情報構造 第五回

ソーティングアルゴリズム

今日の予定

- ソーティングとは
- ソーティングアルゴリズム
  - 3つの単純法
  - ヒープソート

ソーティングとは

- 整列問題
  - 入力：n個の数
  - 出力：小さい（大きい）順の列

ソーティング例
(4, 5, 2, 1) => (1, 2, 3, 4)

- なぜソーティングアルゴリズムを学ぶのか
  - 計算機が遅い頃，ソートを行うために効率の良いアルゴリズムが考えられてきた
    - 現在では，そんなに頑張らなくても良い
    - ライブラリの充実，メモリが大きい，オブジェクト指向
    - 今後，自分でソートアルゴリズムを実装することはないでしょう．．．
  - ではなぜ？
  - 多くのアルゴリズムが存在する
    - 同じタスクのアルゴリズムの違いを体感できる
    - 計算量の違い

ソーティングの前準備

- 集合 A_1, A_2, A_3,… A_nを，項目中のキーの順序で並び替えること
  - 要素 A_i：型item

```
typedef struct{
        int key;
        …
} item;
```

- ソートされる項目は，要素がitem型，要素数がn，すなわちindexが0からn-1の配列Aに格納

```
item A[i];
```

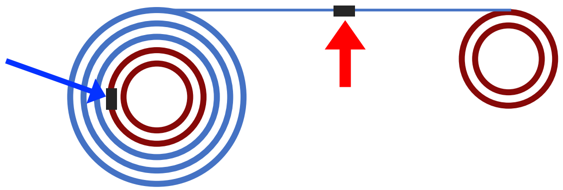
- 第 i 項目のキーは，A[i].key でアクセス
- 時間計算量：
  - ソートする項目の個数（入力サイズ）nの関数 T(n)（以下の総和）
    - キーの比較回数
    - 項目の代入回数

ソーティング法

- 内部ソーティング法
  - 高速のランダムアクセスできる内部記憶（主記憶など）上のソート=> 一つの配列
    - 単純法
    - シェルソート
    - ヒープソート
    - クイックソート
    - 木ソート
    - 基数ソート

どこでも同じ時間でアクセス

- 外部ソーティング法
  - 大きな領域を持ち，順次アクセスしかできない外部記憶（磁気テープなど）上のソート => 複数の配列（たくさんのメモリを使う）
    - マージソート



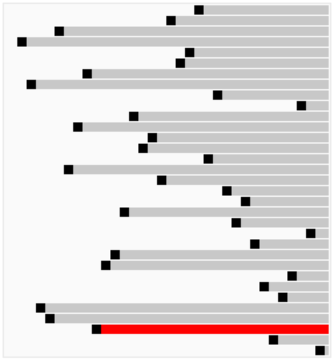
どこかひとつずつ順次アクセス

3つの単純法

単純法

- 3種類の方法
  - 単純交換法（バブルソート）
  - 単純選択法
  - 単純挿入法
- いずれも時間計算量は
  - $O(n^2)$

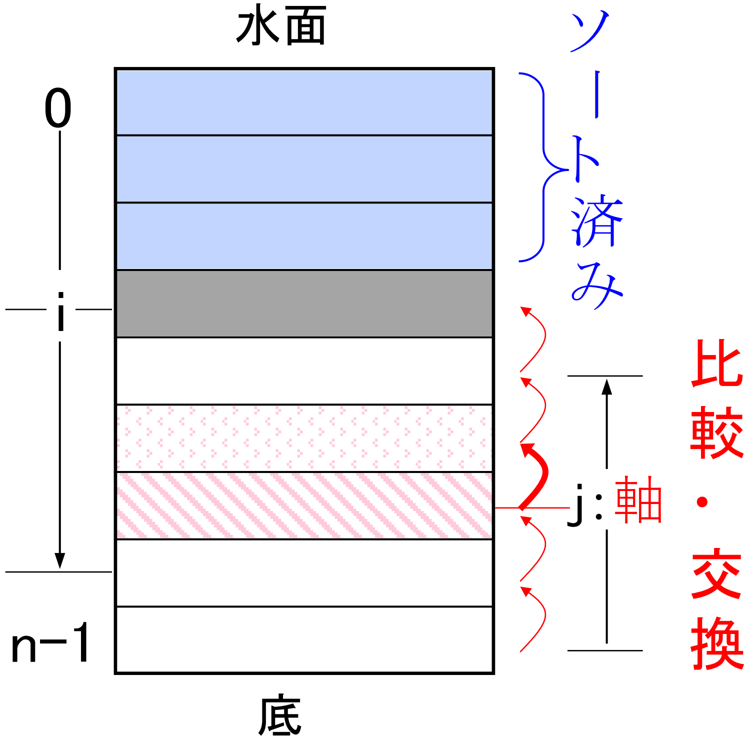
単純交換法（バブルソート）



バブルっぽい図（大きい値を上へ）
Wikipediaより

- アルゴリズム
  1. 一番下（底）の要素を軸
  2. ひとつ上の要素と比較
     - 大きい時：上下を**交換**する
  3. 軸を一つ上に
     1. 水面ではないなら：2へ
     2. 水面なら：一番上を除いて1.へ

- 泡が上っていくように見える



バブルソートの第iフェーズ

バブルソート（C言語）

```
void BubbleSort(int A[]) {
    int i,j,temp;
    for(i=0; i<(n−1); i++)
      for(j=n−1; j>i; j−−)
          if( A[j−1]>A[j] ){
              temp=A[j−1];
              A[j−1]=A[j];
              A[j]=temp; }
 }
```

i=0〜n−2の繰返し(iの設定・比較:2回)
j=n−1〜i+1のn−i−1回繰返し(jの設定・比較: 2回)
比較1回 必ず行われる
代入3回 最悪時必ず交換

最良時はこれらの代入は行われない

- 入力サイズ:ソートされる配列要素数 n

最悪時間計算量 $\sum_{i=0}^{n-2}(2+6(n-i-1))= 3n^2-n-2$

最良時間計算量 $\sum_{i=0}^{n-2}(2+3(n-i-1))= \frac{3}{2}n^2+\frac{1}{2}n-2$

時間計算量は, $O(n^2)$
領域計算量は、配列要素数のnと変数 i, j, temp より n+3 : O(n)

バブルソートの第０フェーズ

配列 44 55 06 42 94 18 12 67

第０フェーズ

0	i→44	44	44	44	44	44	**44**	06
1	55	55	55	55	55	**55**	**06** ←j	44
2	06	06	06	06	**06**	**06** ←j	55	55
3	42	42	42	**42**	**12** ←j	12	12	12
4	94	94	**94**	**12** ←j	42	42	42	42
5	18	**18**	**12** ←j	94	94	94	94	94
6	**12**	**12** ←j	18	18	18	18	18	18
7	**67** ←j	67	67	67	67	67	67	67
	第1ステップ						第7ステップ	結果

比較

比較交換

バブルソートの過程

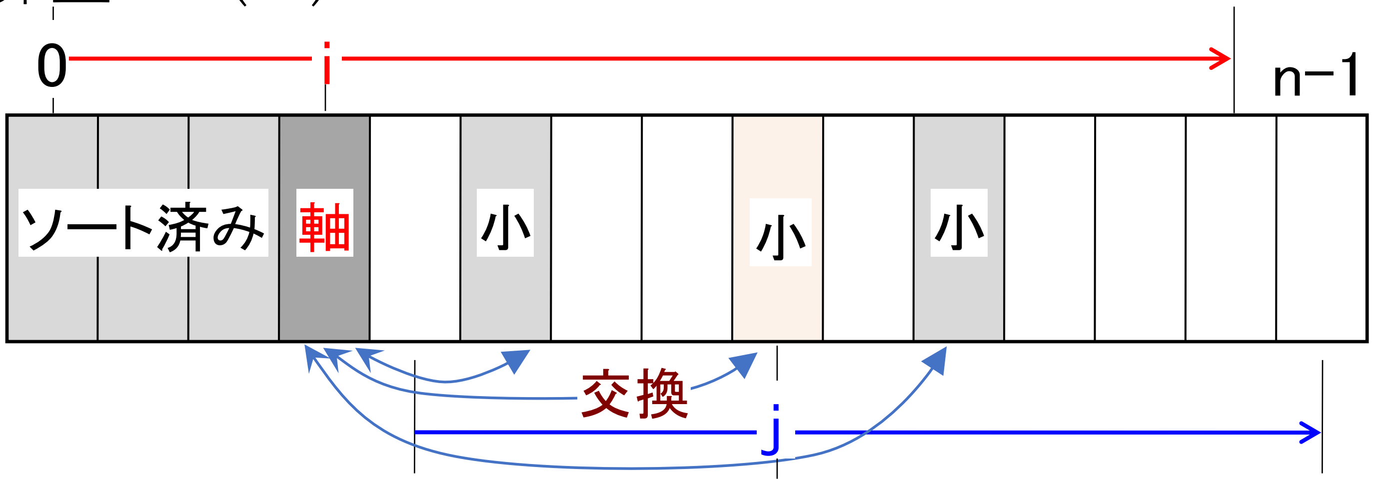
配列 44　55　06　42　94　18　12　67

0	i→44	06	06	06	06	06	06	06
1	55	i→44	12	12	12	12	12	12
2	06	55	i→44	18	18	18	44	44
3	42	12	55	i→44	42	42	55	55
4	94	42	18	55	i→44	44	44	44
5	18	94	42	42	55	i→55	55	55
6	12	18	94	67	67	67	i→67	67
7	67	67	67	94	94	94	94	94

第０フェーズ　　第３フェーズ　　第６フェーズ　結果

単純選択法

- 軸 A[i] を設定しA[0]からA[n-1]まで以下を行う
- 軸 A[i] 以降の要素 A[j] として，A[i+1]からA[n-1]まで順に調べていき，A[j]のキーが軸のキーA[i].key未満のとき，A[j] を選択し軸A[i] と交換する
- 比較は必ず$(n^2-n)/2$回行われる
  - 時間計算量：$O(n^2)$



単純選択法の第０フェーズ

i=0 j

44 55 06 42 94 18 12 67

軸

比較

44 55 06 42 94 18 12 67

比較 選択し交換

06 55 44 42 94 18 12 67

06 55 44 42 94 18 12 67

06 55 44 42 94 18 12 67

06 55 44 42 94 18 12 67

06 55 44 42 94 18 12 67

最小

第0フェーズ

単純選択法の第1フェーズ

	i=1	j					
06	55	44	42	94	18	12	67
	軸						
06	44	55	42	94	18	12	67
06	42	55	44	94	18	12	67
06	42	55	44	94	18	12	67
06	18	55	44	94	42	12	67
06	12	55	44	94	42	18	67

つぎに
小さい

第1フェーズ

単純選択法のプログラムと計算量

```
for(i=0; i<(n-1); i++){
   for(j=l; j<=(n-1); j++){
      if(A[i].key>A[j].key){
         temp=A[j];
         A[j]=A[i];
         A[i]=temp}}}
```

A[i] を軸とする（iの**設定**と**比較: 2回**）

n-i- 1 回繰り返す（jの**設定**と**比較: 2回**）

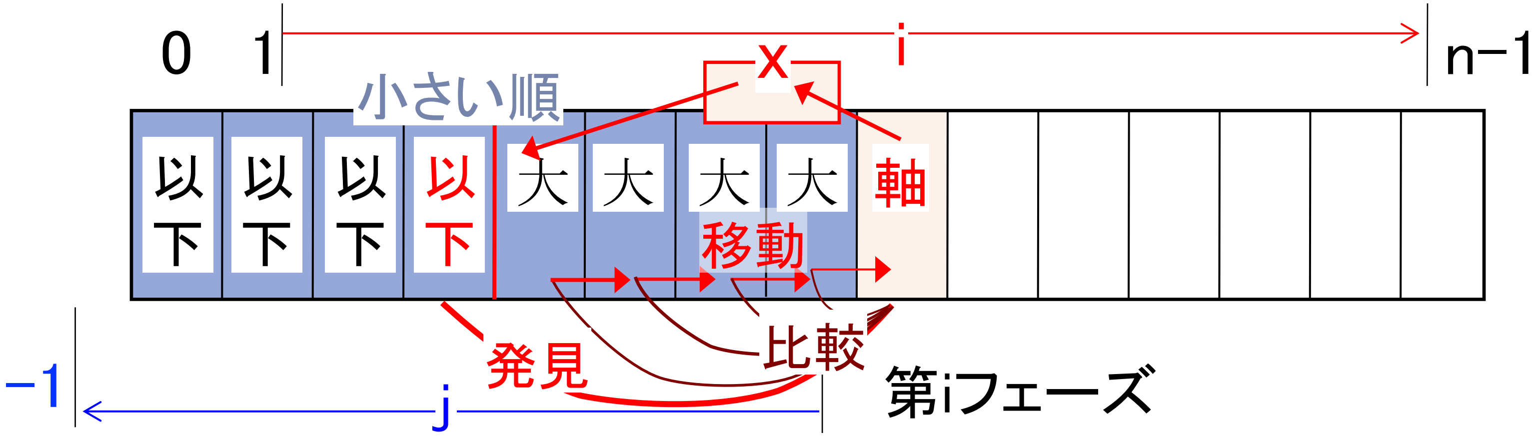
比較1回が必ず行われる

最悪3回の交換（最良は0回）

- 時間計算量
  - 最悪の場合：軸A[i]とすべてのA[j]が交換
  - 最良の場合：軸A[j]とA[i]は交換されない（比較は必ず行われる）
  - $T_{max}(n) = \sum_{i=0}^{n-2}\big(2 + 6(n - i - 1)\big) = 3n^2 - n - 2$
  - $T_{min}(n) = \sum_{i=0}^{n-2}\big(2 + 6(n - i - 1)\big) = 3n^2 - n - 2$
  - どちらもO(n^2)
- 領域計算量
  - n+3（n,temp, i, j）
  - O(n)

単純挿入法

- 軸 A[i] を設定しA[0]からA[n-1]まで以下を行う
- A[0]から軸A[i-1]までは小さい順に並んでいる
- jをi-1から一つずつ減らしていき，軸A[i].key以下のA[j].keyが見つかったらA[j]の右にA[i]を**挿入**する
- jが負（-1）になったら，A[0]からA[i-1]のkeyがすべてA[j].keyより大きい場合なので A[i]をA[0]先頭に**挿入**する.



