情報構造 第六回

ソーティングアルゴリズム（つづき）

今日の予定

- 内部ソートと外部ソート
- 内部ソート
  - 3つの単純法
  - ヒープソート
  - クイックソート
- 外部ソート
  - マージソート

ピタゴラスイッチ

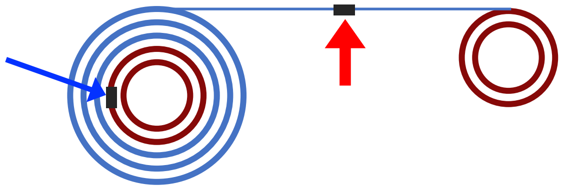
- じゃがいもソート
  - クイックソート
- しめじソート
  - マージソート

内部ソートと外部ソート

- 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

クイックソート

クイックソート

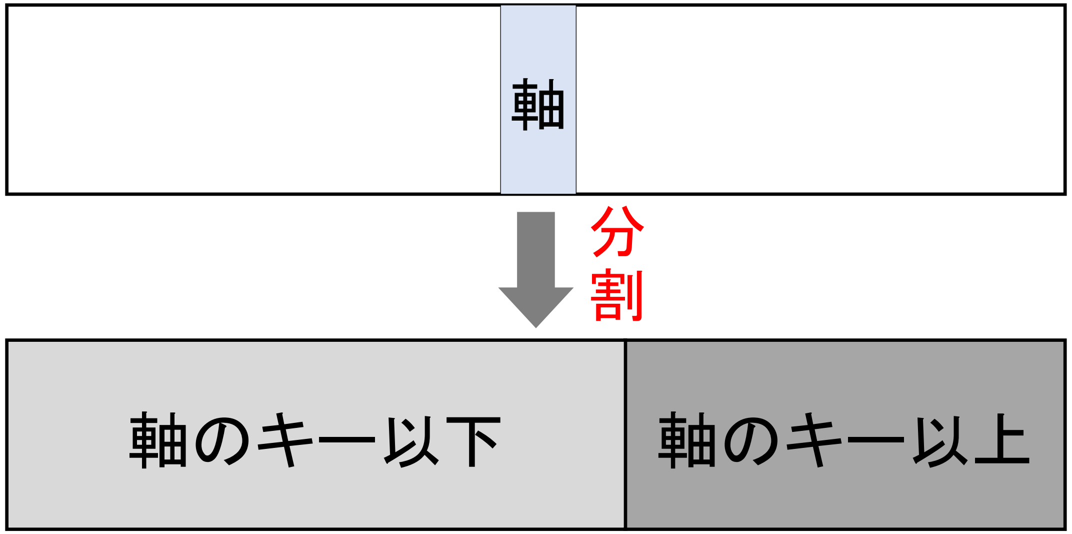
- アントニー・ホーアによって発明される
- 劇的な時間効率
- 配列の分割がポイントとなるため分割ソートとも呼ばれている

分割（partition）第１版

- 分割：配列のある要素を軸として，軸のキー以下の要素と，以上の要素に分ける

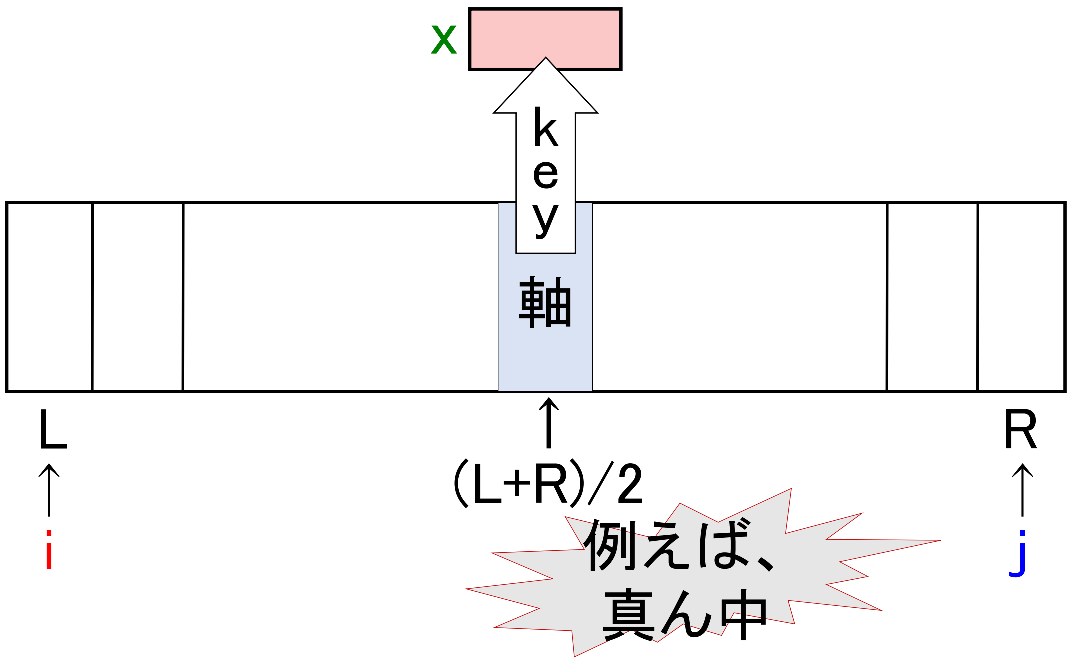
- 部分配列 A[L], A[L+1], …, A[R] の分割
  - 最初は L=0, R=n-1

- ステップ 1: 軸の選択
- ステップ 2: 走査
  終了判定　=>　終了
  　　　　　=>　ステップ 3
- ステップ 3: 交換
  終了判定　=>　終了
  　　　　　=>　ステップ 2



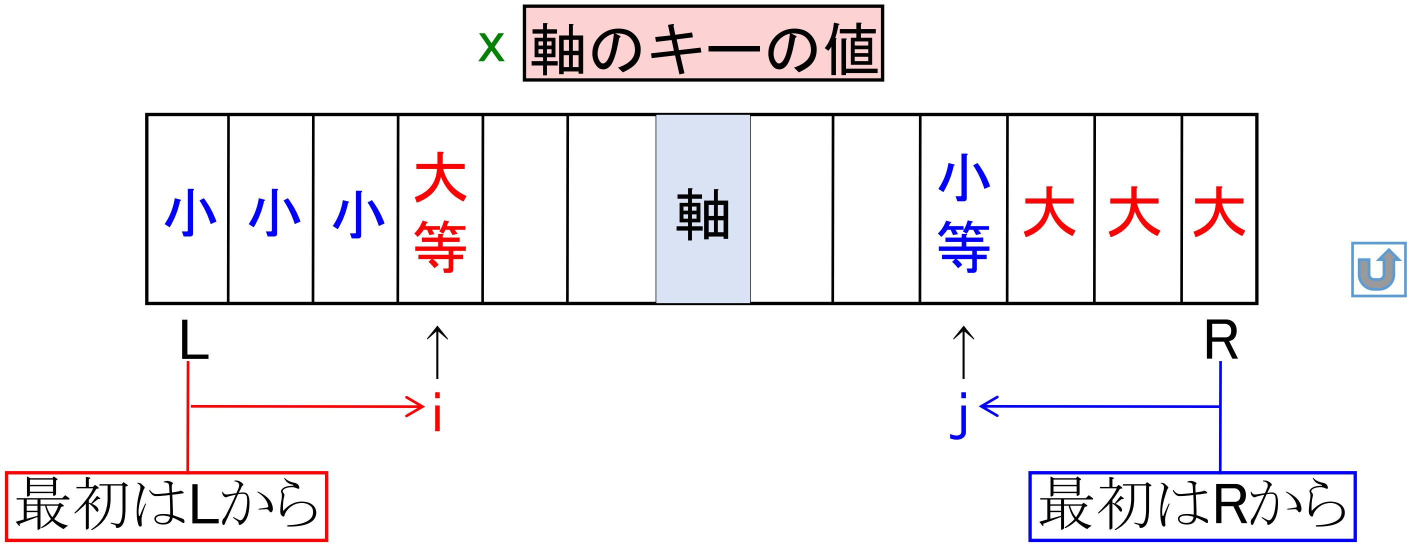
ステップ 1: 軸の選択

- 配列の一部からA[L], A[L+1], …, A[R] からランダムに任意の要素を選び軸とする．軸のkeyをxに設定
- 捜査のindexはi=L, j=Rとする



ステップ2: 走査

- 配列の左から右への走査
  - A[i].key≧x の要素がみつかるまで i をインクリメント
- 配列の右から左への走査
  - A[j].key≦x の要素がみつかるまで j をデクリメント

