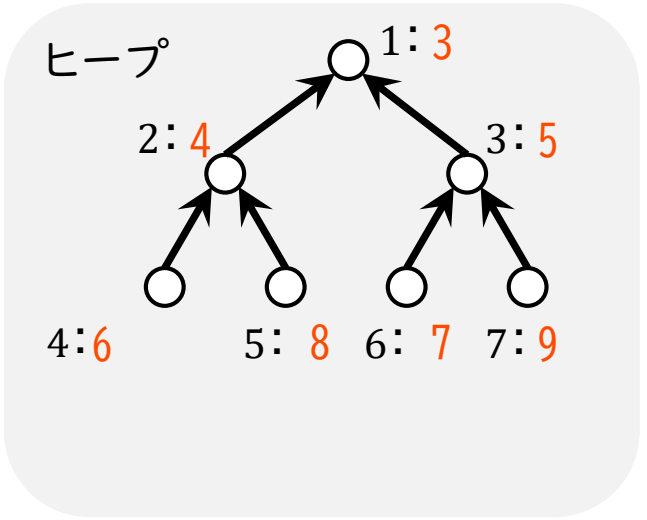
■ データの追加（up-heap）： $O(\log n)$
- $h[n+1]$にデータ追加
- 親の値より小さければ交換 ⇒ 根に向かって繰り返す

$n = |V|, m = |E|$

■ 根の削除(down-heap)： $O(\log n)$
- 最後のデータを根に移動
- 小さい子より大きければ交換 ⇒ 交換したら葉に向かって繰り返す

点数=データ数n
⇒ 木の高さ$h = \lfloor \log_2 n \rfloor$





$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$	$h[8]$
2	4	3	6	8	7	5	9

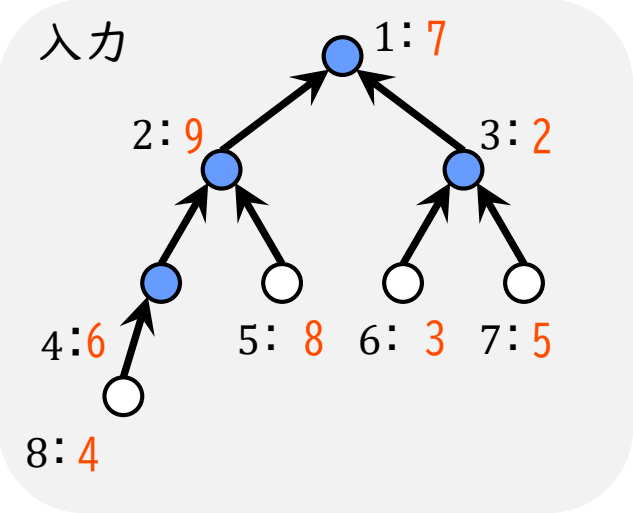
$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$
3	4	5	6	8	7	9

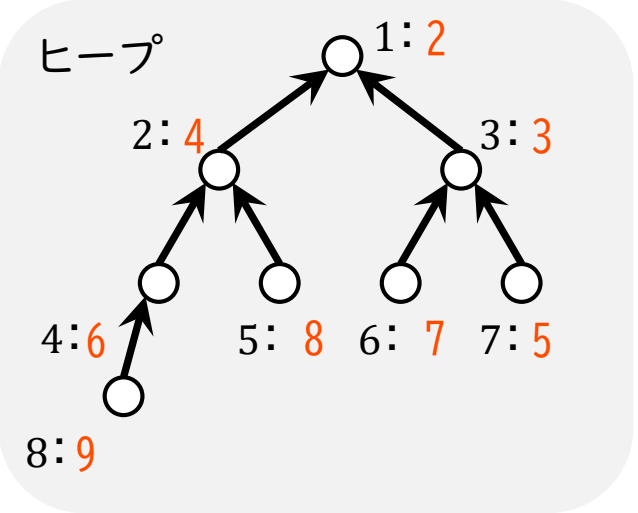
ヒープ(heap)構造の利用

- ヒープソート $O(n \log n)$
  - ヒープの構築 ： $O(n)$
    - 葉側の内点から順にdown-heap
      - $O\left(\sum_{i=0}^{k-1}(k-i)2^i\right) = O(n)$
  - データを昇順で出力： $O(n \log n)$
    - 根の削除(down-heap)の繰り返し

$n = |V|, m = |E|$

点数=データ数n
⇒ 木の高さ$h = \lfloor \log_2 n \rfloor$





$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$	$h[8]$
7	9	2	6	8	3	5	4

$h[1]$	$h[2]$	$h[3]$	$h[4]$	$h[5]$	$h[6]$	$h[7]$	$h[8]$
2	4	3	6	8	7	5	9

問題：素数

- PRIMES（判定問題）
  - 入力
    - $n \in \mathbb{N}$
  - 質問
    - nは素数か?