単純挿入法のプログラムと計算量

```
for(i=1; i<=n-1; i++){
        x=A[i];
        j=i-1;
        while(j>=0 && x.key<A[j].key){
                A[j+1]=A[j];
                j=j-1;}
        A[j+1]=x;}
```

iの**設定**，**比較**：2回
軸の**設定**：1回
代入：1回
最悪i回のループ（比較2回，脱出時1回）
代入
代入

- 軸の挿入

- while文
  - A[0]からA[i-1]のkeyがA[i].keyより大きいときは，jが-1となるが，&&は左演算子が不成立のときは右演算子は評価しない（A[-1]は評価しない）
  - 最悪：A[0]までの比較・代入が**i回**，whileループの脱出の判定まで含め4i+1回
  - 最良：whileループはA[i-1]との**比較**ですぐ止まるので脱出判定の**1回**のみ
- 時間計算量
  - $T_{max}(n) = \sum_{i=0}^{n-1}(2+1+1+4i+1+1) = 2n^2+2n-4 = O(n^2)$
  - $T_{min}(n) = \sum_{i=0}^{n-1}(2+1+1+0+1+1) = 6(n-1) = O(n)$

単純挿入法の各フェーズ

x 55 i=1

第1フェーズ j設定　44　55　06　42　94　18　12　67

軸

j=0

比較:挿入場所j+1発見

軸以下の要素A[0]発見　44　55　06　42　94　18　12　67

55　軸

j<0　比較　i=2

第2フェーズ j設定

jが負なのでA[0]に挿入　44　55　06　42　94　18　12　67

軸

j=1

06

発見　比較 i=3

第3フェーズ j設定　06　44　55　42　94　18　12　67

以下の要素A[0]発見　42　軸

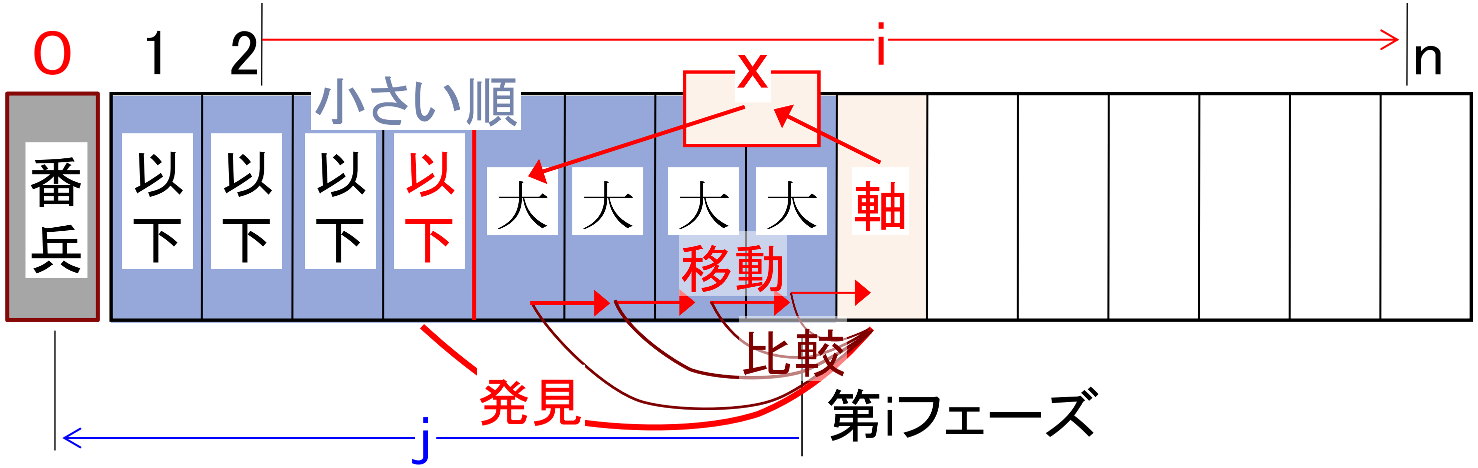
j=2

A[1]に挿入　06　42　44　55　94　18　12　67

軸

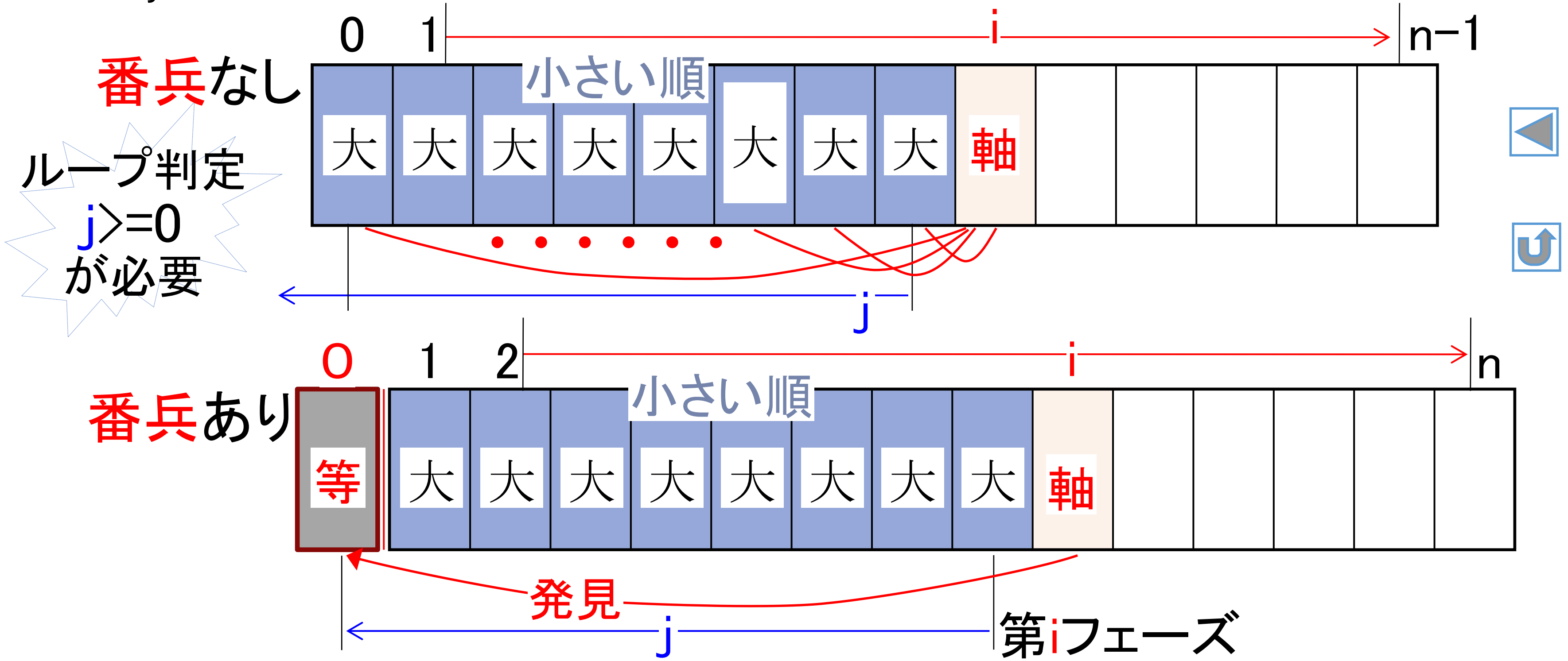
番兵を用いた単純挿入法

- 配列Aをn+1個の要素
  - A[1]からA[n]がソート対象，A[0]は「番兵」としてindex **jの監視**に利用する
- 番兵A[0] に 軸[i] を設定
- A[1]からA[i-1]は小さい順に並んでいる
- jをi-1から一つずつ減らしていき，軸A[i].key以下のA[j].keyが見つかったらA[j]の右にA[i]を**挿入**する
- 番兵がいるためjは0より小さくならない（jの判定がいらない）



番兵の働き

- index jはどこで止まるか？
  - jは軸のキー以下の要素が見つかったとき停止する
  - i未満のすべてのA[j]が軸のキーより大きいときには
    - jを減らす時にwhileループの条件判定j>=0が必要だが，番兵を使うと最悪番兵でjを停止できる



番兵の単純挿入法の各フェーズ

番兵 x 55

i=2

番兵の設定 55 44 55 06 42 94 18 12 67

軸

j=1

比較:挿入場所j+1発見

軸以下を探索 55 44 55 06 42 94 18 12 67

軸

55

発見 比較 i

番兵の設定
軸以下を探索 06 44 55 06 42 94 18 12 67

軸

06

発見 比較 i

番兵の設定
軸以下を探索 42 06 44 55 42 94 18 12 67

軸

42

第4要素まで
小さい順 42 06 42 44 55 94 18 12 67

軸

番兵あり単純挿入法のプログラムと計算量

`for(i=2;i<=n; i++){`	iの設定，比較：2回	
`    x = A[i];`	軸の設定	
`    A[0] = x;`	番兵の設定	3回
`    j= i – 1;`	代入	
`    while(x.key<A[j].key){`	最悪i回	
`        A[j+1] = A[j];`	代入	
`        j=j+1;}`	代入	
`    A[j+1]=x;}`	軸の挿入	

- whileループ
  - 最悪：番兵までの比較がi回より，whileループによるすべての代入は 2(i – 1)回
  - 最良：whileループはj= i-1 ですぐとまる．比較1回
- 時間計算量
  - $T_{max}(n) = \sum_{i=2}^{n}(2+3+i+2(i-1)+1) = \frac{3}{2}n^2 + \frac{11}{2}n - 7 = O(n^2)$
    $T_{min}(n) = \sum_{i=2}^{n}(2+3+1+1) = 7(n-1) = O(n)$