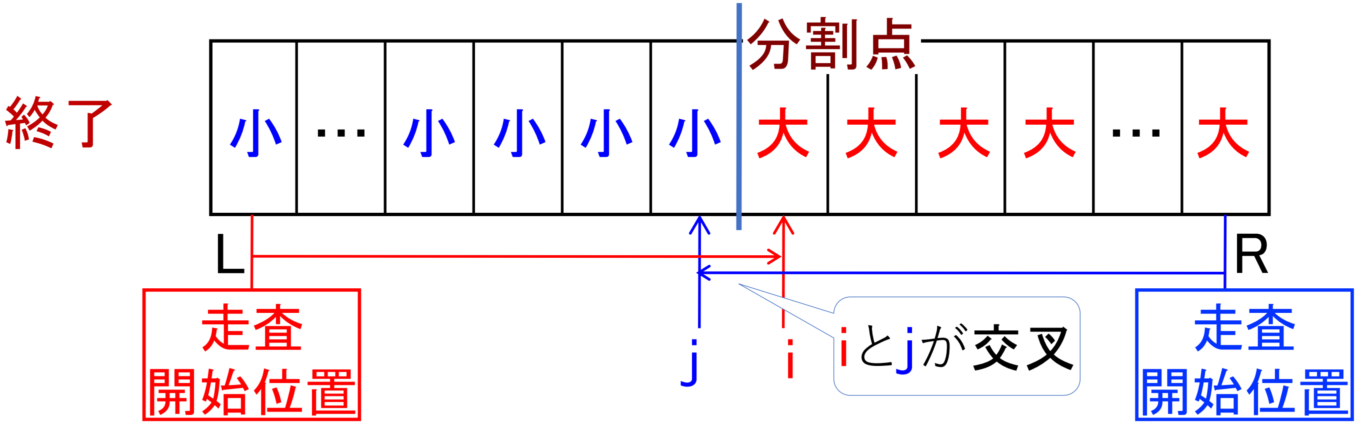


★走査中、iとR、jとLとの比較は行わない！

iとjとの比較は、**交換の前後だけ**で行う

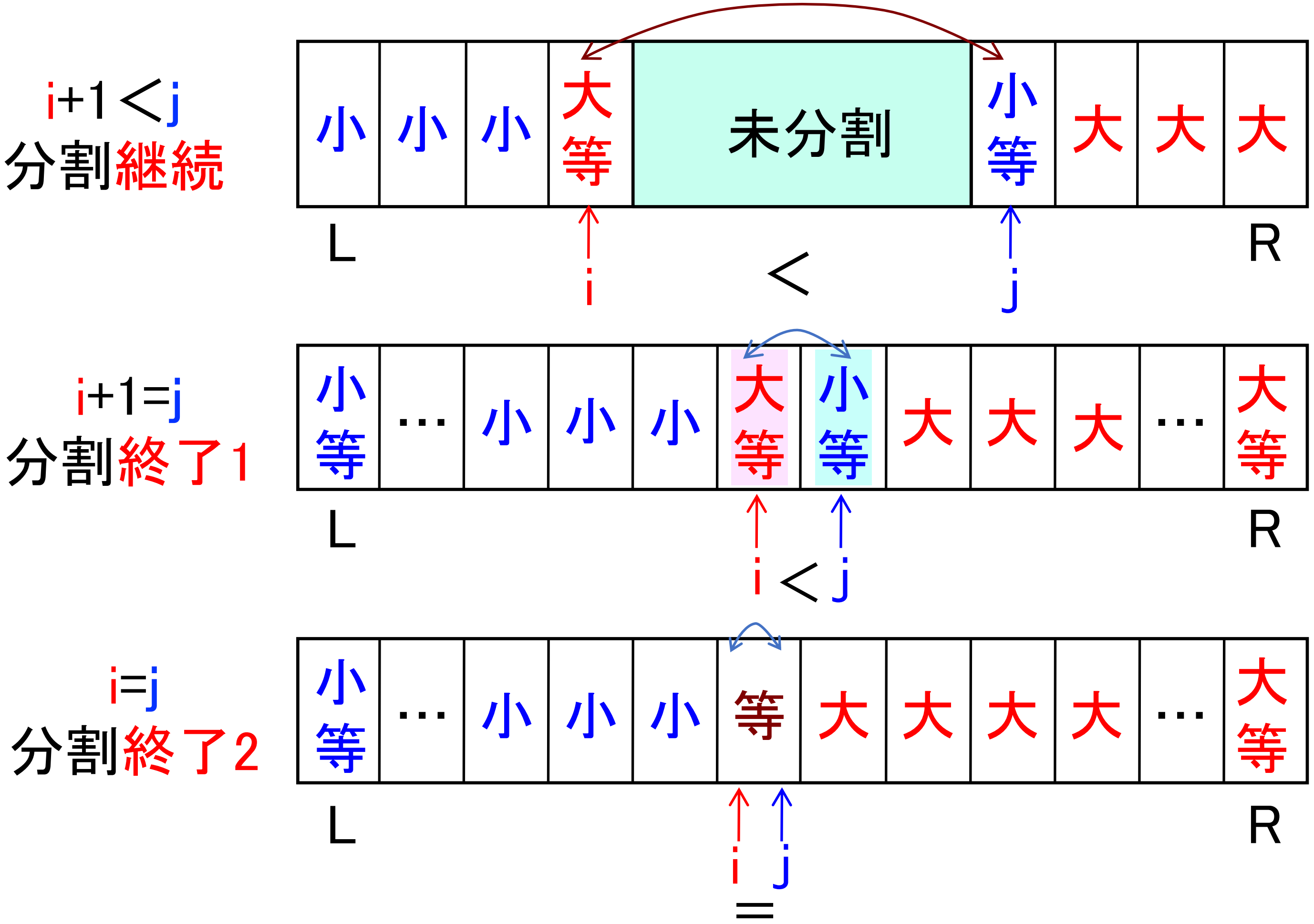
走査直後の終了判定

- 走査直後の終了判定
  - i>j ならば，分割終了
  - i≦j ならば，ステップ3の交換へ（A[i]とA[j]を交換）



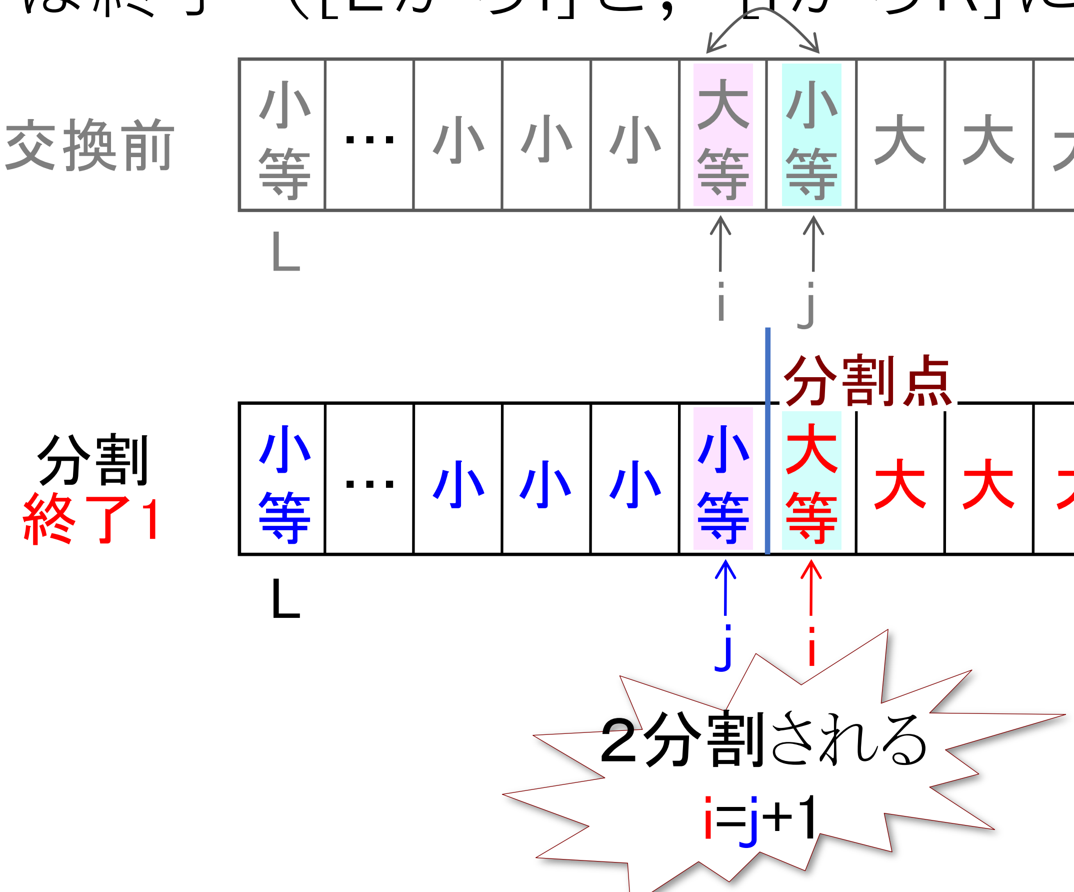
ステップ3: 交換

- i≦j ならば，A[i]とA[j]を交換し，i++; j--;



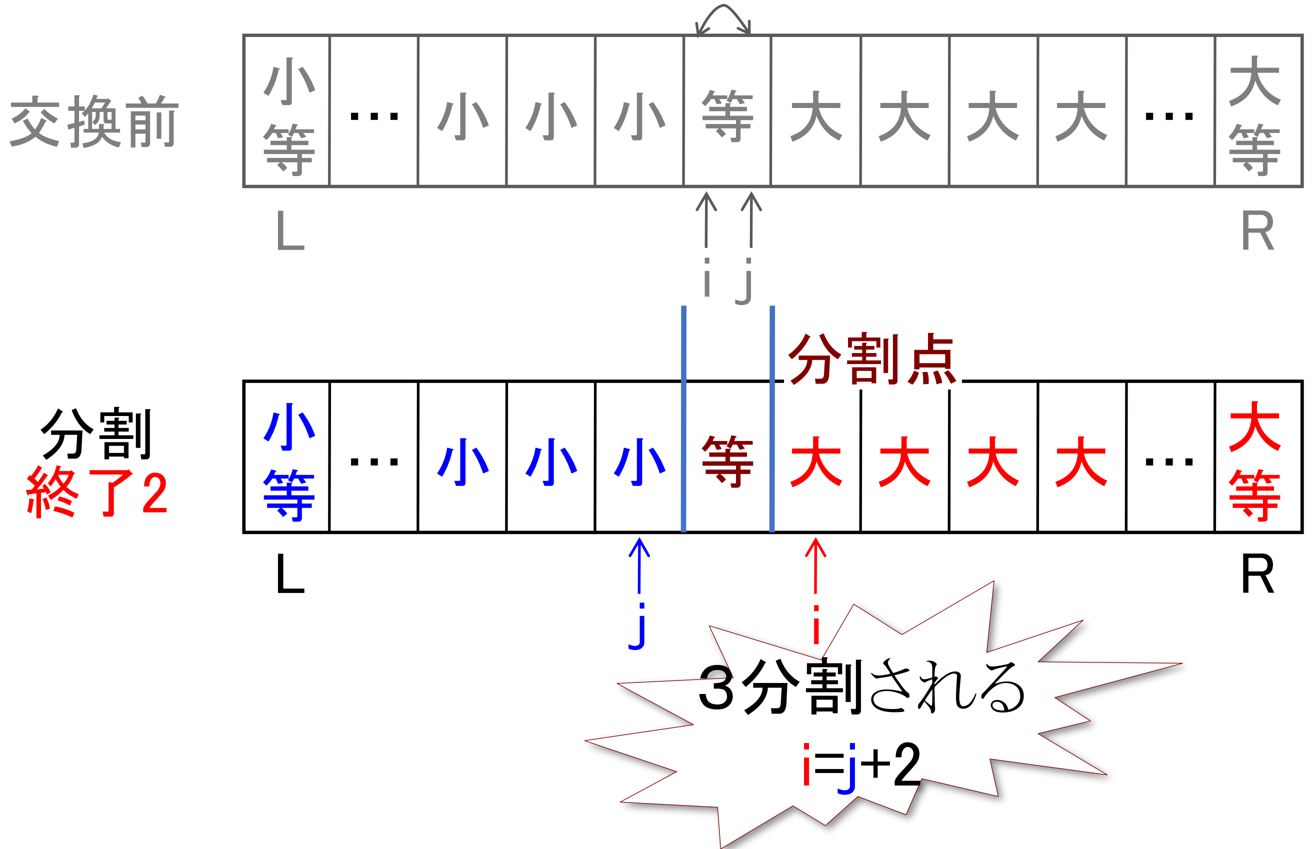
交換直後終了判定：分割終了１

- i+1=j　のとき，　i_{new}=i+1 , j_{new}= j- 1=i（i++, j--）
- i>j ならば終了（[Lからi]と，[iからR]に分割）



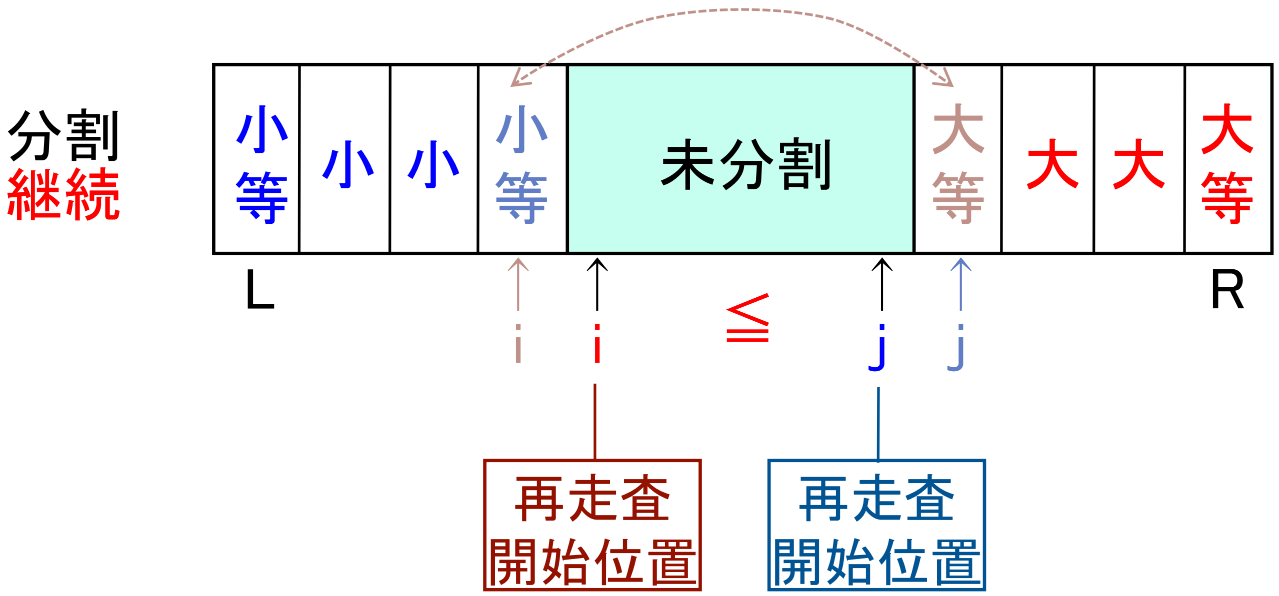
交換直後終了判定：分割終了２

- i=j のとき，i_{new}=i+1, j_{new}=j-1=i-1（i++, j--）
- i>jならば終了（[Lからi-1], [i], [i+1からR]に分割）