素数判定アルゴリズム

■ Naïve algorithm for PRIMES

- Exhaustive Search
- 入力:$n \in \mathbb{N}$
- 出力:“Yes” or “No”

Step 0: Set $i = 2$

Step 1: If $i > \lfloor \sqrt{n} \rfloor$, then output “YES”, and halt

Step 2: If i divides n, then output “NO”, and halt

Step 3: Set $i = i + 1$, and return to Step 1

- 正当性 : 自明
- 計算複雑度 : $O(\sqrt{n})$

- このアルゴリズムは多項式時間アルゴリズムか?

整数の大きさ（確認）

■ 整数nの大きさ$|n|$（入力の大きさ）

- 10進数表現$d_h d_{h-1} \ldots d_1 d_0$（基数10, $d_i \in \{0,1,2,\ldots,9\}$）

$$n = d_h \times 10^h + d_{h-1} \times 10^{h-1} + \cdots + d_1 \times 10^1 + d_0 \times 10^0$$

  - 例
    - $1892_{(10)} = 1 \times 10^3 + 8 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$
- 2進数表現$b_s b_{s-1} \ldots b_1 b_0$　（基数2 $b_i \in \{0,1\}$）

$$n = b_s \times 2^k + b_{s-1} \times 2^{k-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$$

  - 例
    - $101_{(2)} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5_{(10)}$
    - $11001_{(2)} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 25_{(10)}$

■ 整数nはsビットで表現される（2進数の場合）

- $|n| = s = \lfloor \log_2 n \rfloor + 1 = \Theta(\log n)$

素数判定アルゴリズム

■ Naïve algorithm for PRIMES

- Exhaustive Search
- 入力:$n \in \mathbb{N}$
- 出力:“Yes” or “No”

Step 0: Set $i = 2$

Step 1: If $i > \lfloor \sqrt{n} \rfloor$, then output “YES”, and halt

Step 2: If i divides n, then output “NO”, and halt

Step 3: Set $i = i + 1$, and return to Step 1

n : $b_s b_{s-1} \dots b_1 b_0$

$|n| = s = \Theta(\log n)$

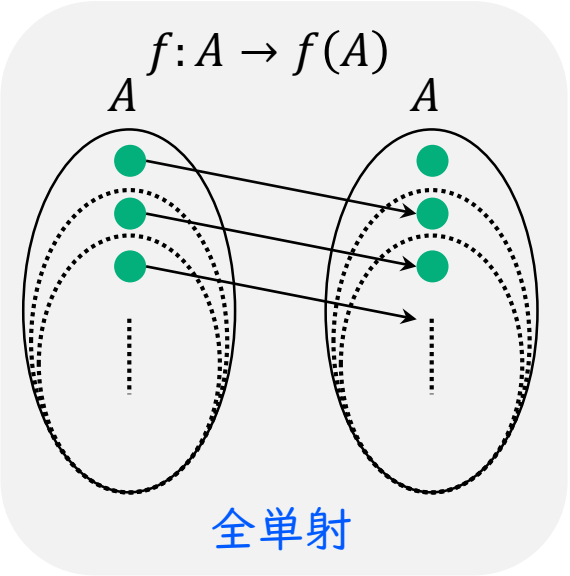
$n = \Theta(2^s) = \Theta(2^{|n|})$

- 正当性 : 自明
- 計算複雑度 : $O(\sqrt{n})$
  - $O(\sqrt{n}) = O(\sqrt{2^s}) = O\left(2^{\frac{s}{2}}\right) = O\left(2^{\frac{|n|}{2}}\right)$
  - 指数オーダー

無限（確認）

■ 無限集合
- そのある真部分集合と同型である集合
- $\exists S' \subset S, S \cong S'$

➢ $\mathbb{N} \neq \mathbb{N} \setminus \{0\} \subset \mathbb{N}$
- ✓ $\mathbb{N} \cong \mathbb{N} \setminus \{0\}$
  - ■ 全単射$f: n \mapsto n+1$ が存在



■ 可算無限
- 非負偶数$\mathbb{E}$，正奇数$\mathbb{O}$
- $\mathbb{N} = (\mathbb{E}, \mathbb{O})$，$\mathbb{E} \subset \mathbb{N}$，$\mathbb{O} \subset \mathbb{N}$