まとめ：二つの単純挿入法の計算量

- 時間計算量
  - 最悪時間計算量：$O(n^2)$
  - 最良時間計算量：$O(n)$

- 領域計算量
  - $O(n)$

- すべての単純法は
  - 時間計算量がよくない：$O(n^2)$
  - 領域計算量は良好：$O(n)$ => 入力配列が小さい場合は有効

ヒープソート

ヒープ構造を利用したソート

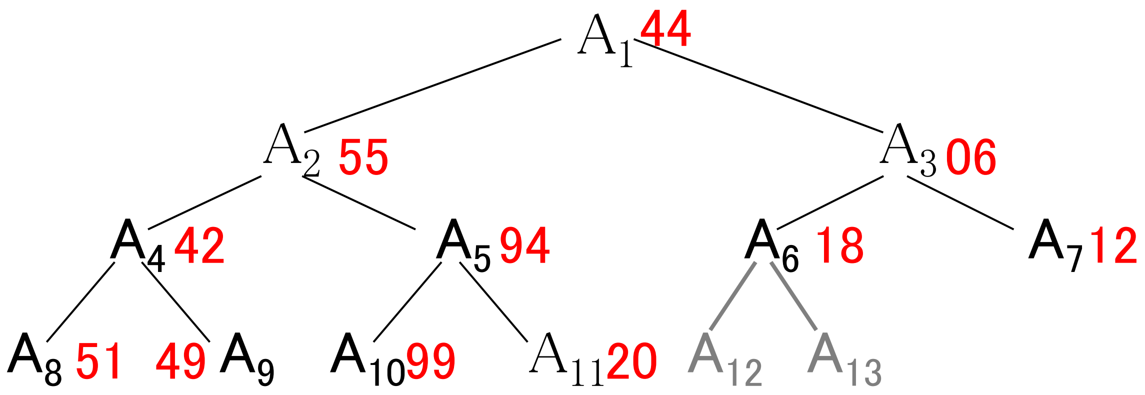
完全2分木とは

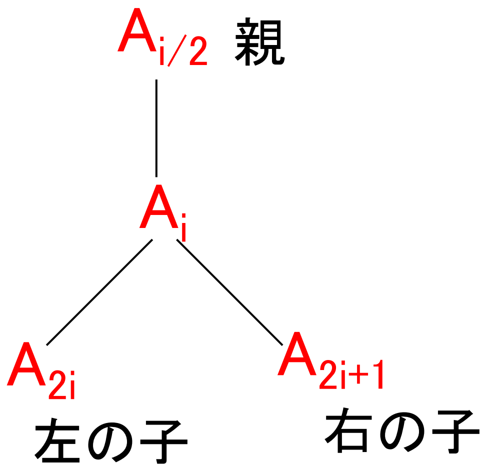
- 2分木：全長点の子数が最大2個の根付き木
- 完全2分木：葉以外の頂点の子がちょうど2個ですべての葉の高さが等しい2分木

- 配列 => 木構造（完全2分木）
- 配列Aは A[1]からA[n] （A[0]は使わない）

頂点iにおいて
- 親：i/2
- 左の子：2i,
- 右の子：2i+1

	1	2	3	4	5	6	7	8	9	10	11
A	44	55	06	42	94	18	12	51	49	99	20



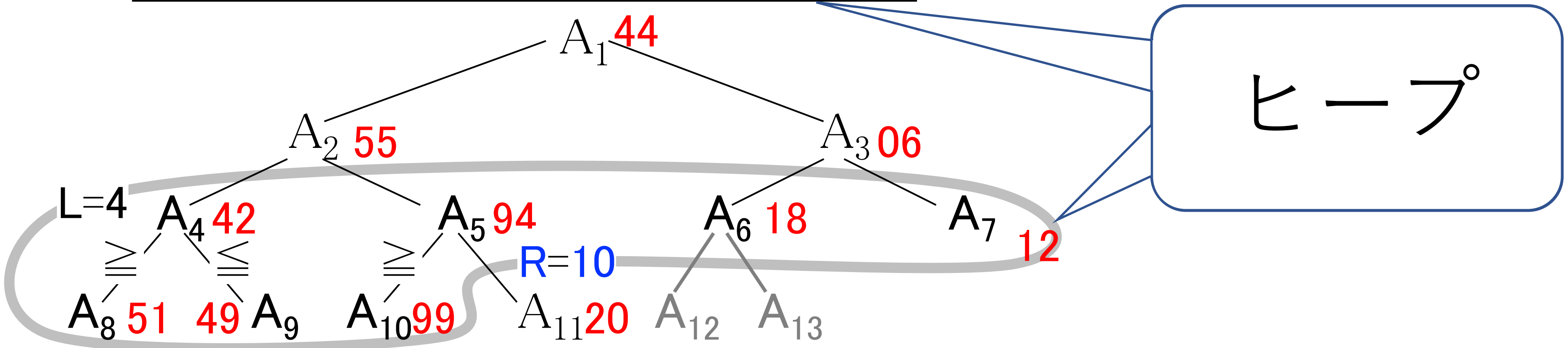


ヒープとは

- 完全2分木の各頂点はデータをひとつずつ持ち，必ず「ヒープ条件」を満たす
- ヒープ条件（最小ヒープ）
  - ある頂点のデータの値は，その親の持つデータの値以上である

  - 配列A（index 1からn）の部分列A[L], A[L+1], …, A[R]において
  - すべての i=L, …, Rに対して以下を満たす
    - 2i≦Rならば，A[i].key≦A[2i].key
    - 2i+1≦Rならば，A[i].key≦A[2i+1].key

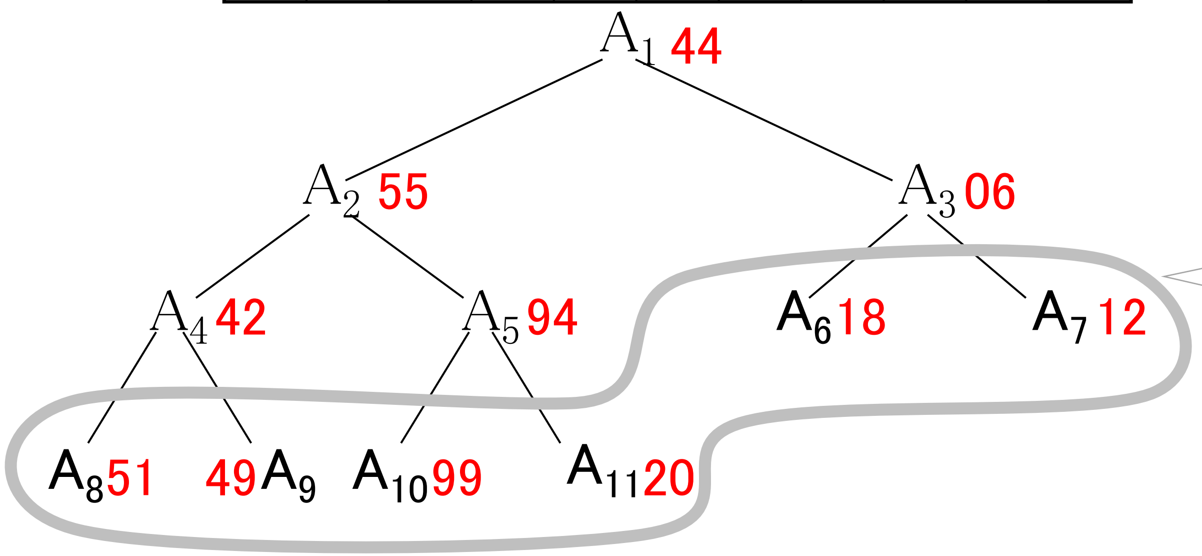
	1	2	3	4	5	6	7	8	9	10	11
A	44	55	06	42	94	18	12	51	49	99	20



ヒープ化：最初からヒープ

- 部分列A[6],…, A[11]を考える
  - i=L(6),…, R(11) のヒープ条件
    - 2i≦Rならば，A[i].key≦A[2i].key
    - 2i+1≦Rならば，A[i].key≦A[2i+1].key
  - 2i≦R， 2i+1≦Rが成立しないので，後半条件は無視しヒープ条件が成立する

	1	2	3	4	5	6	7	8	9	10	11
A	44	55	06	42	94	18	12	51	49	99	20

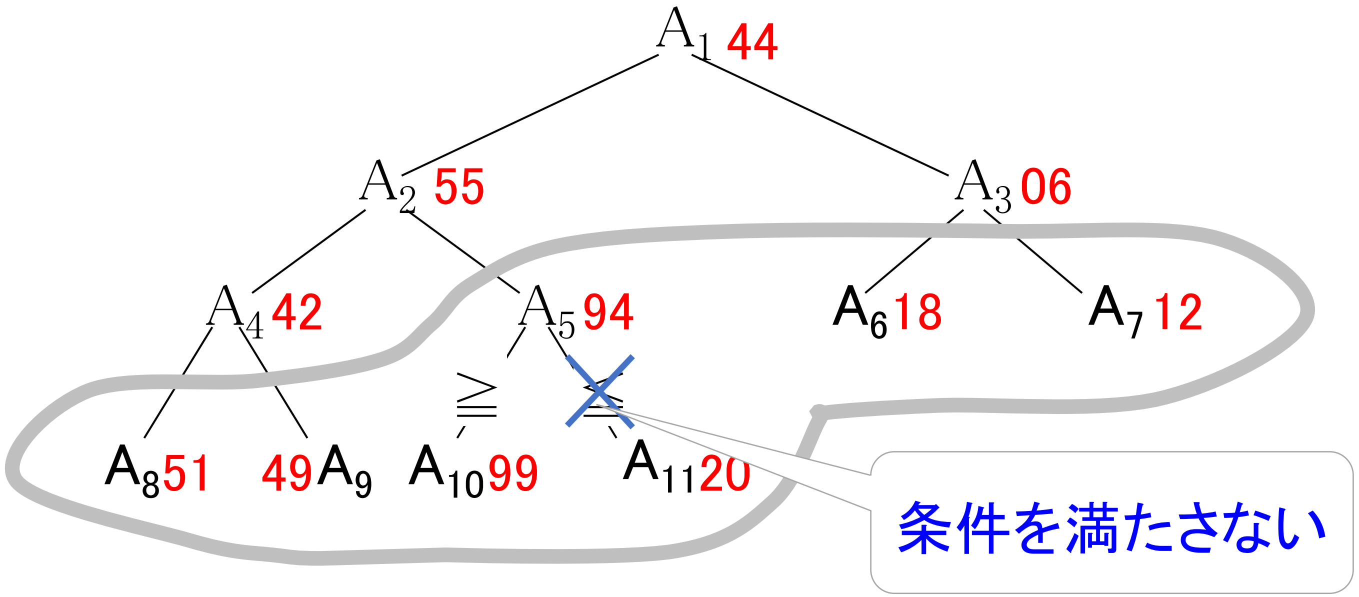


配列の右半分
A[6]…A[11]は
必ずヒープ

葉はヒープ

ヒープ化：ヒープの配列を左に拡大

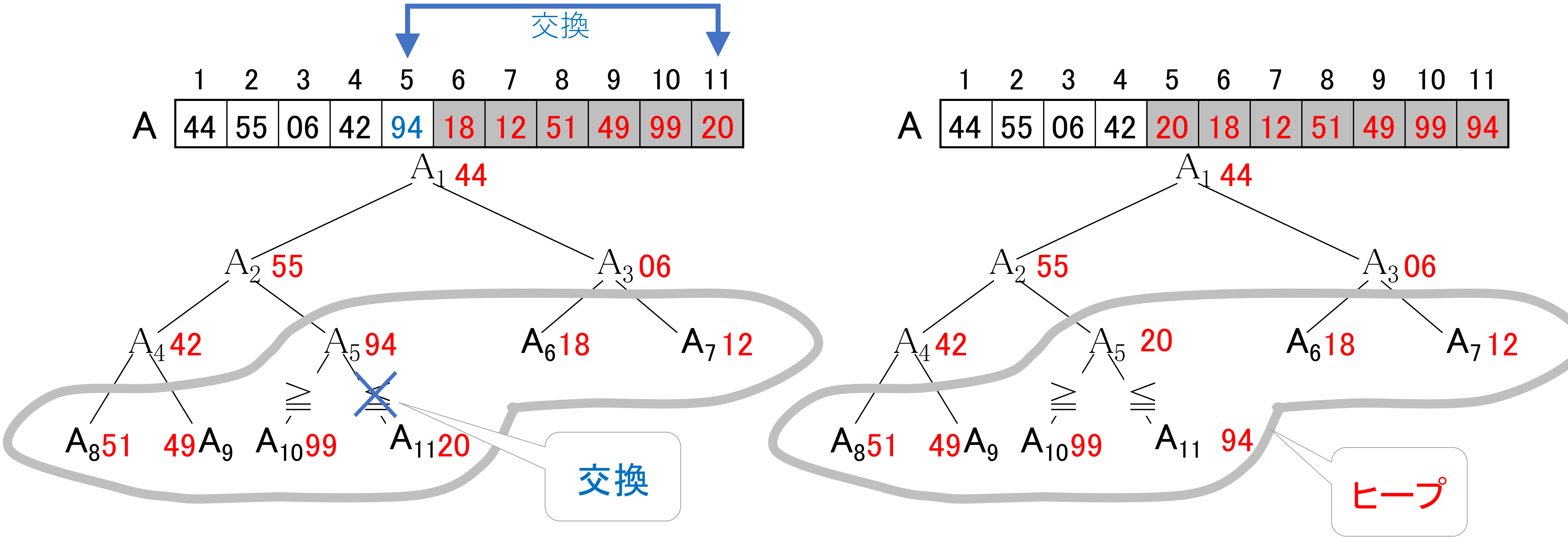
	1	2	3	4	5	6	7	8	9	10	11
A	44	55	06	42	94	18	12	51	49	99	20