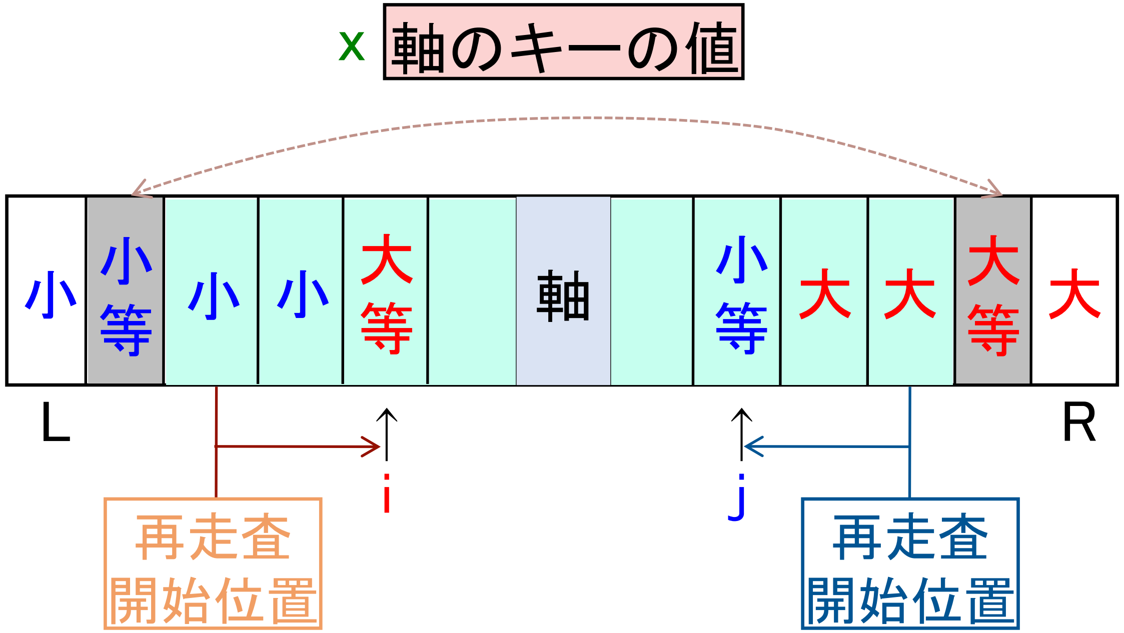
交換直後終了判定：分割継続

- i≦jならば，ステップ2に戻り分割を継続（再走査）

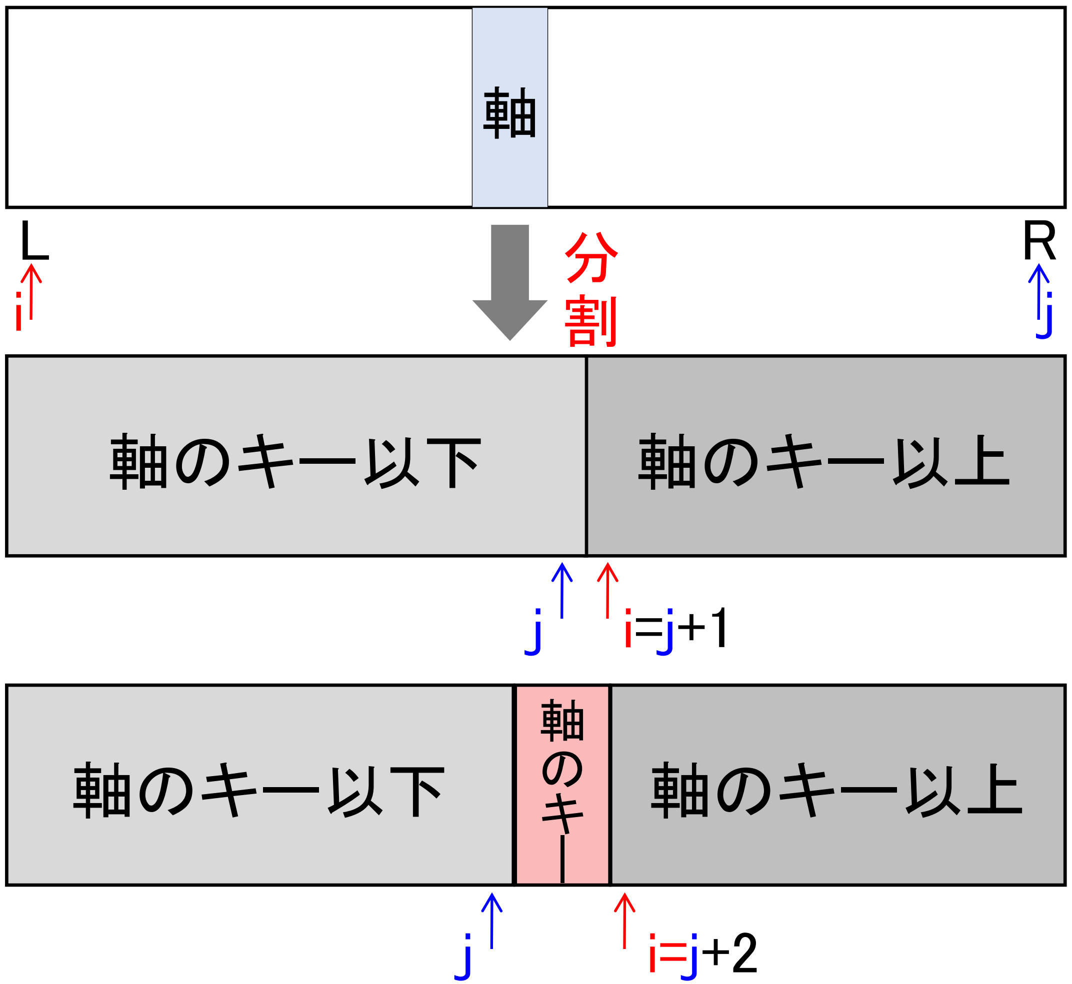


再走査のステップ２

- 交換直後に進めたiとjから再走査がはじまり，
- 走査直後の終了判定へ続く



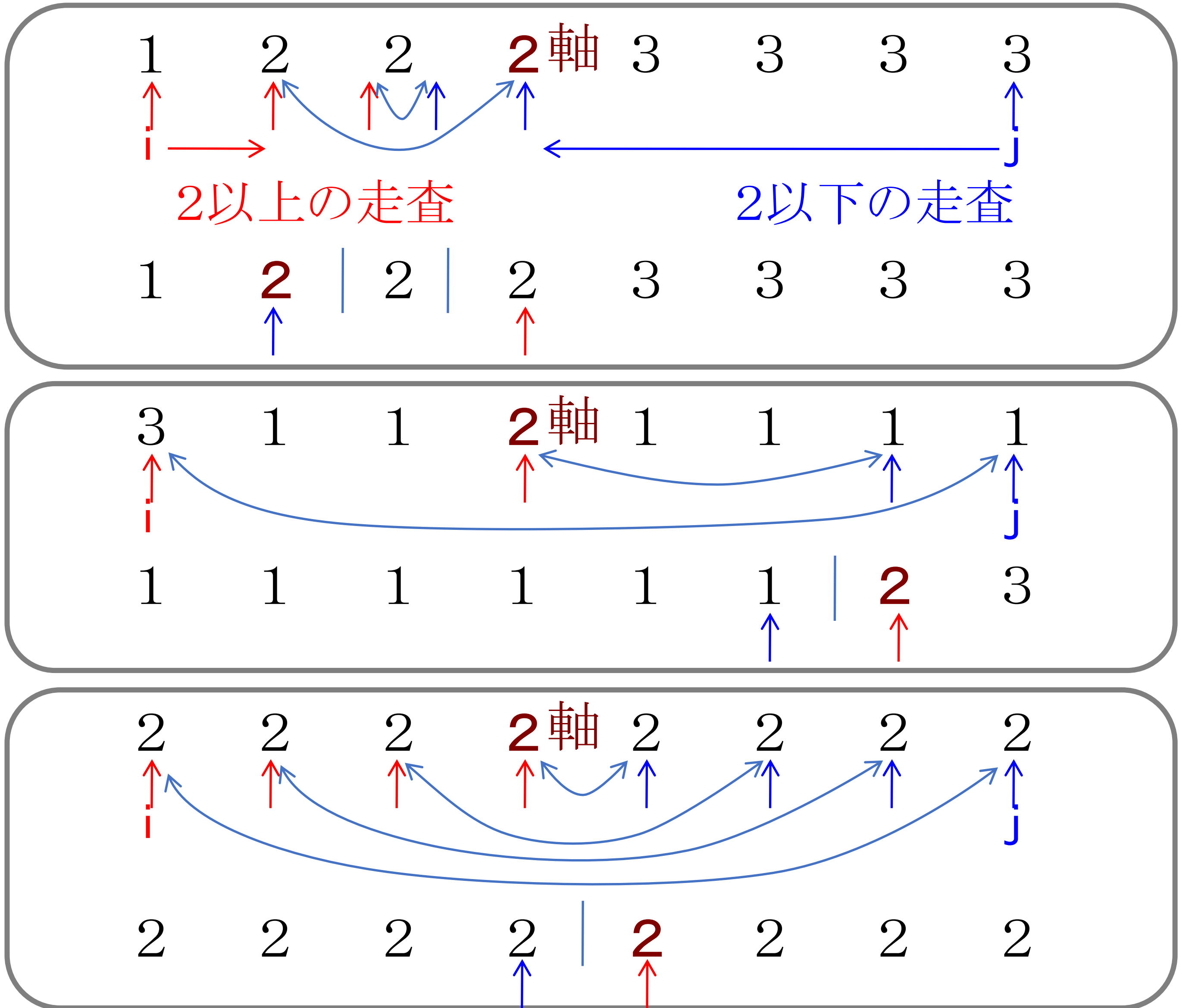
分割第１版の結果



分割 第１版 例：44, 55, 12, 42, 94, 18, 06, 67

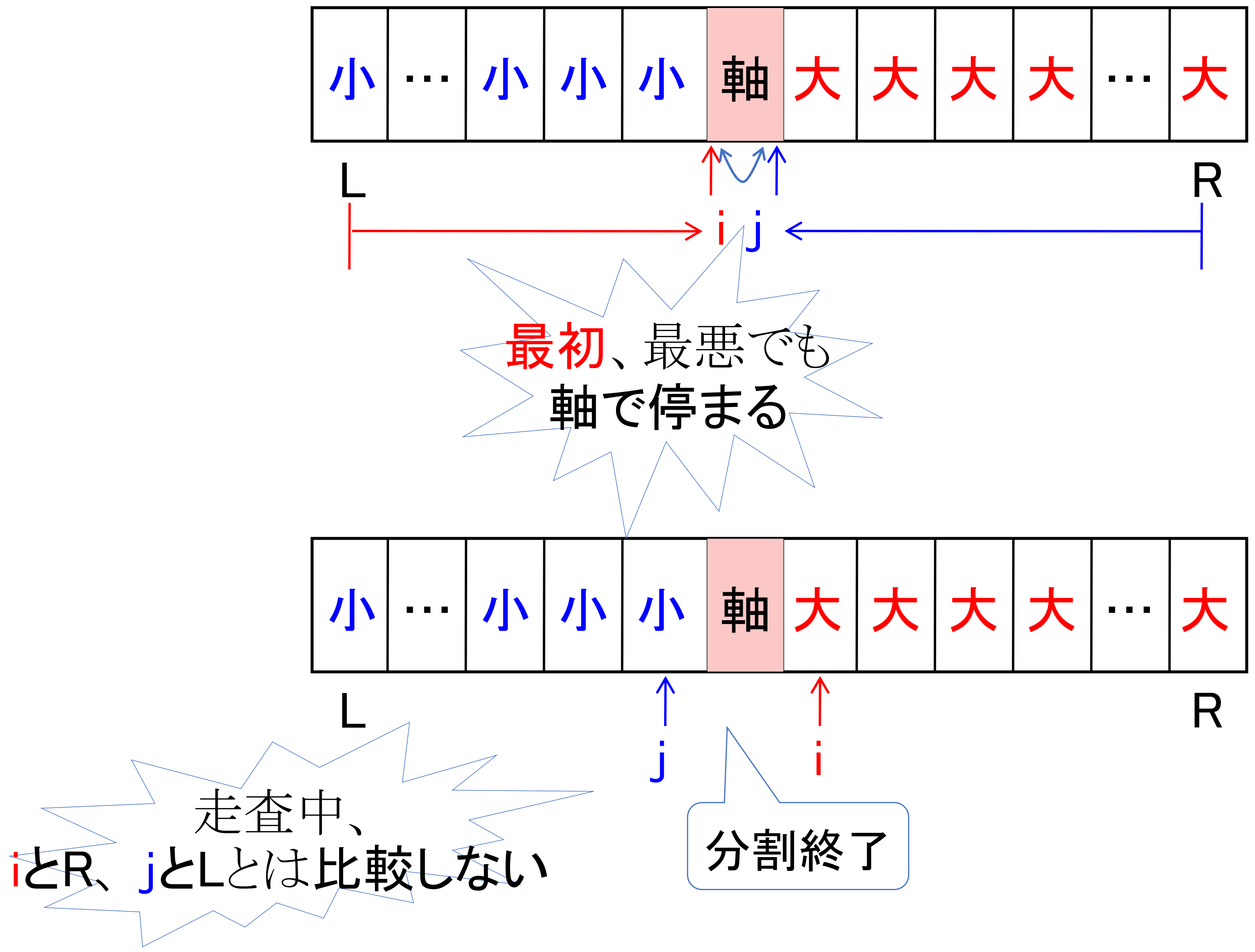
軸選択	44	55	12	42	94	18	06	67
	L=0			軸=(L+R)/2=3				R=7
走査	44	55	12	42	94	18	06	67
	i→軸のキー以上走査				軸のキー以下走査←j			
交換	44	55	12	42	94	18	06	67
	i			i≦j			j	
走査	06	55	12	42	94	18	44	67
		i→		i≦j より、		←j		
				ここから再走査				
交換	06	55	12	42	94	18	44	67
		i		i≦j		j		
走査	06	18	12	42	94	55	44	67
			i→	i≦j	←j			
				ここから再走査				
交換	06	18	12	42	94	55	44	67
				i j				
終了	06	18	12	42	94	55	44	67
	L		j		i			R
	≦42			＝交換後 i>j	≧42			