➢ $\mathbb{N} \cong \mathbb{E}$，$\mathbb{N} \cong \mathbb{O}$

➢ $\mathbb{N} \cong \mathbb{E} \cong \mathbb{O} \cong \mathbb{Z} \cong \mathbb{N} \times \mathbb{N}$
- ➢ $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{E}| = |\mathbb{O}| = |\mathbb{N} \times \mathbb{N}|$

■ 非可算無限

➢ $\mathbb{R} \cong 2^{\mathbb{N}} = \{S | S \subseteq \mathbb{N}\} \not\cong \mathbb{N}$
- カントール(Cantor)の対角線論法
  - ➢ $|\mathbb{N}| < |\mathbb{R}| = |2^{\mathbb{N}}|$

プログラムは万能か？

- 異なる関数は、異なるプログラムコードで記述される
  - すべての関数を記述できるプログラム言語は存在するか？

- アルファベット集合Σ上の長さnの記号列の集合
  - $\Sigma^n = \underbrace{\Sigma \times \Sigma \times \cdots \times \Sigma}_{n}$
- 有限長の記号列の集合Σ^*
  - $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n \cong \mathbb{N}$
- プログラムの集合$\mathcal{C}_\Sigma \subseteq \Sigma^*$
  - $|\mathcal{C}_\Sigma| \leq |\Sigma^*| = |\mathbb{N}| = \aleph_0$

✓ プログラムの数は可算無限

無限長の記号列は非可算無限

- 自然数$\mathbb{N}$上の関数$f: \mathbb{N} \to \mathbb{N}$の集合$\mathbb{N}^{\mathbb{N}}$
  - $|\mathbb{N}| < |\mathbb{N}^{\mathbb{N}}|$

✓ 関数の数は非可算無限

➢ プログラムで表現できない関数が存在

Bad Coding Example (1)

- `int ffnum = ff;  //(#ff in an input circuit)`
- `bool CConstGraph::bellmanford( const float period )`

```
1.   {
2.     bool visit[ffnum];
3.     for( int i=0; i<ffnum; i++ )
4.         visit[i] = false;
5.     for( int start = 0; start < ffnum; ) {
6.       visit[start] = true;
7.       if( !bellmanford( start, visit, period ) ) return false;
8.       while( ++start < ffnum )
9.         if( !visit[start] ) break;
10.    }
11.    return true;
12.  }
```

この記述では配列サイズは定数
ffnumの値とは無関係

- ➢ compiler error, warning なし
- ✓ 小さいデータで動作
- ✓ 大きいデータでsegmentation fault

Bad Coding Example (2)

- vector<cgFF> cgff; // Flip Flop
- void CConstGraph::addFF(cgFF ff) {

```
1.   if( cgff.size() != 0 && ( cgff.size() % 2000 ) == 0 )
2.     cout << "reading " << cgff.size() << " registers" << endl;
3.     cgff.push_back( ff );
4.   }
```

cgpath.size()の時間計算量$O(n)$

- list<cgPath> cgpath; // FF Path
- void CConstGraph::addPath(cgPath pt) {

```
1.   if( cgpath.size() != 0 && ( cgpath.size()  % 10000 ) == 0 )
2.     cout << "reading " << cgpath.size() << " paths" << endl;
3.     cgpath.push_back( pt );
4.   }
```

関数の時間計算量$O(n) \Rightarrow O(n^2)$

- ➢ コード挿入の結果，データ読み込み時間
  - ✓ 10秒が10分に
  - ✓ 1分が1時間に

Bad Coding Example (3)

■ inline size_t __stl_hash_string(const char* __s) {

```
1.     unsigned long __h = 0;
2.     for ( ; *__s; ++__s)
3.       __h = 5*__h + *__s;
4.     return size_t(__h);
5.   }
```

- ➢ データ読み込み時間
- ✓ ハッシュなし 1時間
- ✓ ハッシュ関数(a) 2分
- ✓ ハッシュ関数(b) 1分

■ ハッシュ関数

- ストリング(s) (既存)
  - ■ __stl_hash_string(s.c_str());
- 辺(ストリング対)(p.first,p.second) (独自に用意する必要あり)
  - ✓ 定義(a)
  - ■ __stl_hash_string(p.first.c_str())+__stl_hash_string(p.second.c_str());
  - ✓ 定義(b)
  - ■ __stl_hash_string((p.first + p.second).c_str());