ヒープ化：ふるい落とし

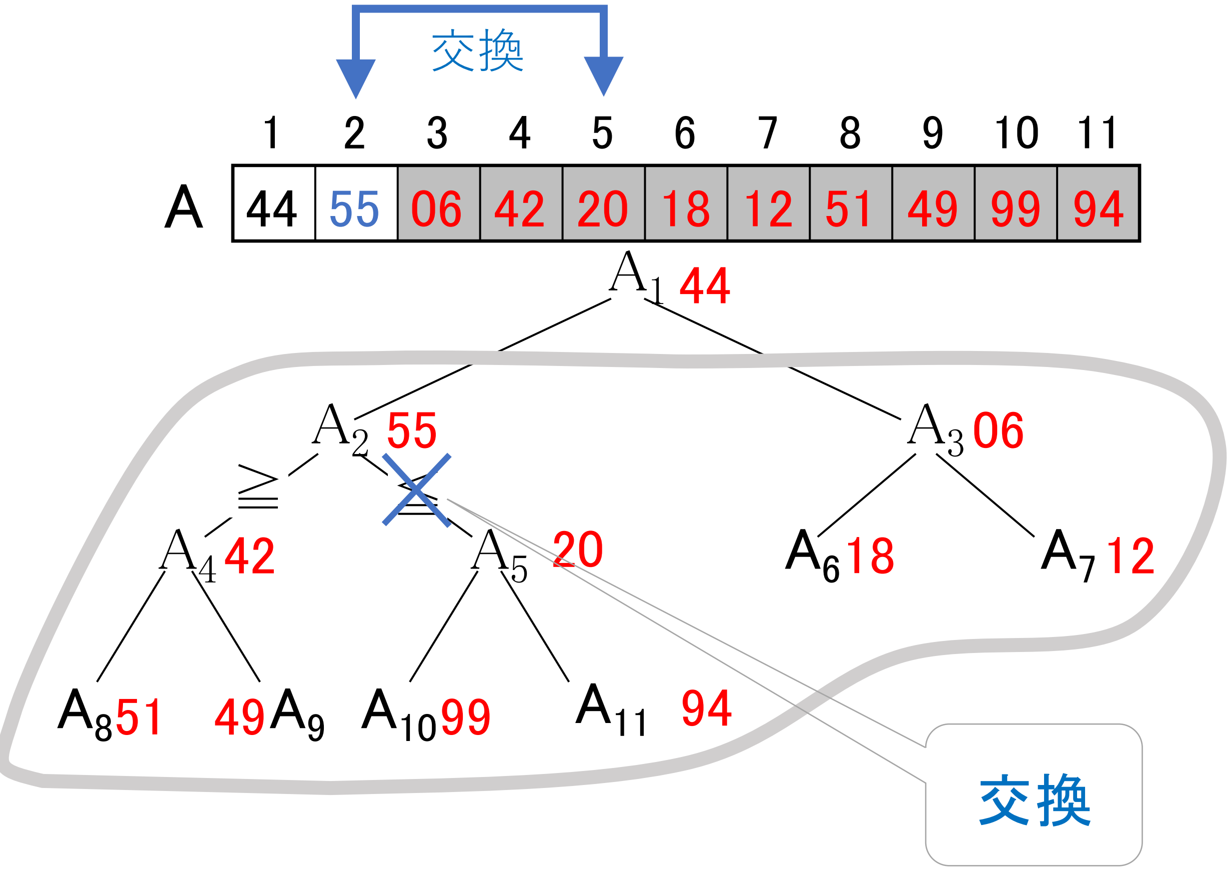
- ヒープA[L+1],…, A[R]が与えられているとき
- A[L]を追加し，A[L], A[L+1],…, A[R]をヒープになるようにする
- この操作を A[L]のふるい落とし（シフト）と呼ぶ
  1. i=L
  2. 子の数で場合分け
     - 無し：　　　　2*i > Rならば，おわり
     - 左の子だけ：2*i = R ならば，j=2*i
     - 左右両方：　2*i+1≦Rならば，キーの小さい方のインデックスをjとする
  3. A[i].keyとA[j].keyを比較
     - A[i].key≦A[j].key ならば，おわり
     - そうでなければ，A[i]とA[j]を交換して，処理2へ

ヒープ化：ふるい落とし

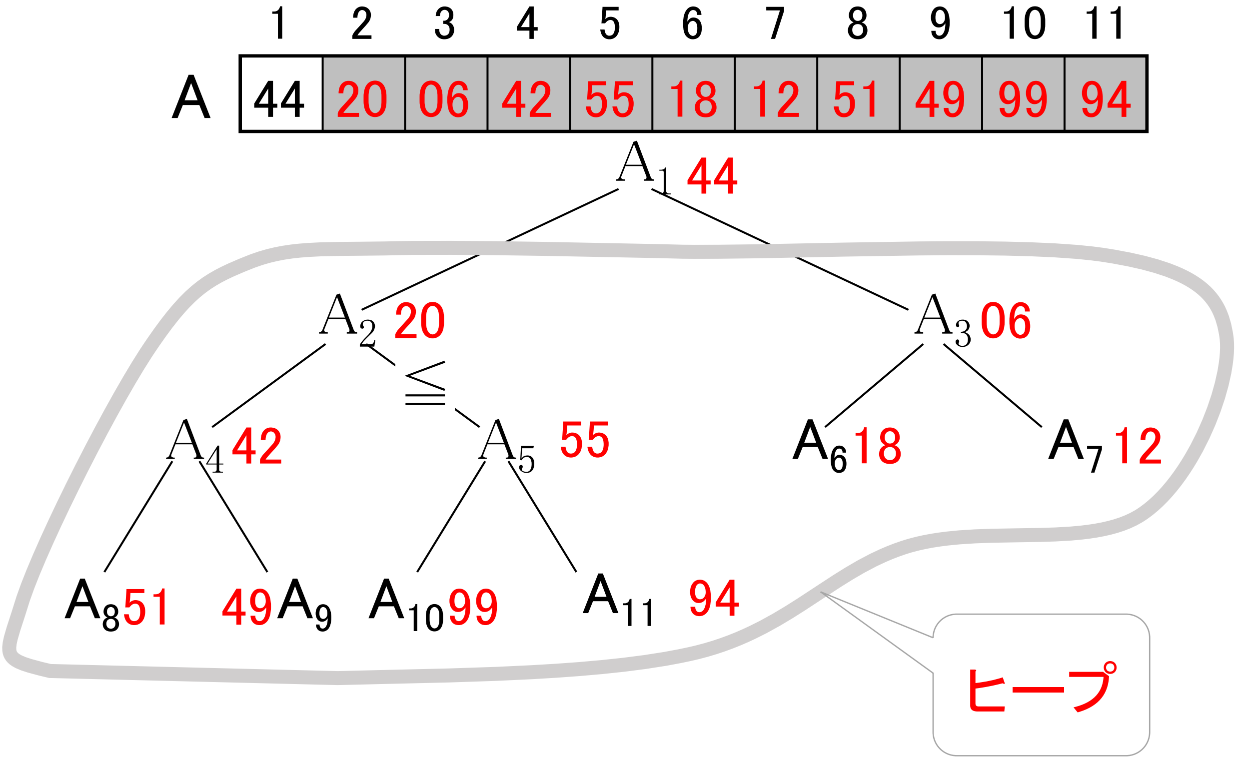


交換
1 2 3 4 5 6 7 8 9 10 11
A 44 55 06 42 94 18 12 51 49 99 20
A1 44
A2 55
A3 06
A4 42
A5 94
A6 18
A7 12
A8 51
49 A9
A10 99
A11 20
≧
≦
交換
1 2 3 4 5 6 7 8 9 10 11
A 44 55 06 42 20 18 12 51 49 99 94
A1 44
A2 55
A3 06
A4 42
A5 20
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
≧
≦
ヒープ

ヒープ化：ふるい落とし

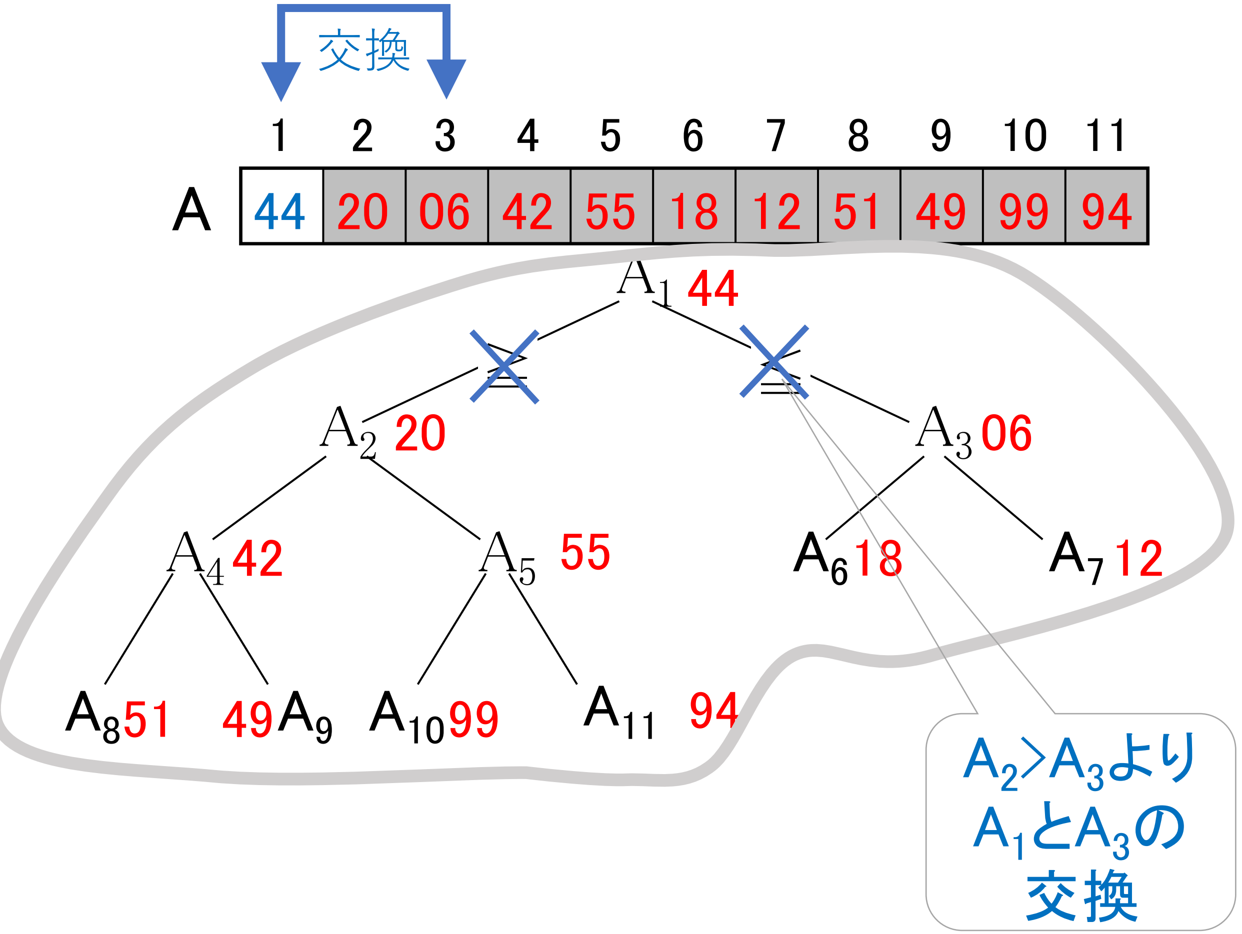


交換
1 2 3 4 5 6 7 8 9 10 11
A 44 55 06 42 20 18 12 51 49 99 94
A1 44
A2 55
A3 06
≧
≦
A4 42
A5 20
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
交換