注意：軸も移動する

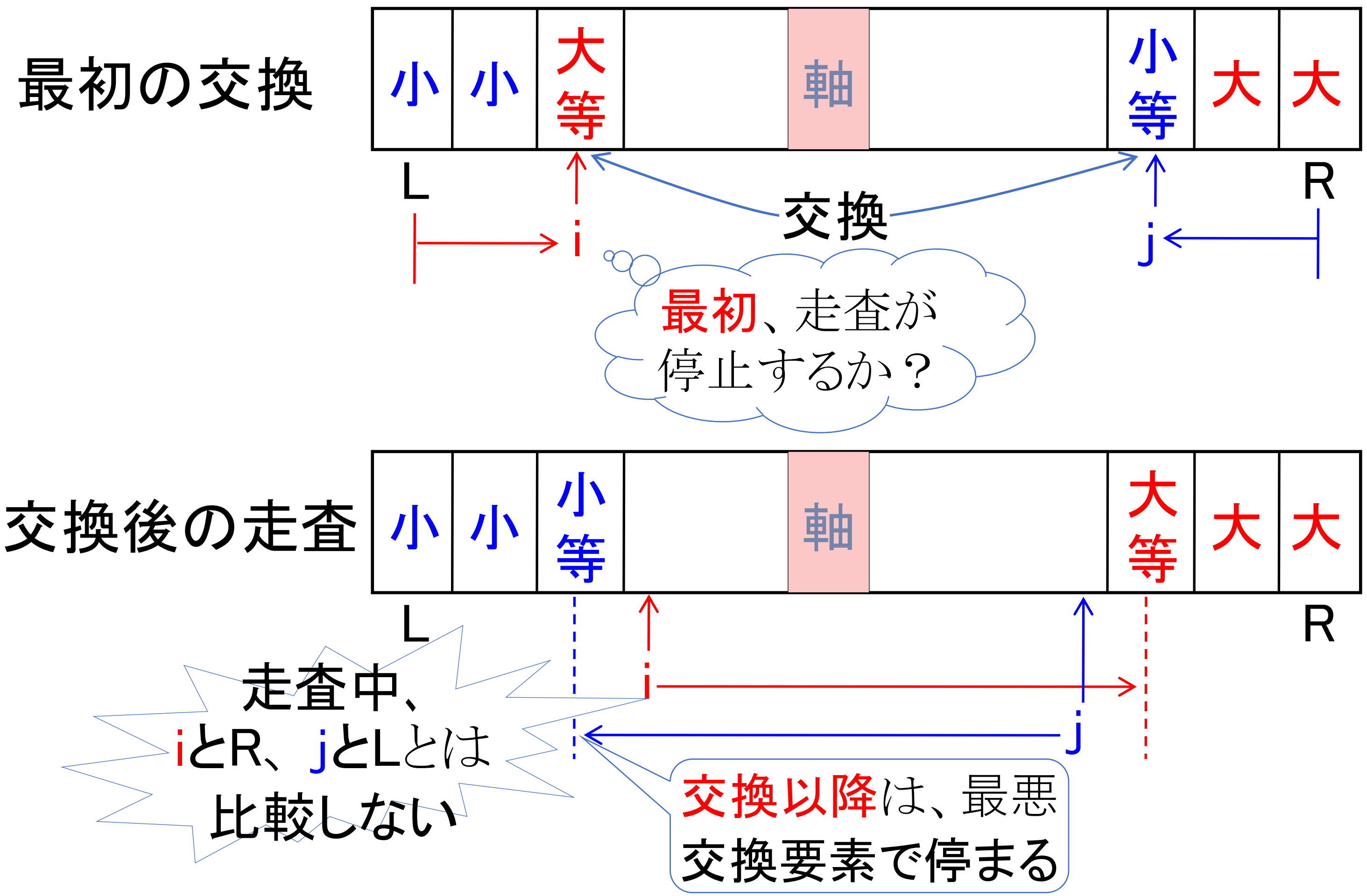


1 2 2 2軸 3 3 3 3
i
j
2以上の走査
2以下の走査
1 2 2 2 3 3 3 3
3 1 1 2軸 1 1 1 1
i
j
1 1 1 1 1 1 2 3
2 2 2 2軸 2 2 2 2
i
j
2 2 2 2 2 2 2 2

分割第１版：最初の走査の停止



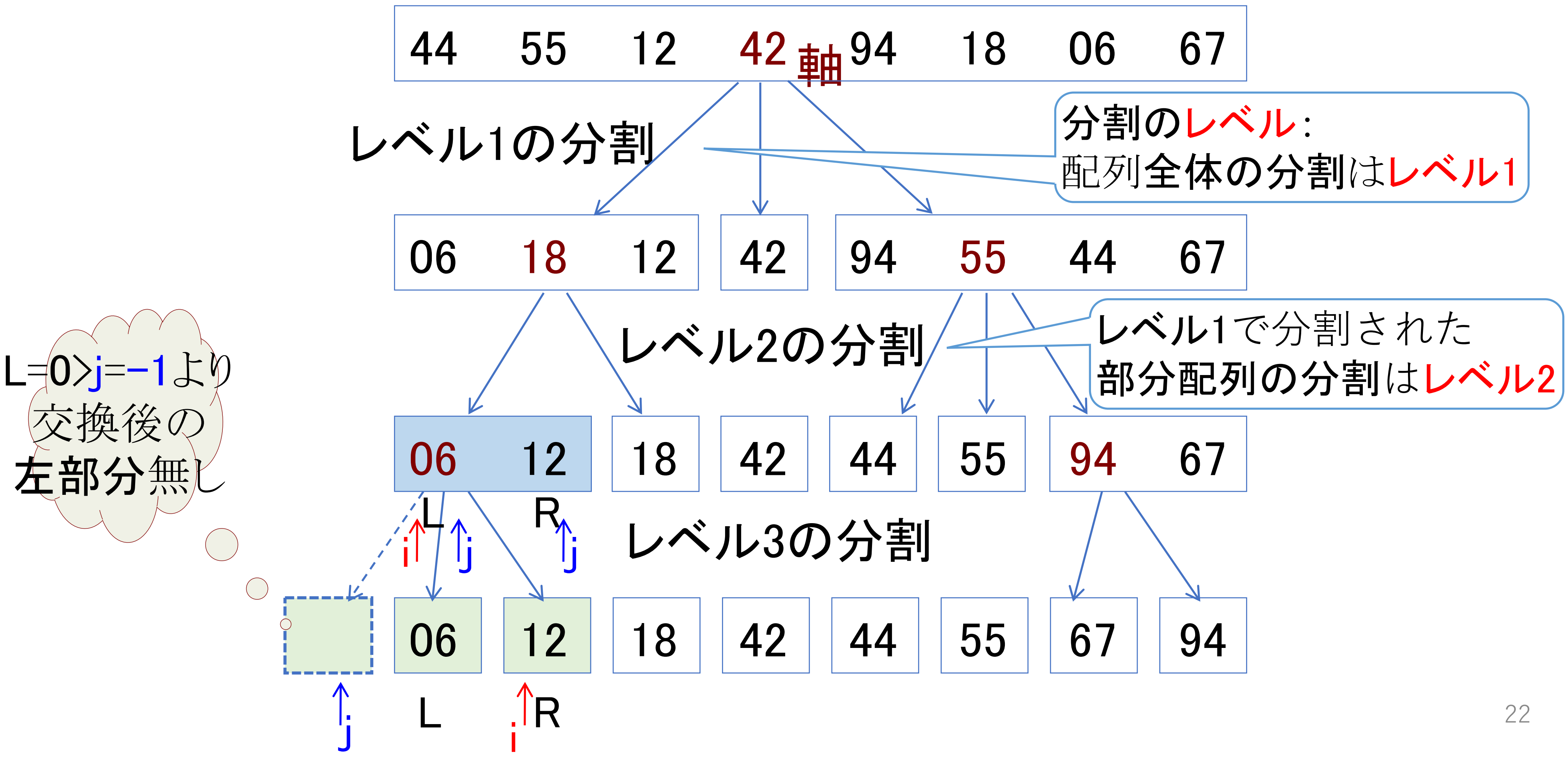
分割 第１版：交換後の走査の停止



クイックソートの原理

- 初期列としてA[0],…, A[n-1] を与える
  - L=0: R=n-1;
- 部分配列A[L],…, A[R]を，分割（第１版）を使って分割
  - A[L]…A[j] と A[i]…A[R]
  - 各配列の要素数が
    - 2個以上なら分割を繰り返す（再帰的分割）
    - 2個未満の列は分割終了

クイックソート（第１版）の例



クイックソート（第１版）C言語

`void qsort(int L, int R){`	A[L], …A[R]の再帰的分割手続き
`int i, j; item w;`	
`i=L; j=R;`	
`x=A[(L+R)/2].key;`	軸のキーを設定
`do{`	
`while(A[i].key<x) i++;`	左から走査
`while(x<A[j].key) j--;`	右から走査
if(~~i>j~~ i<=j){	走査直後の比較
`w=A[i]; A[i]=A[j]; A[j]=w;`	交換
`i++; j--; }}`	走査インデックスを進める
`while (i<=j);`	交換直後の比較
`if(L<i) qsort(L, j);`	各部分列を再帰的に再分割
`if(i<R) qsort(i, R); }`	（Lとj, Rとi の比較はここだけ）
`void quicksort(){`	メイン
`qsort(0, n-1); }`	配列全体を与える

まちがっていました
(5/22)