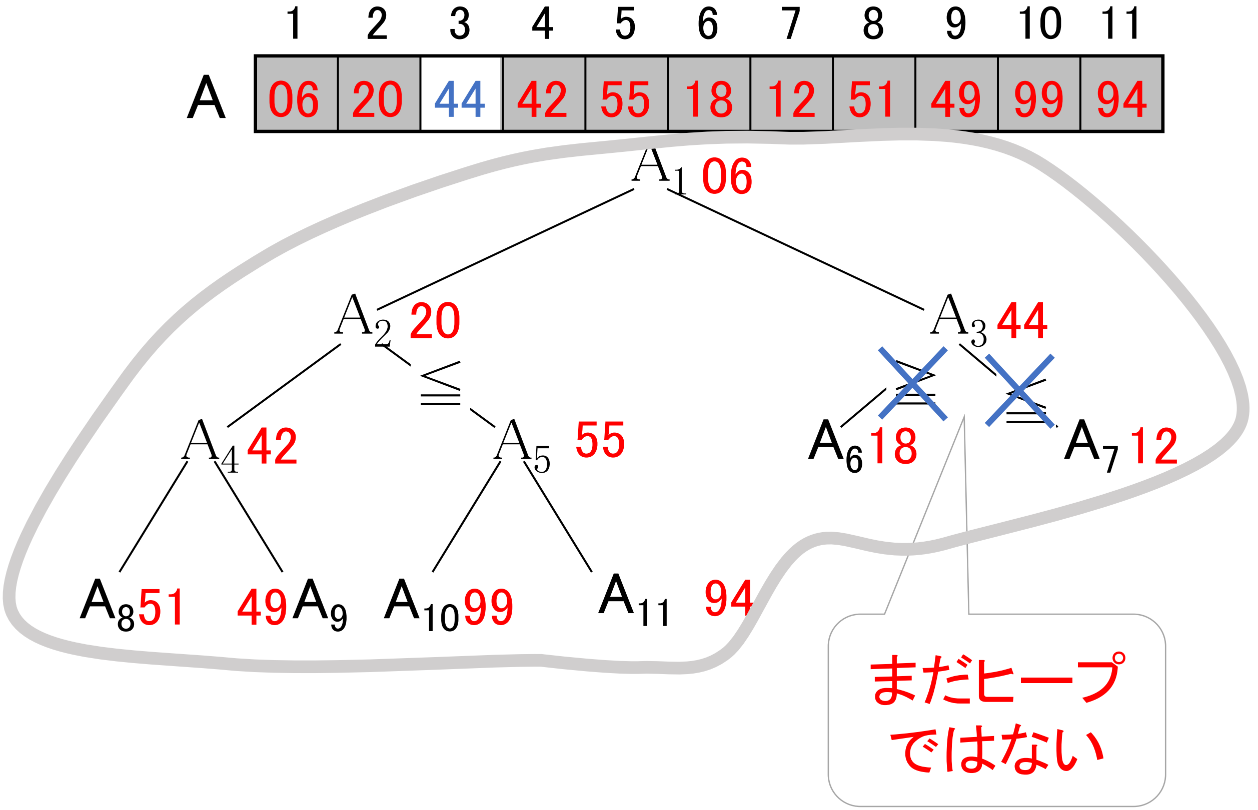


1 2 3 4 5 6 7 8 9 10 11
A 44 20 06 42 55 18 12 51 49 99 94
A1 44
A2 20
A3 06
≦
A4 42
A5 55
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
ヒープ

ヒープ化：ふるい落とし

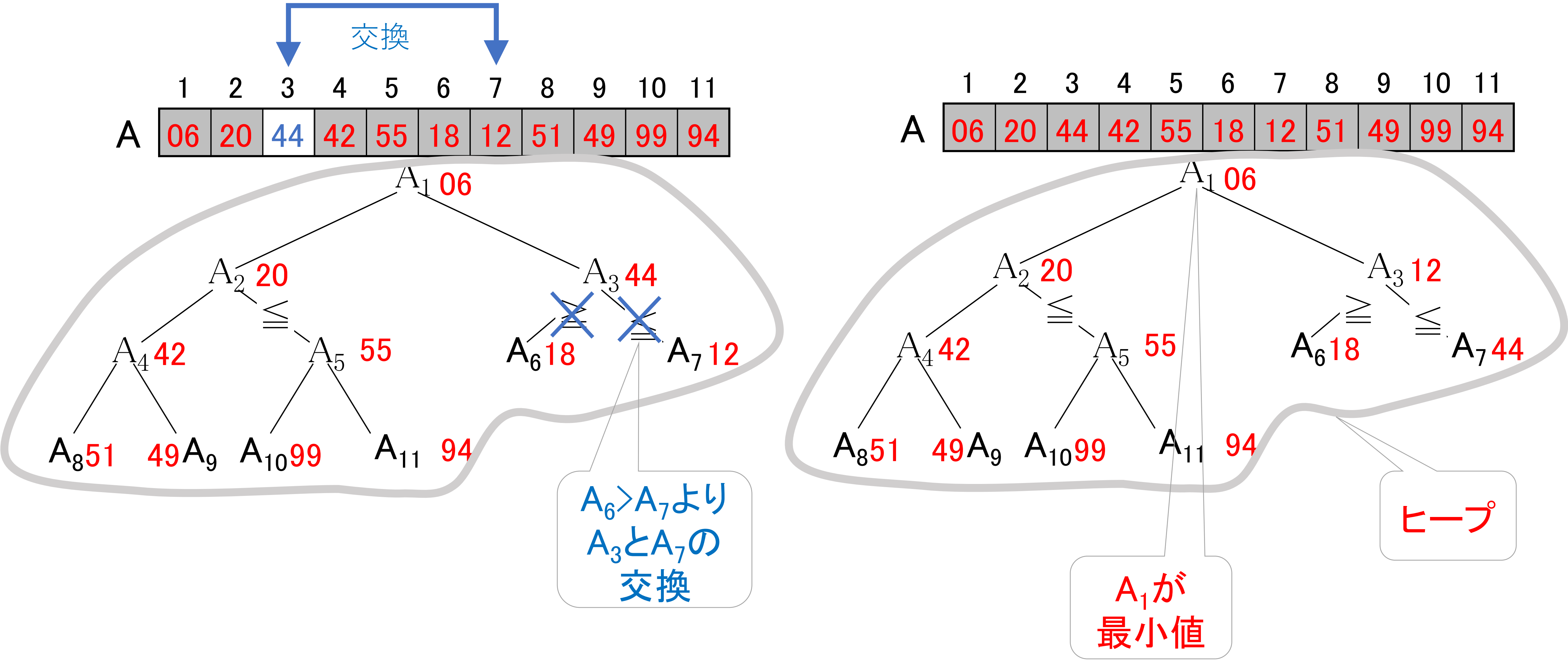


交換
1 2 3 4 5 6 7 8 9 10 11
A 44 20 06 42 55 18 12 51 49 99 94
A1 44
A2 20
A3 06
A4 42
A5 55
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
A2>A3より
A1とA3の
交換



1 2 3 4 5 6 7 8 9 10 11
A 06 20 44 42 55 18 12 51 49 99 94
A1 06
A2 20
A3 44
A4 42
A5 55
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
まだヒープ
ではない

ヒープ化：ふるい落とし



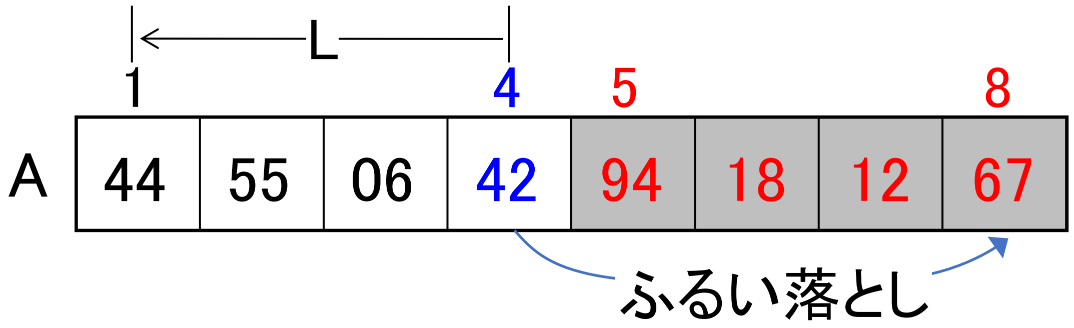
交換
1 2 3 4 5 6 7 8 9 10 11
A 06 20 44 42 55 18 12 51 49 99 94
A1 06
A2 20
A3 44
A4 42
A5 55
A6 18
A7 12
A8 51
49 A9
A10 99
A11 94
A6>A7より
A3とA7の
交換
1 2 3 4 5 6 7 8 9 10 11
A 06 20 44 42 55 18 12 51 49 99 94
A1 06
A2 20
A3 12
A4 42
A5 55
A6 18
A7 44
A8 51
49 A9
A10 99
A11 94
ヒープ
A1が
最小値

ヒープソート

- ヒープソートの原理
  1. ヒープの構成
     - 配列A[1],…, A[n] をヒープにする
  2. ソート列の構成
     - ヒープ A[1],…, A[n] をソート列にする

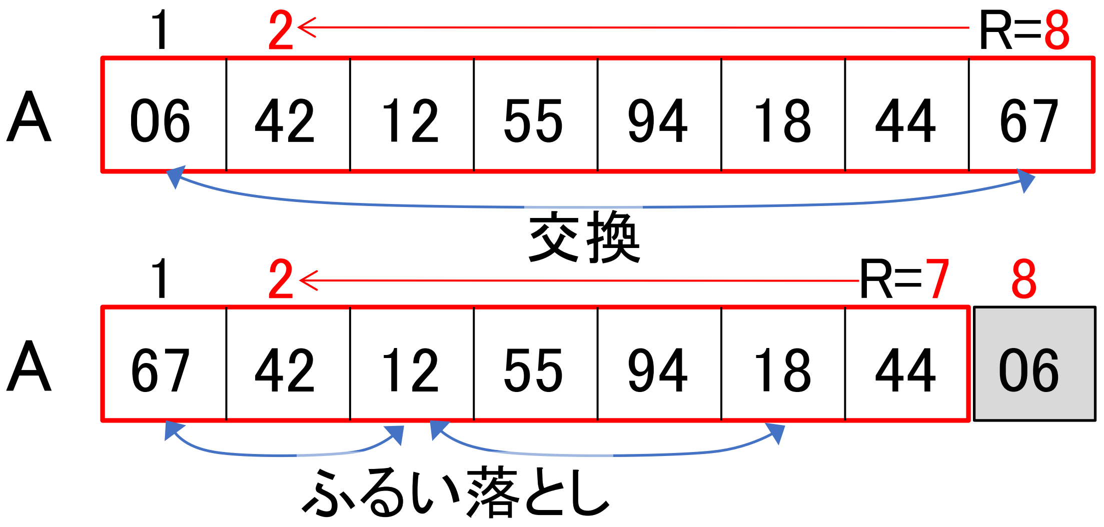
1. ヒープの構成

- 配列A[1],…, A[n] をヒープにする
  - A[n/2 + 1], …, A[n] は子がいないので始めからヒープ
  - A[L], A[L+1],…, A[n] をヒープにするためにA[L]のふるい落としを行う（Lは n/2 からデクリメントして 1 まで）



2. ソート列の構成

- ヒープA[1],…, A[R] が与えられたとき，A[1]とA[R]を交換する.
- 交換直後はA[R]がAの要素で最も小さいキーを持つ
- A[2],…, A[R-1]はヒープであるので，A[1]のみふるい落としを行いA[1],…, A[R-1]をヒープにする