例：最悪の場合のクイックソート（第１版）

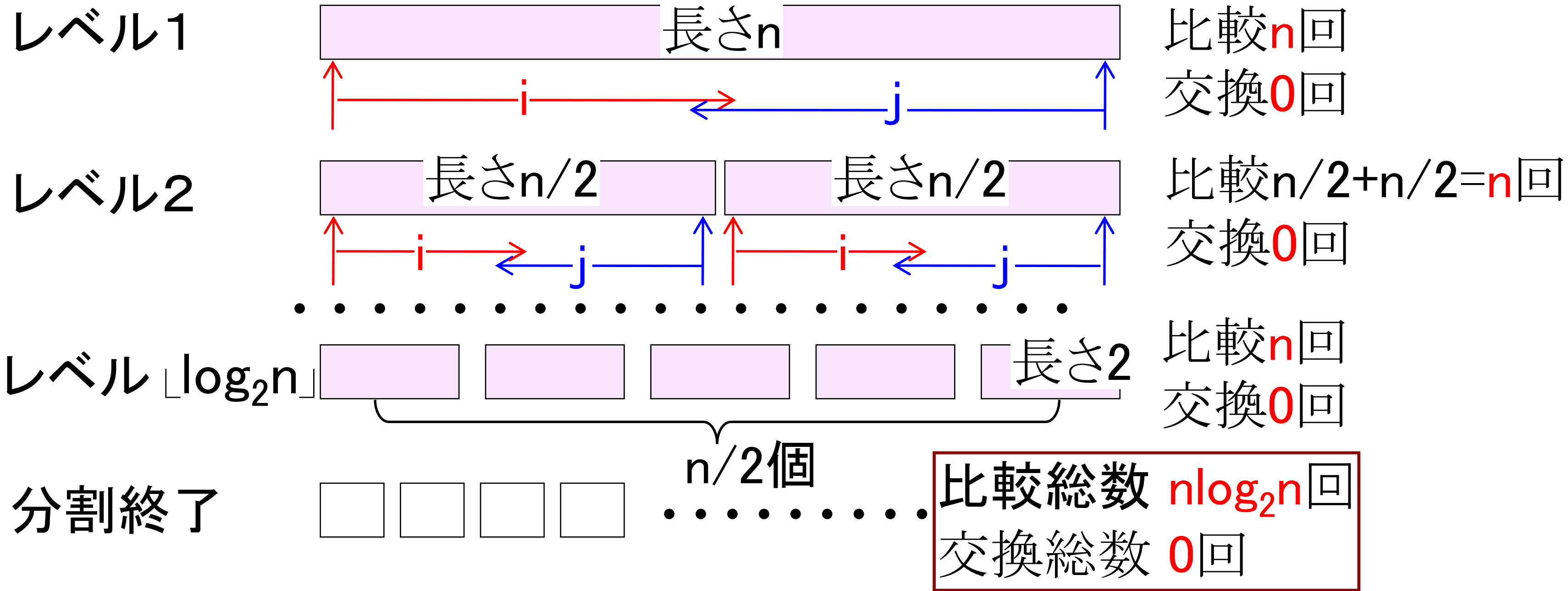
クイックソートの時間計算量

- 最良の場合
  - $O(n\log_2 n)$

- 最悪の場合
  - $O(n^2)$

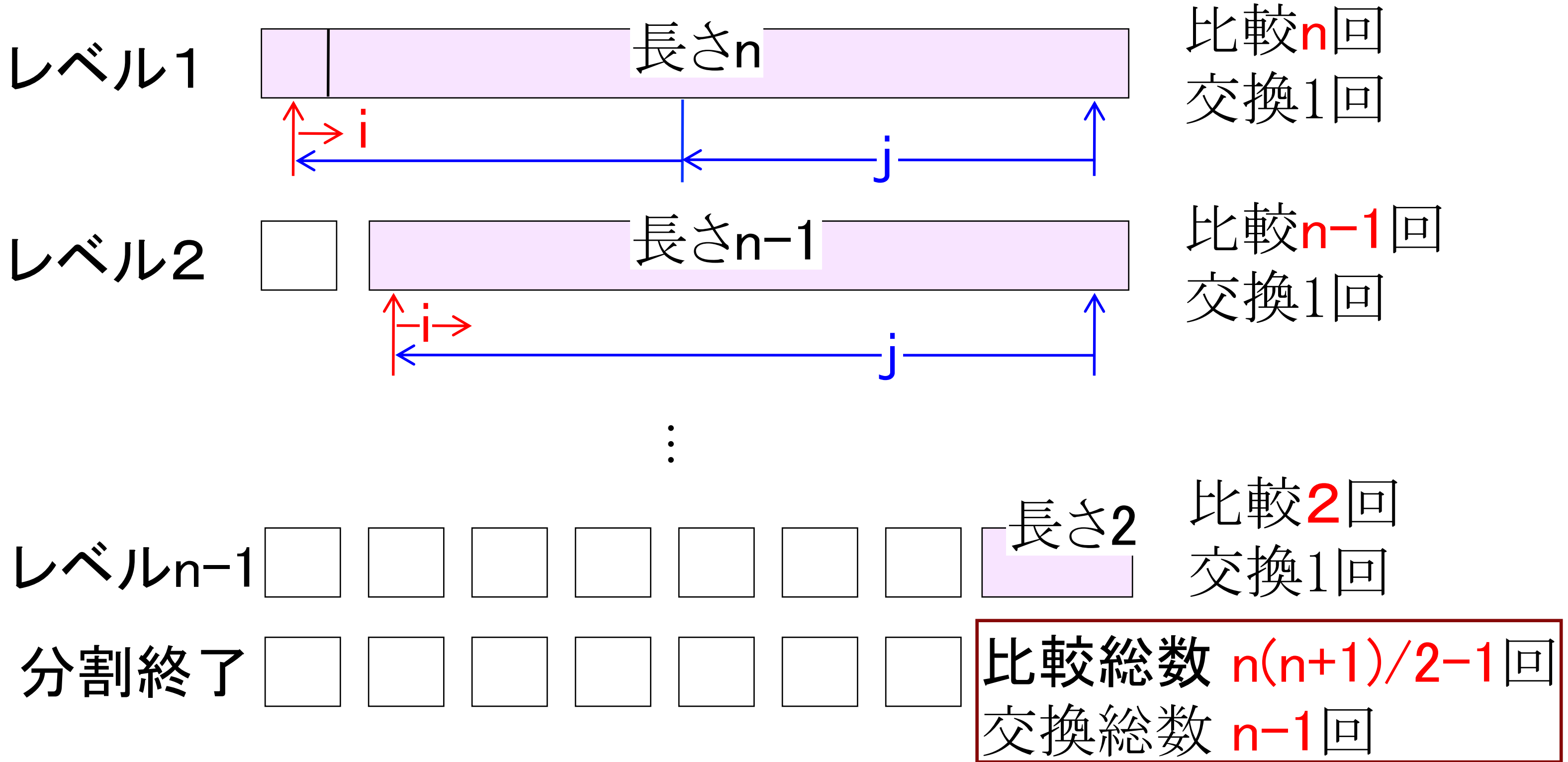
最良の場合の時間計算量

- 最良の場合（小さい順に並んでいるとき）
  - 分割が列を2等分する
- 分割のレベル数は $\log_2 n$
- 走査の過程で，すべての要素と軸を1回比較する
  - 要素と軸を比較は，どのレベルでも配列全体のn回
  - 交換は，各部分列で0回=> 時間計算量は$O(n\log_2 n)$



最悪の場合の時間計算量

- 最悪の場合（軸が最小か最大のとき）
  - 分割がいつも1個と残りになる
    - 分割のレベル数は n-1
  - 軸と比較回数は配列の長さ，交換は1回



=> 時間計算量は $O(n^2)$

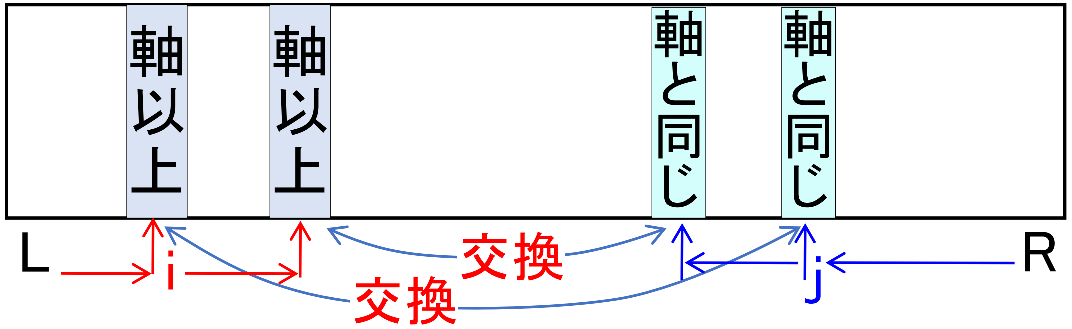
クイックソートの領域計算量

- 分割手続きqsort：局所変数 i, j, x, wやパラメータL, R
  - この領域数は定数c個
  - 再帰的に呼び出されるたびにこの領域数をつかう
- 最良の場合：
  - 等分に分割されていき，レベルは$\log_2 n$：$\log_2 n$回よびだし => $c\log_2 n$
  - 配列の領域を加えて：$n+c\log_2 n$
- 最悪の場合
  - 1つと残りに分割されていきレベルはn-1:n-1回のよびだし => c(n-1)
  - 配列の領域を加えて：n+c(n-1)
- 最良・最悪とも領域計算量はO(n)

分割（第１版）：走査条件の重なり

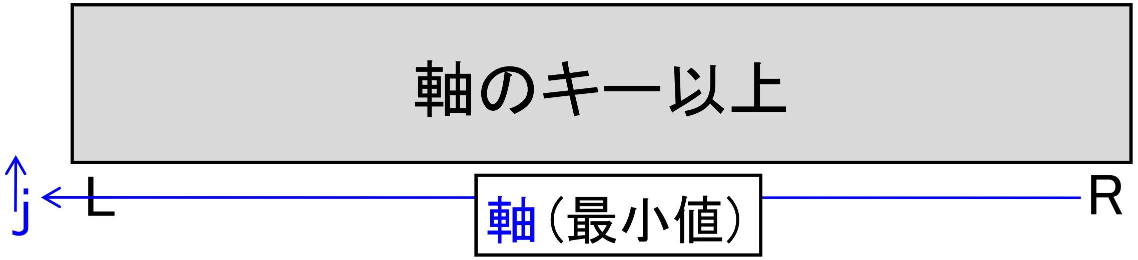
- 左から軸キー以上，右から軸キー以下
  - A[j].key=軸キー のとき，走査が停止し交換
- 左から軸キー以上，右から軸キー未満とする
  - A[j].key=軸キーのとき，走査が停止しない