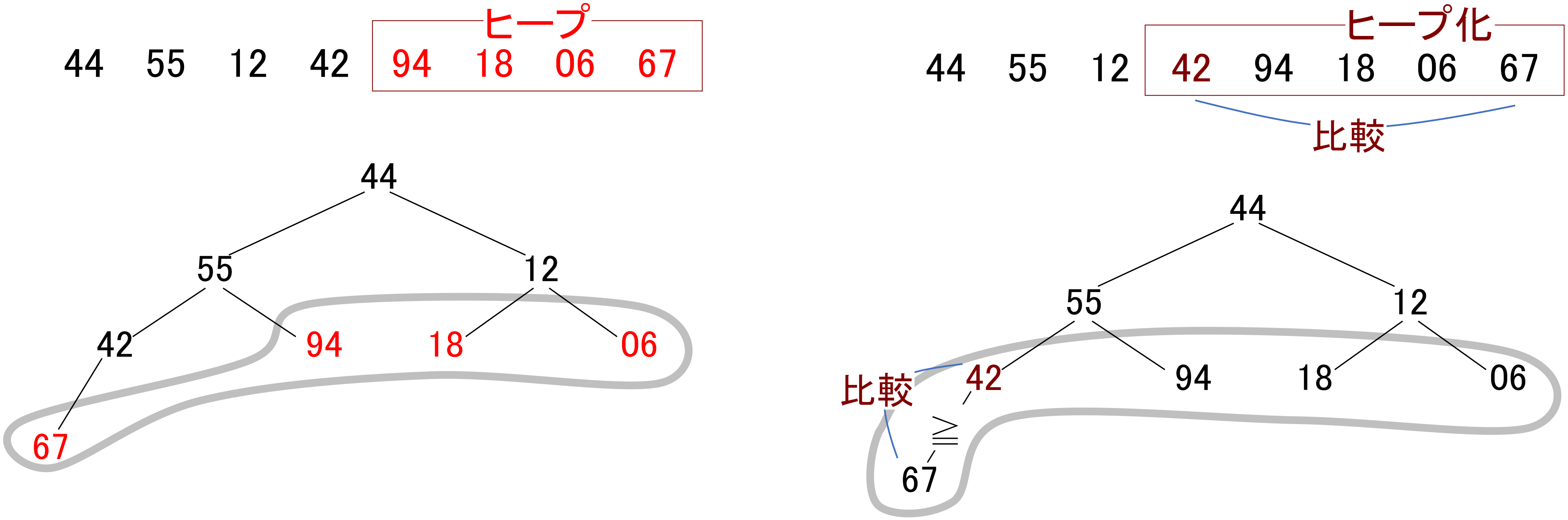


C言語によるヒープソート

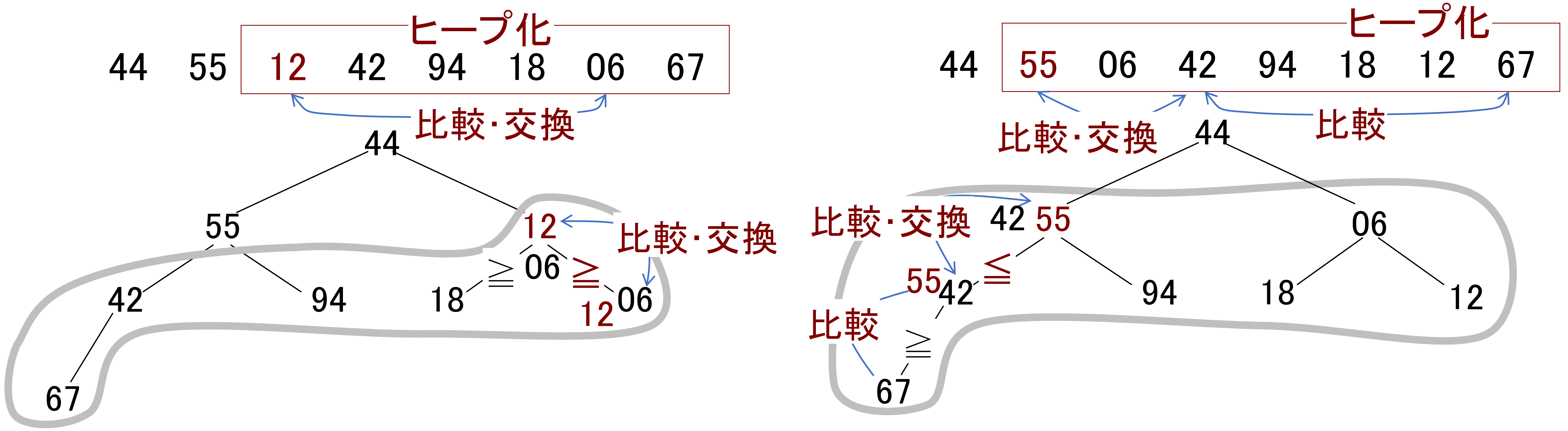
```
/* shift down A[L] into A[L+1],…, A[R] */
void shift(int L, int R){
    int i, j;  item x;
    i=L;  j=2*i;  x=A[i];
    while(i<=R){
        if(j<R){
            if(A[j].key>A[j+1].key){
                j=j+1;}}
        if(x.key<=A[j].key){
            break;
        A[i]=A[j];  i=j;  j=2*i;
    }
    A[i]=x;
}
```

```
void heapsort(){
    int L, R; item x;
    L=(n/2)+1;
    R=n;
    while(L>1){
            L=L-1;
            shift(L, n);}        /* A is heaped */
    while( R>1){
            x=A[1];
            A[1]=A[R];
            A[R]=x;
            R=R-1;
            shift(1, R);}
}
```

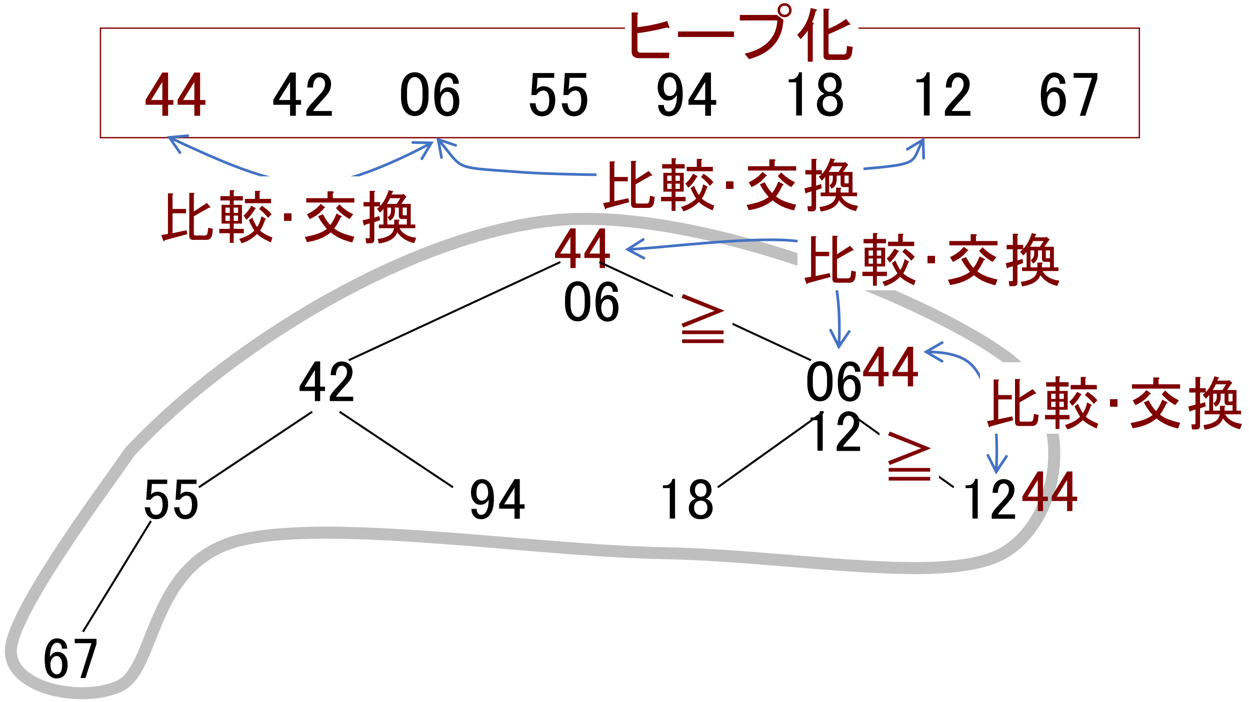
動作例：44, 55, 12, 42, 94, 18, 06, 67
ヒープを構成