  => 交換回数が減る！（同じ値の交換が起きない）



- 問題：軸が最小値のとき，右からの走査が配列の左端を突き抜けてしまう

=> 軸に最小値を選ばないようにする

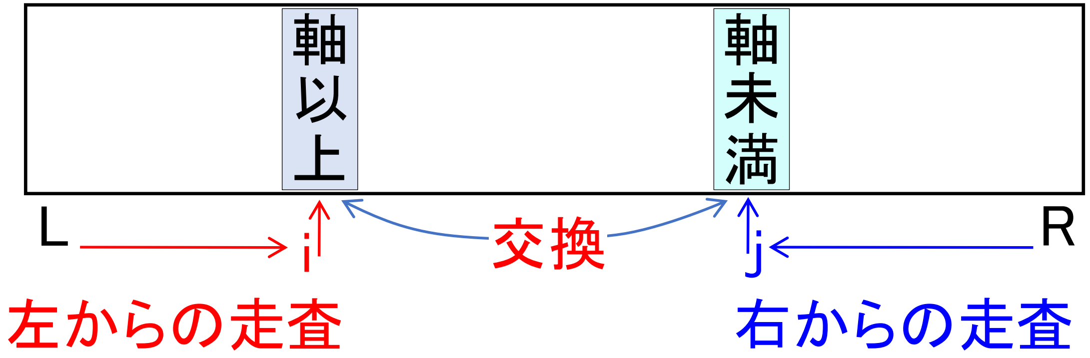


分割（第２版）

- 右からの走査：軸のキー以上を見つける
- 左からの走査：軸のキー未満をみつける
- 軸に最小キーを選ばない

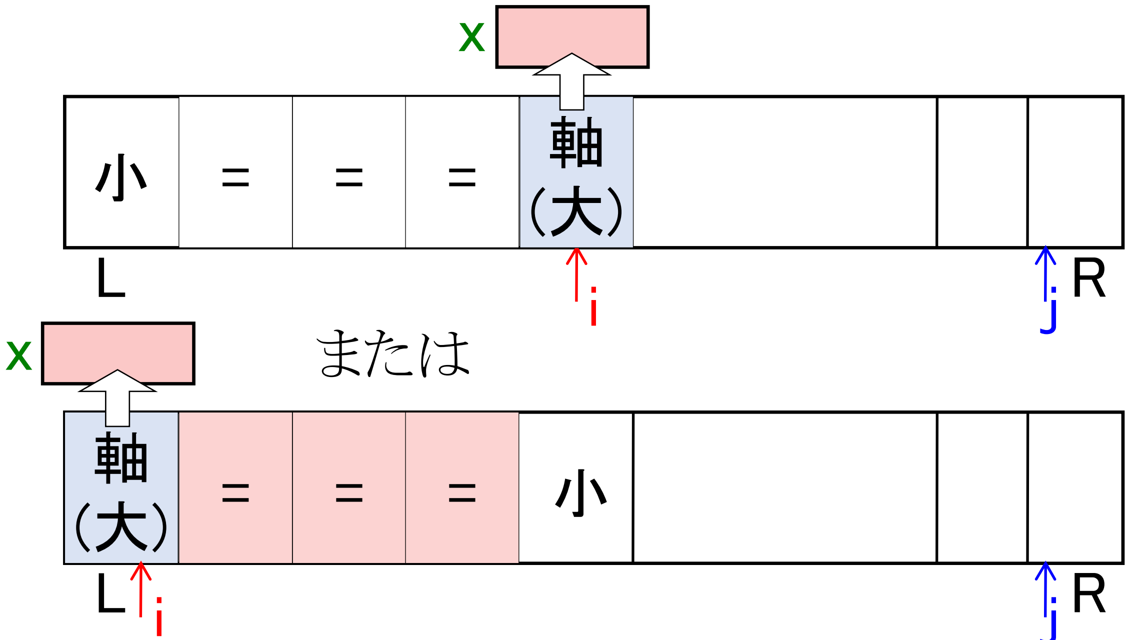
- 部分配列 A[L], A[L+1], …, A[R] の分割
  - 最初は L=0, R=n-1

  - ステップ1: 軸の選択
  - ステップ2: 走査
    終了判定 => 終了
    => ステップ3へ
  - ステップ3: 交換
    終了判定 => 終了
    => ステップ2へ



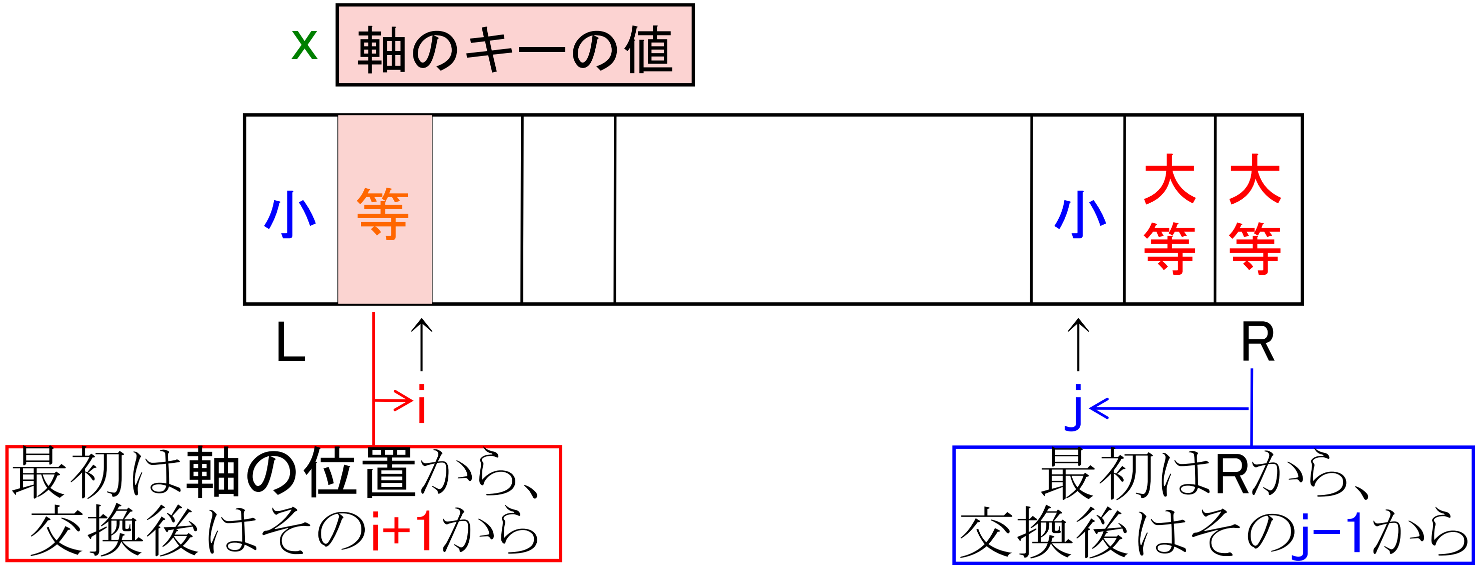
ステップ1: 軸の選択

- A[L], A[L+1],…, A[R]のすべての要素が同じキーのときは，分割終了し，i=R, j=Lとする
- そうでないとき，A[L], A[L+1],…, A[R]の左端から順に見ていき異なる２つのキーのうち大きい方の要素を軸とする．軸のキーをxとする，i=軸の位置，j=Rとする



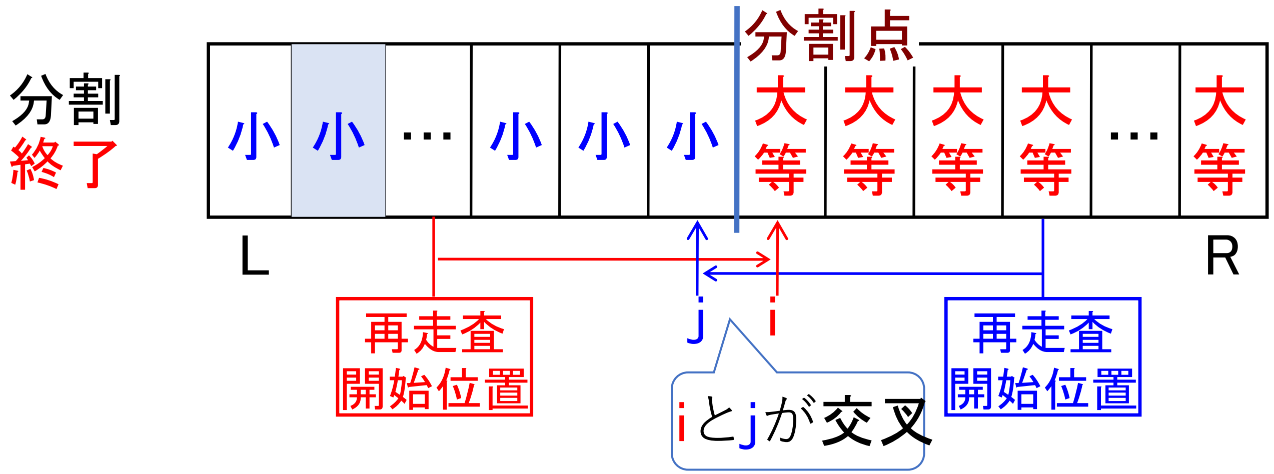
ステップ2: 走査

- 配列の左から右へ走査
  - A[i].key≧x の要素が見つかるまで i をインクリメント
- 配列の右から左へ走査
  - A[i].key<x の要素が見つかるまで j をデクリメント



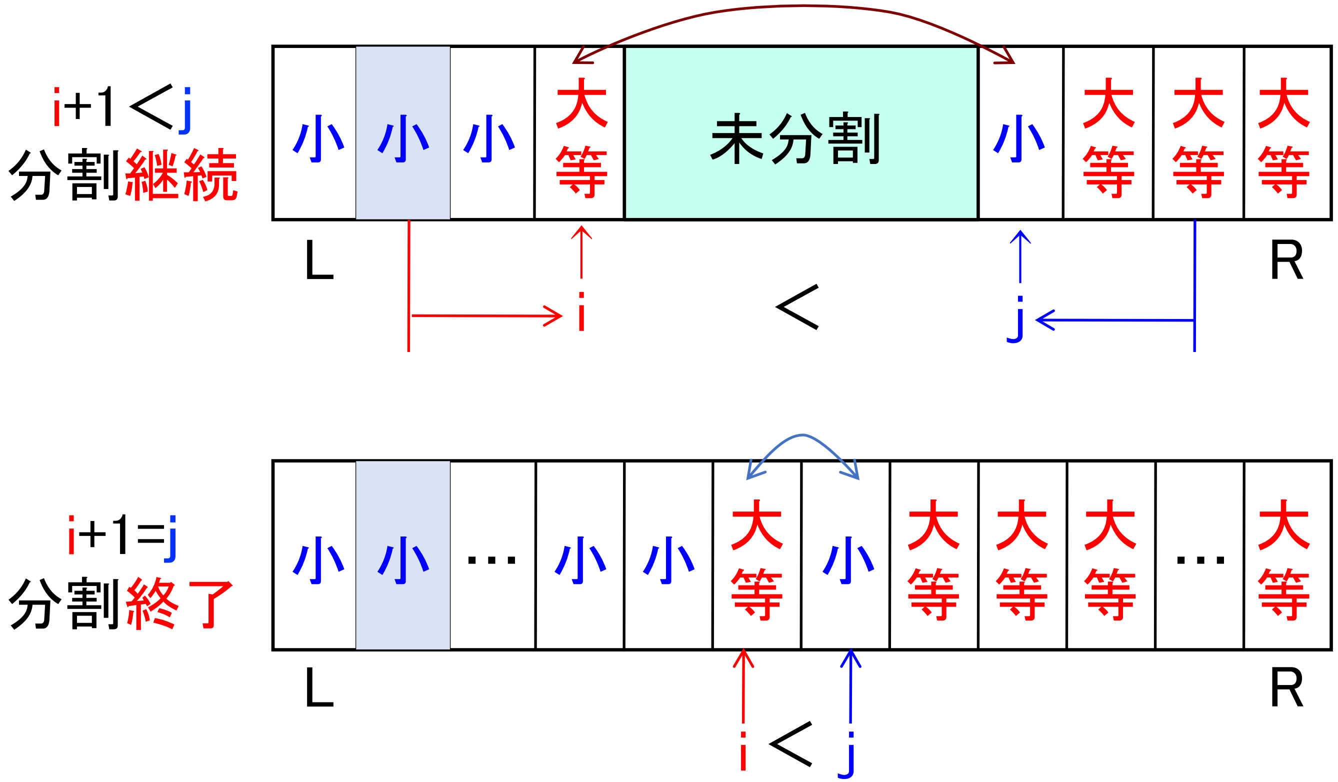
走査直後の終了判定

- 走査直後終了判定
  - i>j ならば，分割終了
  - i<j ならば，ステップ3の交換へ（A[i] と A[j]を交換）



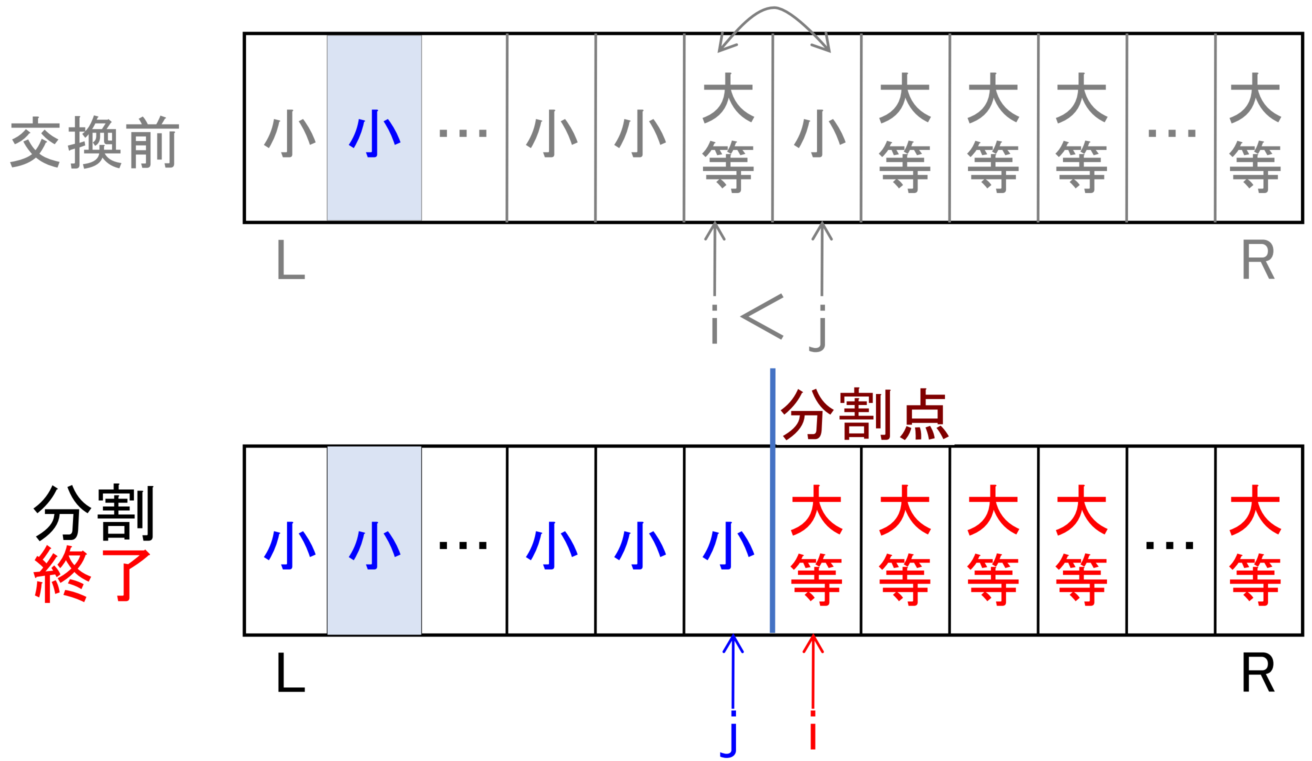
ステップ3: 交換

- i<jならば，A[i]とA[j]を交換し，i++; j--;