動作例：44, 55, 12, 42, 94, 18, 06, 67
ヒープを構成

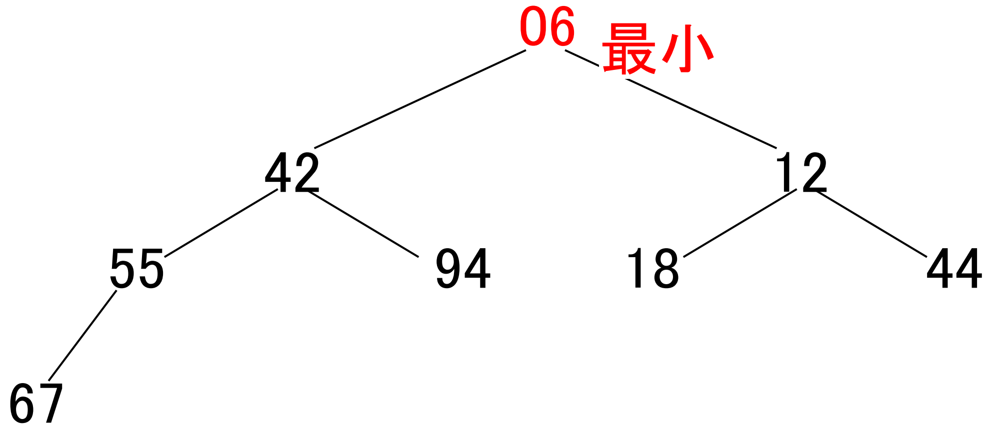


動作例：44, 55, 12, 42, 94, 18, 06, 67
ヒープを構成

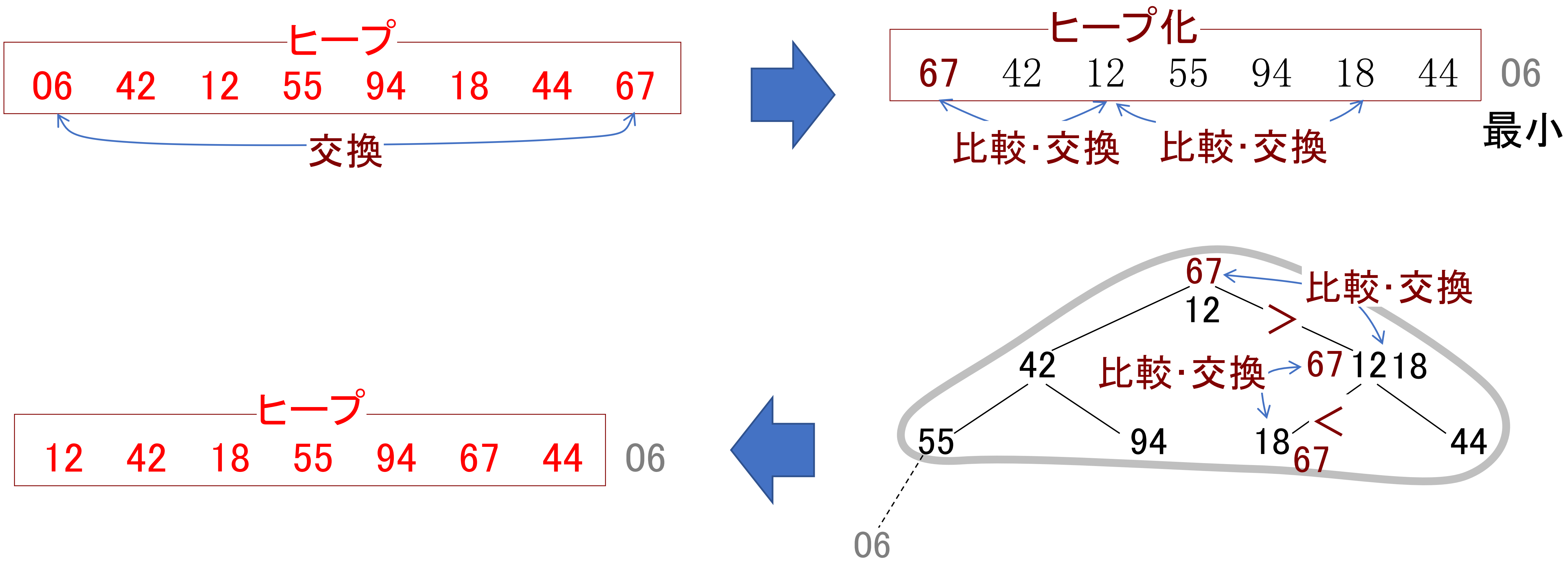


ヒープ

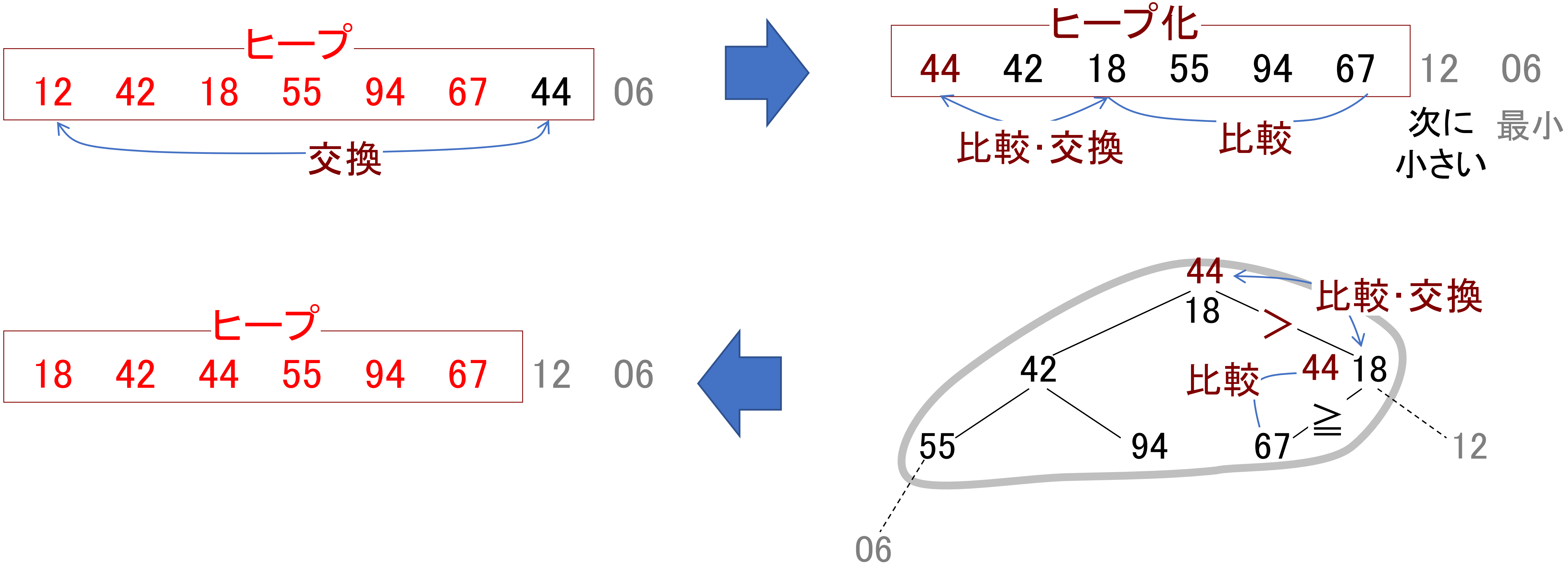
06 42 12 55 94 18 44 67



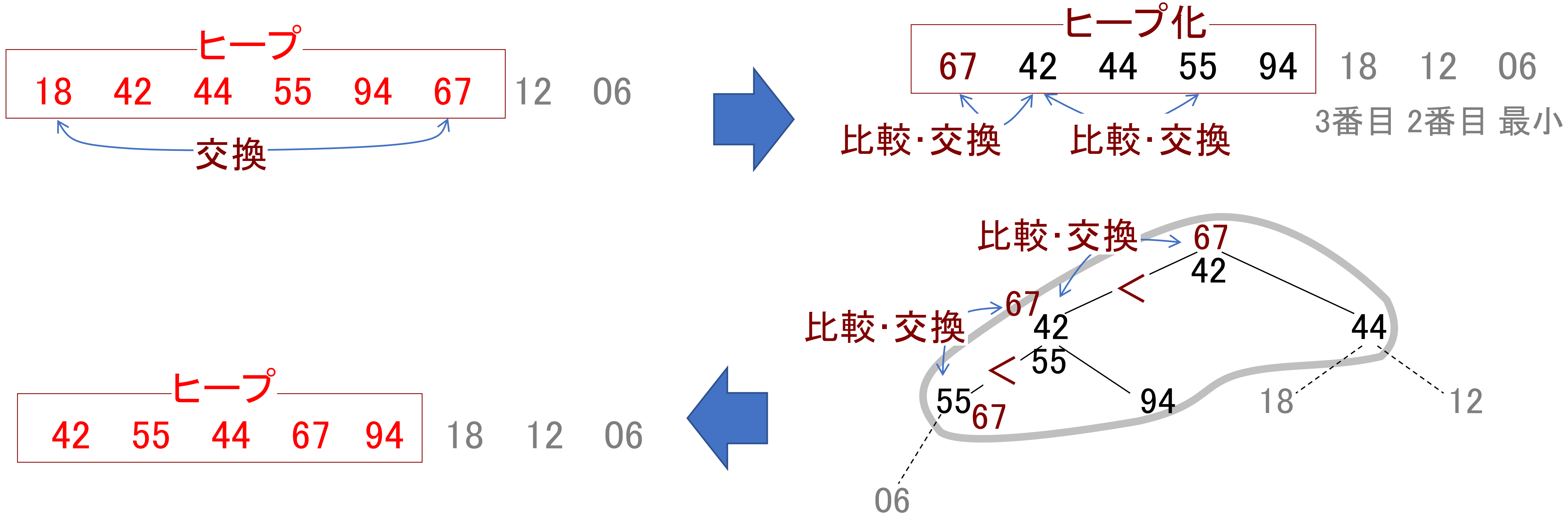
動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



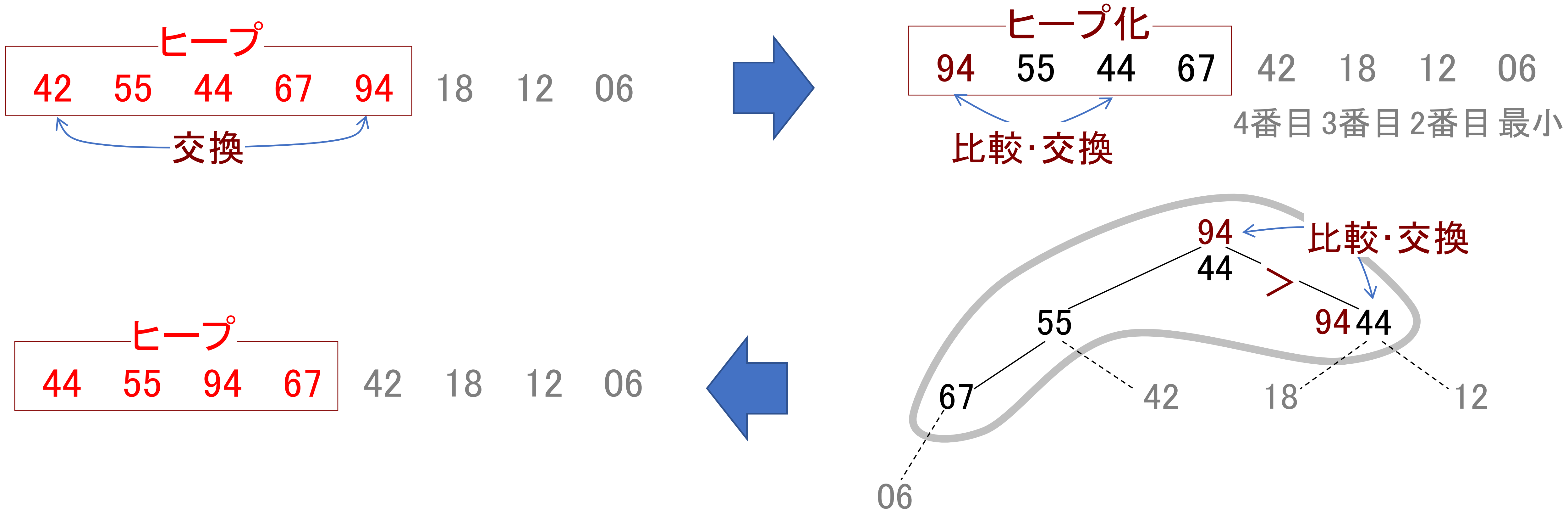
動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



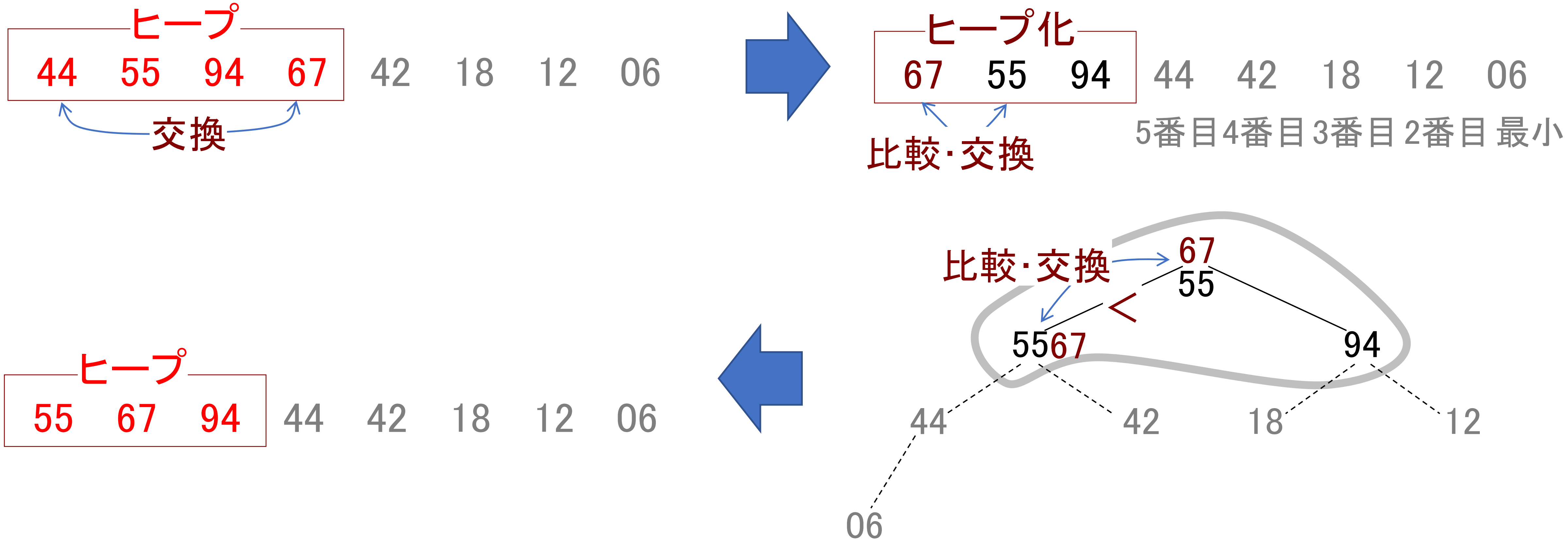
動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



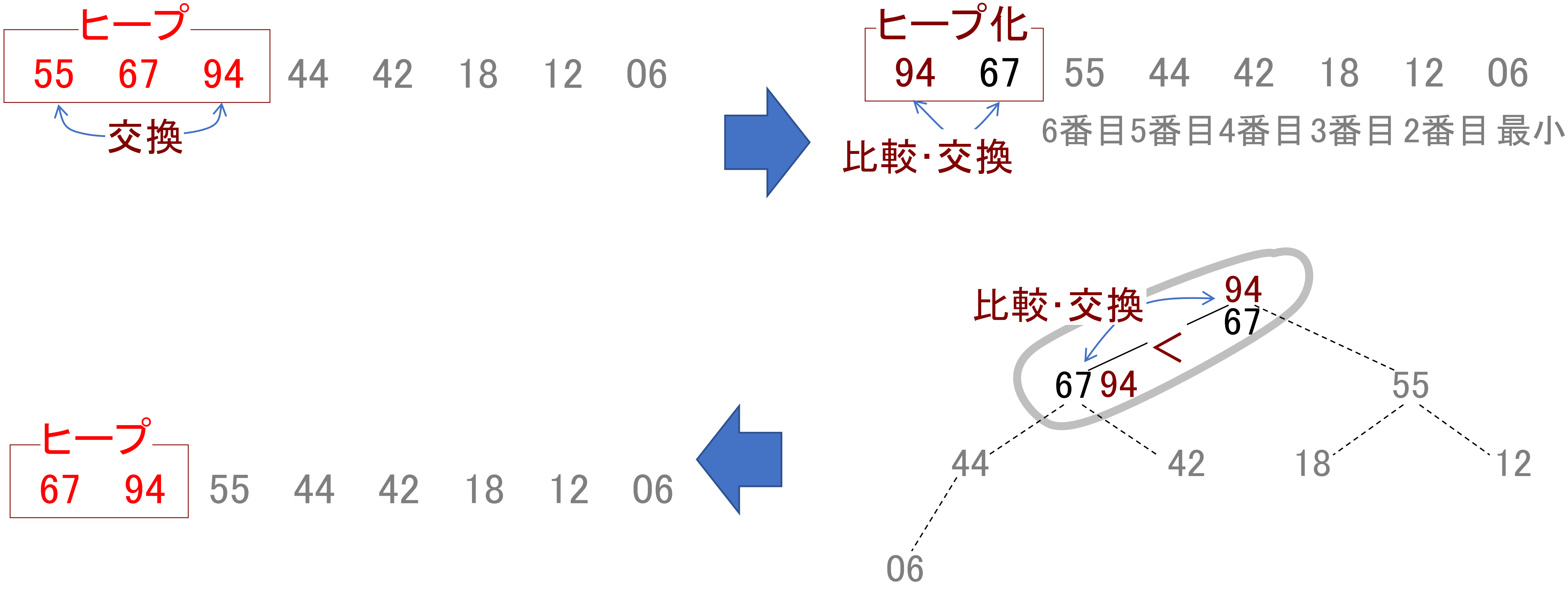
動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成



動作例：44, 55, 12, 42, 94, 18, 06, 67
ソート列を構成

ヒープ

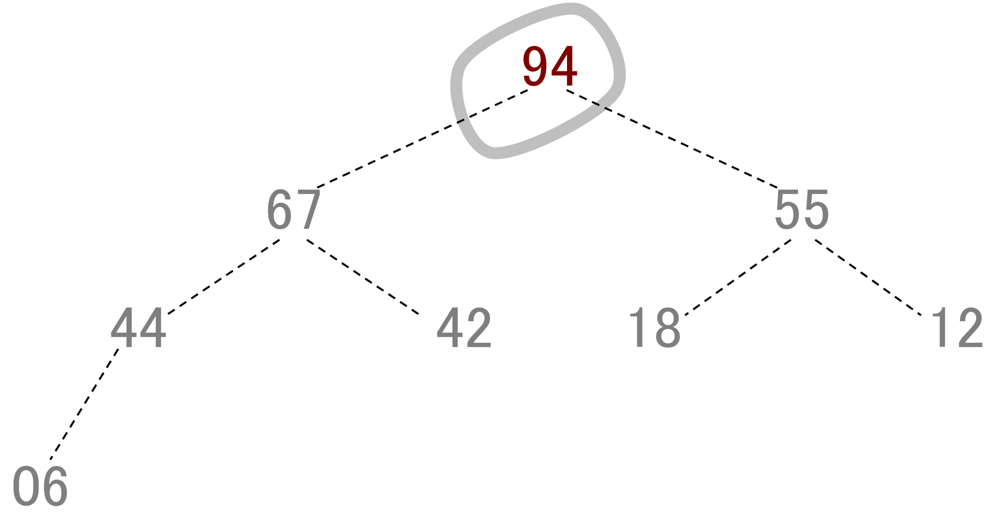
67　94　55　44　42　18　12　06

交換

ヒープ化

94　67　55　44　42　18　12　06

7番目6番目5番目4番目3番目2番目最小

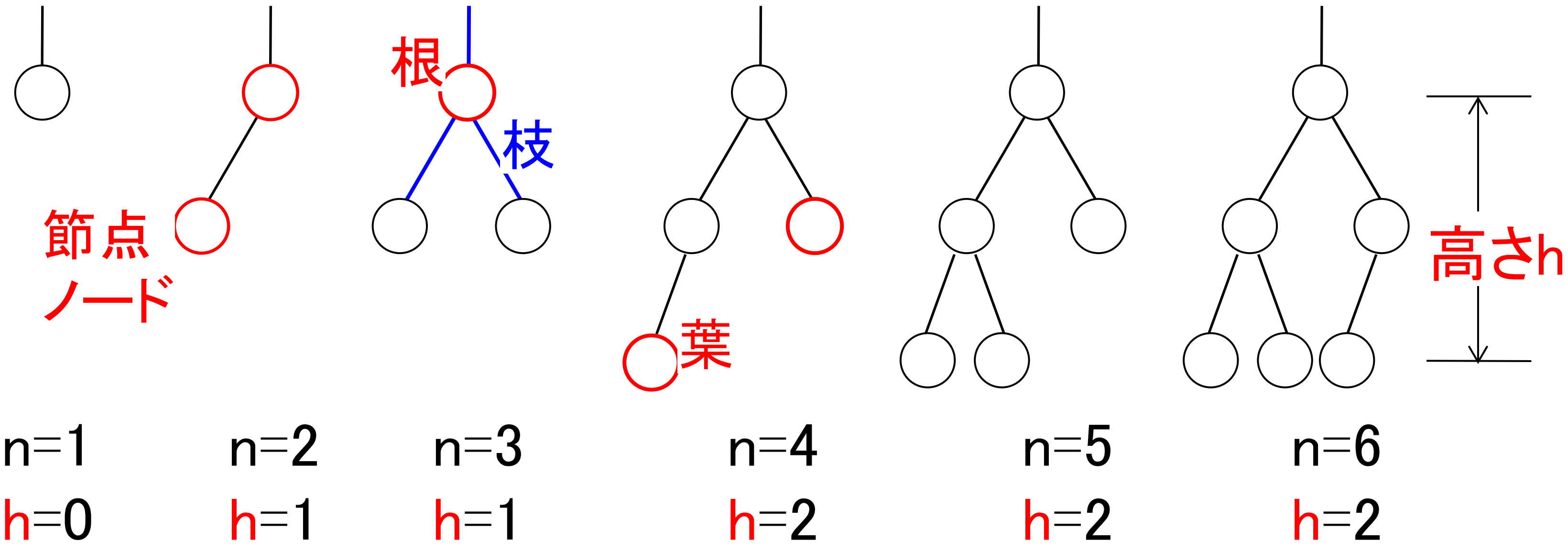


94　67　55　44　42　18　12　06

ソーティング完了:大きい順

2分木の節点数 n と 木の高さ h の関係

- 木の高さ h とは値から各葉に到る経路中の節点数のうちの「最大数-1」をいう
- 節点を左側から増やしていく2分木を左詰め2分木という



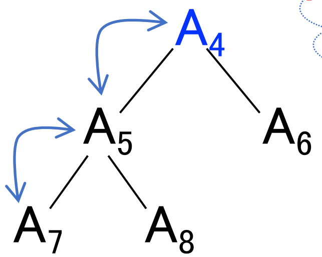
木の高さ $h=\log_2 n$ （nは総節点数）

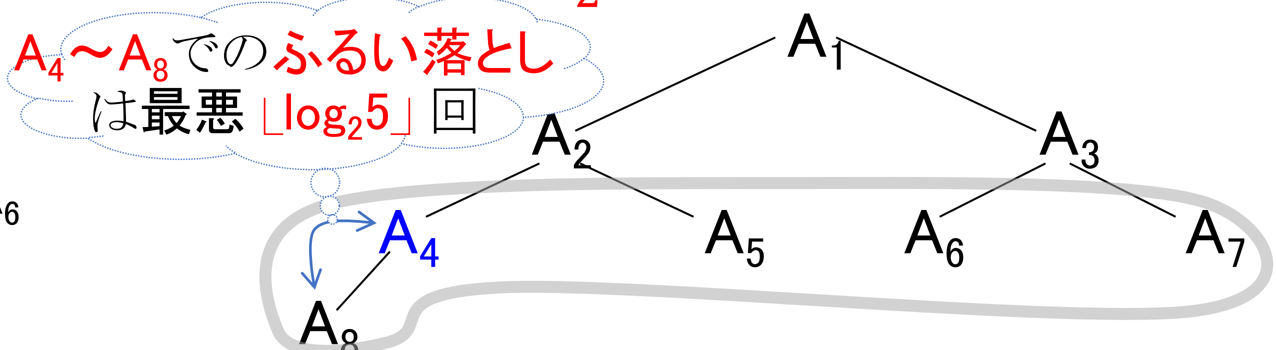
ヒープソートの時間計算量

- ヒープ構成の時間計算量
  - 最悪で $O(n\log_2 n)$ かかる
- ソート列構成の時間計算量
  - 最悪で $O(n\log_2 n)$ かかる
- どちらも同じことから，ヒープソートの時間計算量は
  - 最悪で $O(n\log_2 n)$ かかる

ヒープ構成の時間計算量

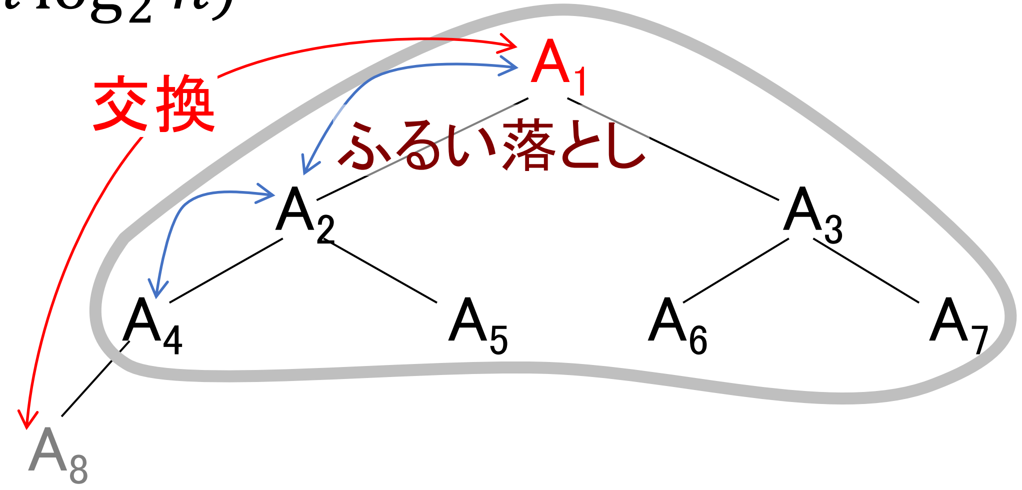
- 最初，$A_{\frac{n}{2}}$のふるい落とし
  - $A_{\frac{n}{2}}$からA_n までの$n-\frac{n}{2}+1=\frac{n}{2}+1$個の節点で行われる
  - $\frac{n}{2}+1$個からなる左詰め2分木の高さが$\log_2(\frac{n}{2}+1)$なので，最悪この高さ分に比例する移動が生じる
- 次に$A_{\frac{n}{2}-1}$のふるい落とし
  - 最悪で$\log_2(\frac{n}{2}+2)$に比例する移動が生じる
- 最後のA_1のふるい落とし
  - 最悪で$\log_2 n$に比例する移動が生じる
- $\log_2(\frac{n}{2}+1)+\log_2(\frac{n}{2}+2)+\cdots+\log_2 n \le \frac{n}{2}\log_2 n = O(n\log_2 n)$





ソート列構成の時間計算量

- ヒープ $A_1, A_2, \cdots, A_n$ で，A_1 と A_n とを交換した後，再び $A_1, A_2, \cdots, A_{n-1}$ をヒープするために，A_1 のふるい落としを行う．
  - これはn-1この節点の左詰2分木の根から行うので，最悪 $\log_2(n-1)$ 回移動する
- 以降，n-2個，n-3個，…，2個の節点の左詰2分木での根のふるい落としが行われるので，最悪で
- $\log_2(n-1) + \log_2(n-2) + \cdots + \log_2 2 \leq (n-2)\log_2(n-1)$
- $\leq n \log_2 n = O(n \log_2 n)$



ヒープソートの領域計算量

- 使用する領域
  - 配列といくつかの変数

  ↓
- ヒープソートの領域計算量
  - O(n)

まとめ

- ソーティングとは
- ソーティングアルゴリズム
  - 3つの単純法
  - ヒープソート

演習問題

- 問題1：
  - つぎの整数列を<u>**3つの単純法**</u>を用いてソートせよ。配列の図を描いて、各操作を明示すること。

6　2　7　1　3　5　4

- 問題2：
  - つぎの整数列は、10要素の配列の第１要素から第10要素とする。この配列の部分配列で<u>**ヒープ**</u>となる<u>**最大長のもの**</u>を指摘せよ。（ヒント：ヒープになり得る部分配列をすべて列挙する）

50　65　10　17　90　44　12　25　20　33

- 問題3：
  - つぎの整数列を<u>**ヒープソート**</u>せよ。ただし、図を用いて、要素の移動・交換を明示すること。

4　5　1　2　6　3　7

提出方法

- コンピュータのドローイングソフトなどを利用してもかまいませんが，手書きで結構です.
- 手書きで書いたレポートは，写真に撮って提出してください
  - 単純挿入ソートは，番兵ありとなしのどちらかがあればよいですが，両方あるとなおよいです
  - クイックソートは，第１版と第２版のどちらかがあればよいですが，両方あるとなおよいです

- 提出方法：LETUS
- 締め切り：2023/5/22 10:30まで