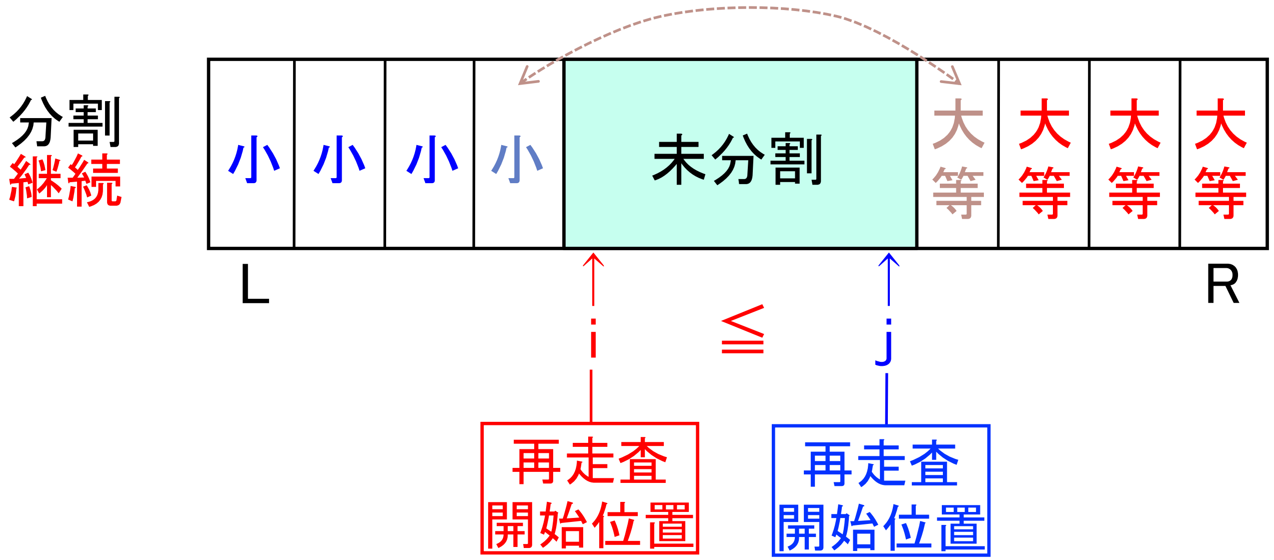
交換直後の終了判定：分割終了

- i>j ならば，終了



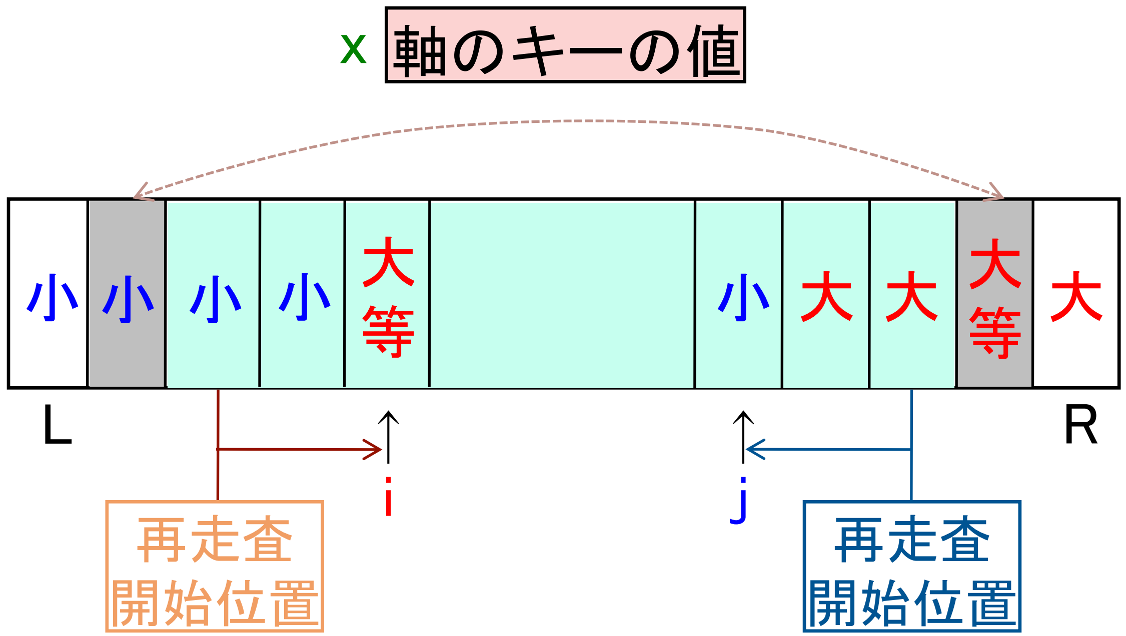
交換直後の終了判定：分割終了

- i<j ならば，ステップ2へ，分割を継続（再走査）

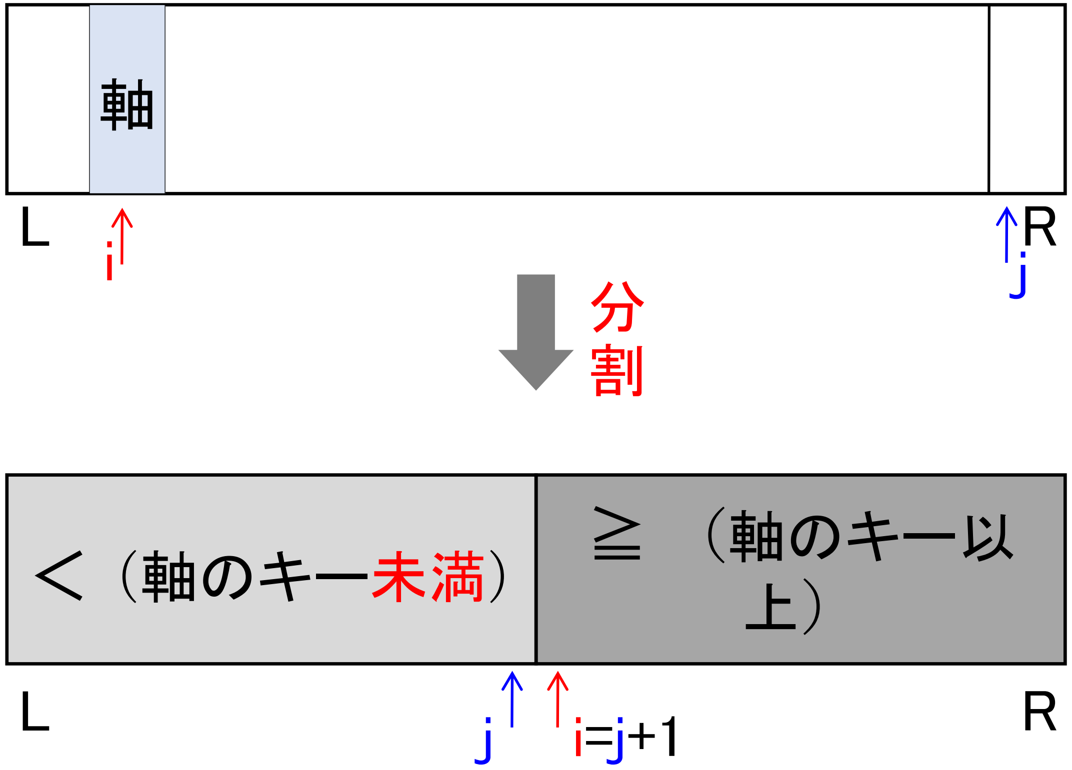


再走査のステップ2

- 交換直後に進めた i と j から再走査が始まり,
- 走査直後の終了判定へと続く



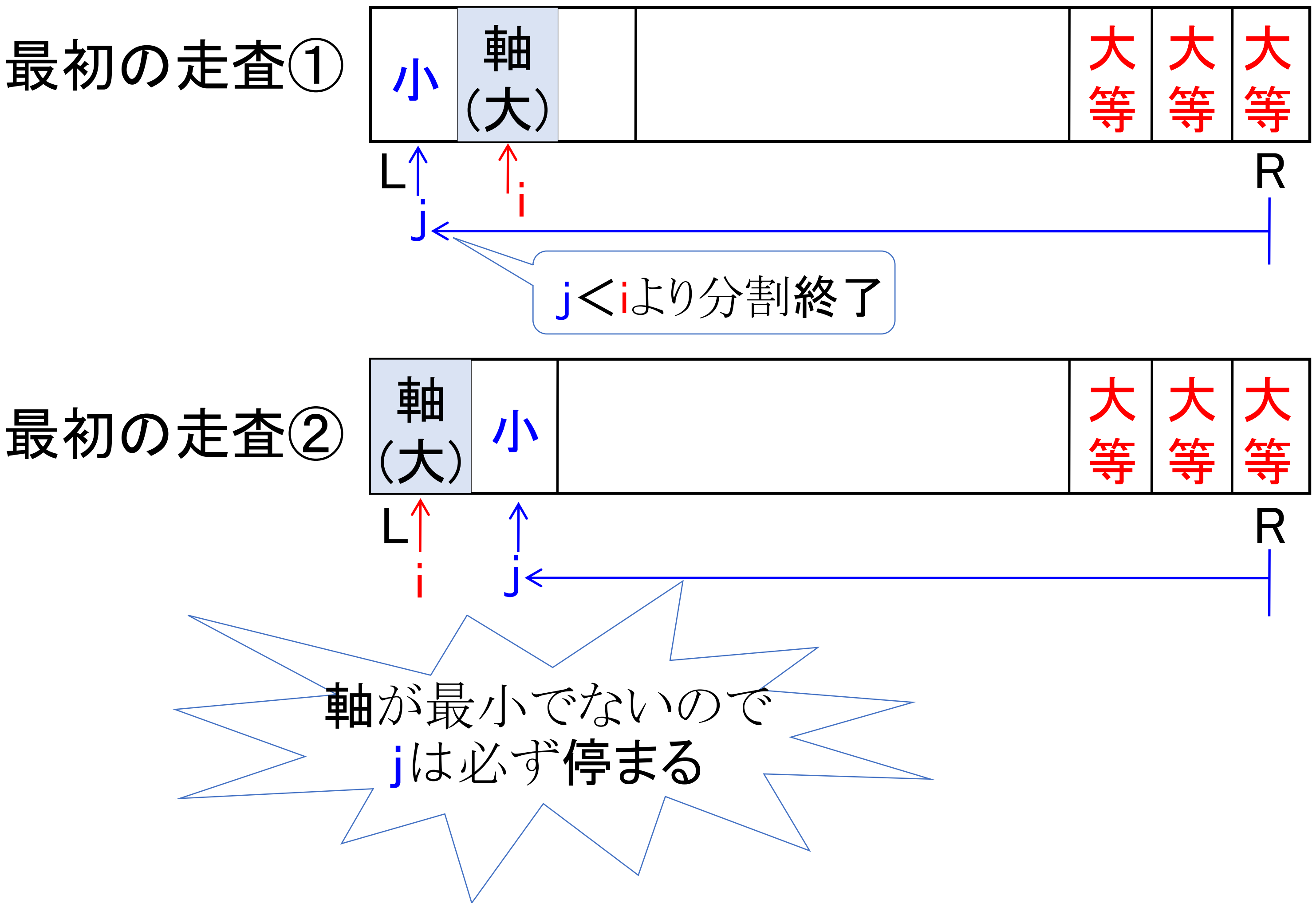
分割第２版の結果



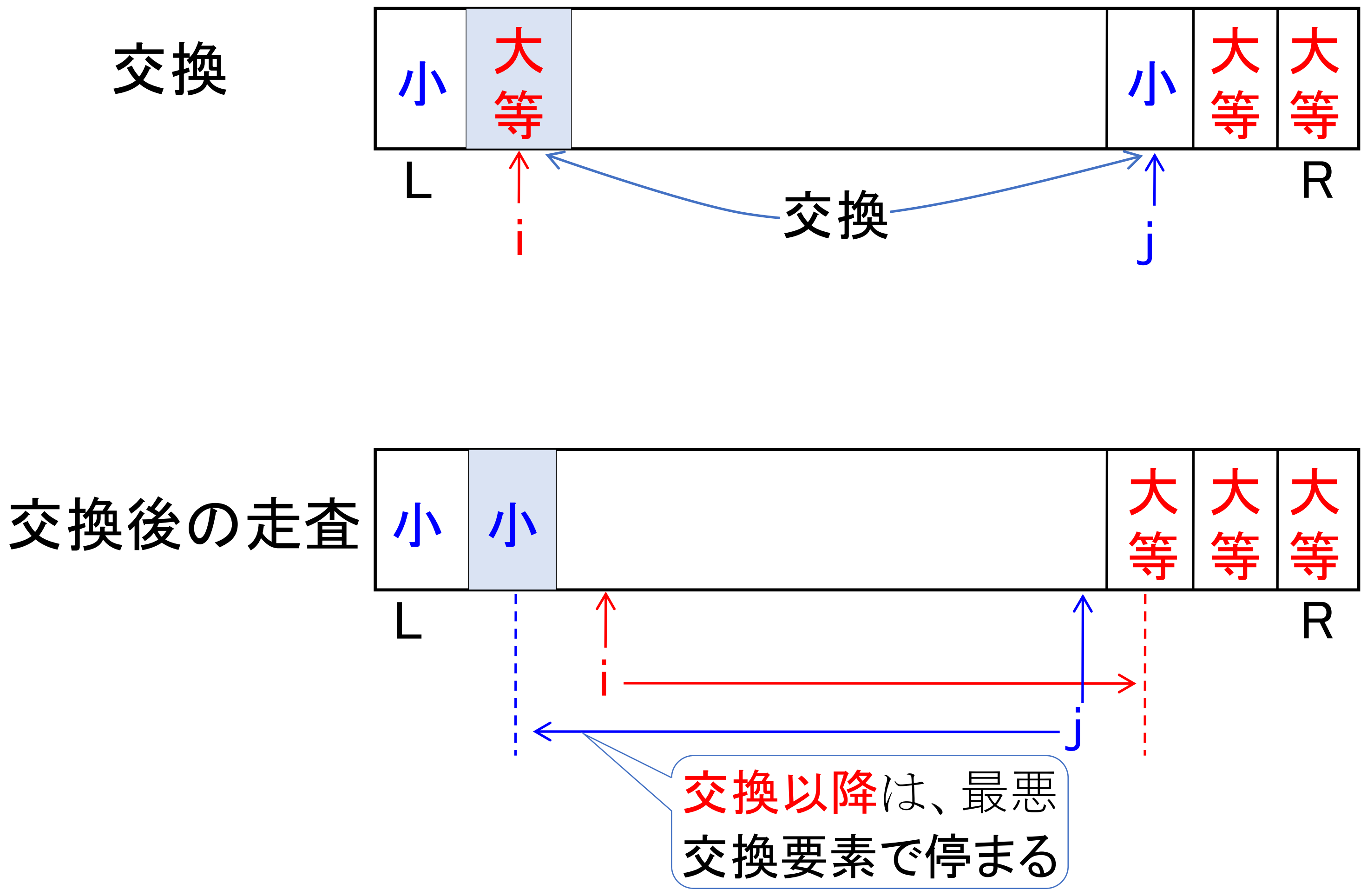
分割第２版例：44, 55, 12, 42, 94, 18, 06, 67

軸選択	44	55 軸	12	42	94	18	06	67
	L=0							R=7
走査	44	55	12	42	94	18	06	67
		i→軸のキー以上走査				軸のキー未満走査←j		
交換	44	55	12	42	94	18	06	67
		i					j	
走査	44	06	12	42	94	18	55	67
			i→次の位置から走査←j					
交換	44	06	12	42	94	18	55	67
					i	j		
終了	44	06	12	42	18	94	55	67
	L				j	i		R
			＜55				≧55	

分割第２版：最初の走査の停止



分割第２版：交換後の走査の停止



例：最小要素で必ずｊが停止

1　2 軸　2　2　3　3　3　3

i→2以上を走査　2未満を走査←j

走査後終了　1 | 2 軸　2　2　3　3　3　3

j　i

2 軸　1　2　2　3　3　3　3

i→2以上を走査　2未満を走査←j

2　1　2　2　3　3　3　3

i　j

交換後終了　1 | 2　2　2　3　3　3　3

j　i

クイックソートの原理（第２版）

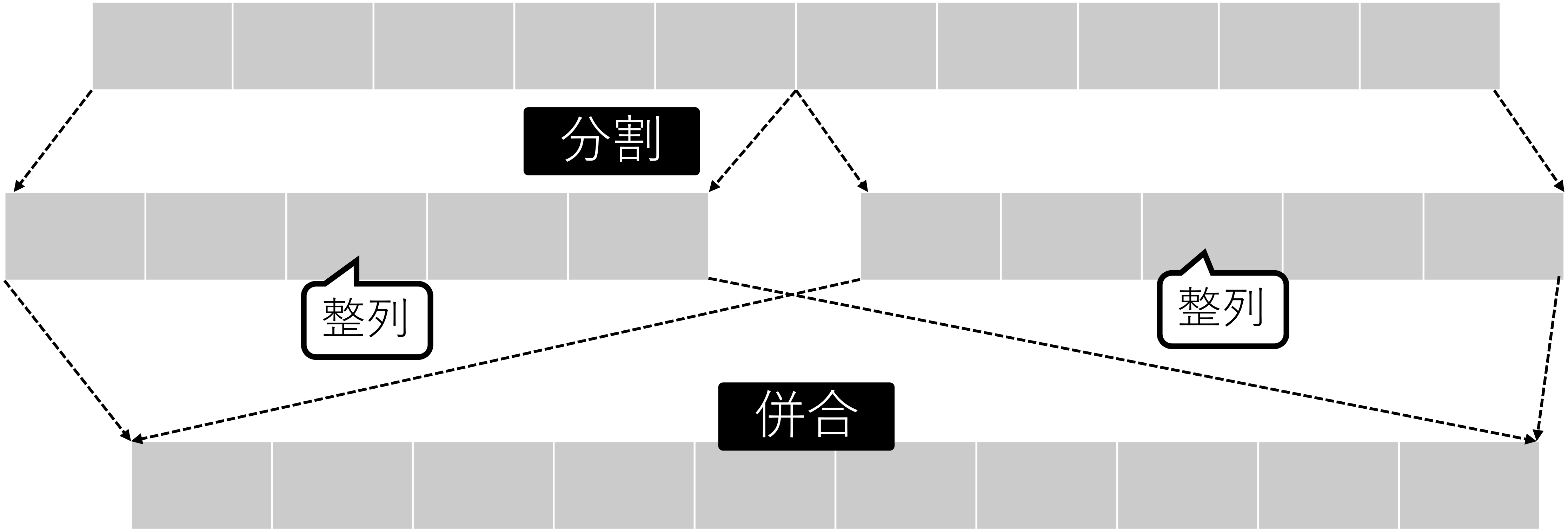
- 初期列として A[0],…, A[n-1] を与える
  - L=0; R=n-1;
- 部分列 A[L],…, A[R] を，分割（第２版）を使い分割
  - A[L],…, A[j] とA[i],…, A[R] に分割
  - 各部分配列の要素数が
    - 2個以上：分割を繰り返す
    - 2個未満：分割終了

クイックソート（第２版）の例

マージソート

マージソート

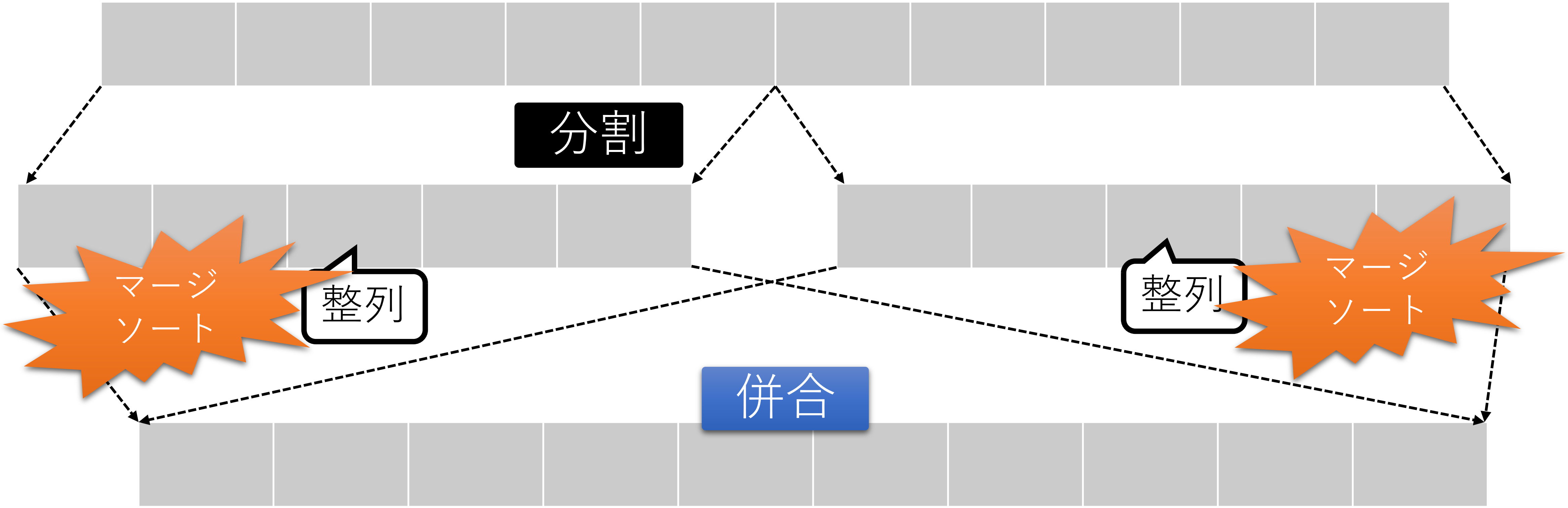
配列を前半と後半に分割して、それぞれを整列した後に併合する方法



マージソート

分割統治法

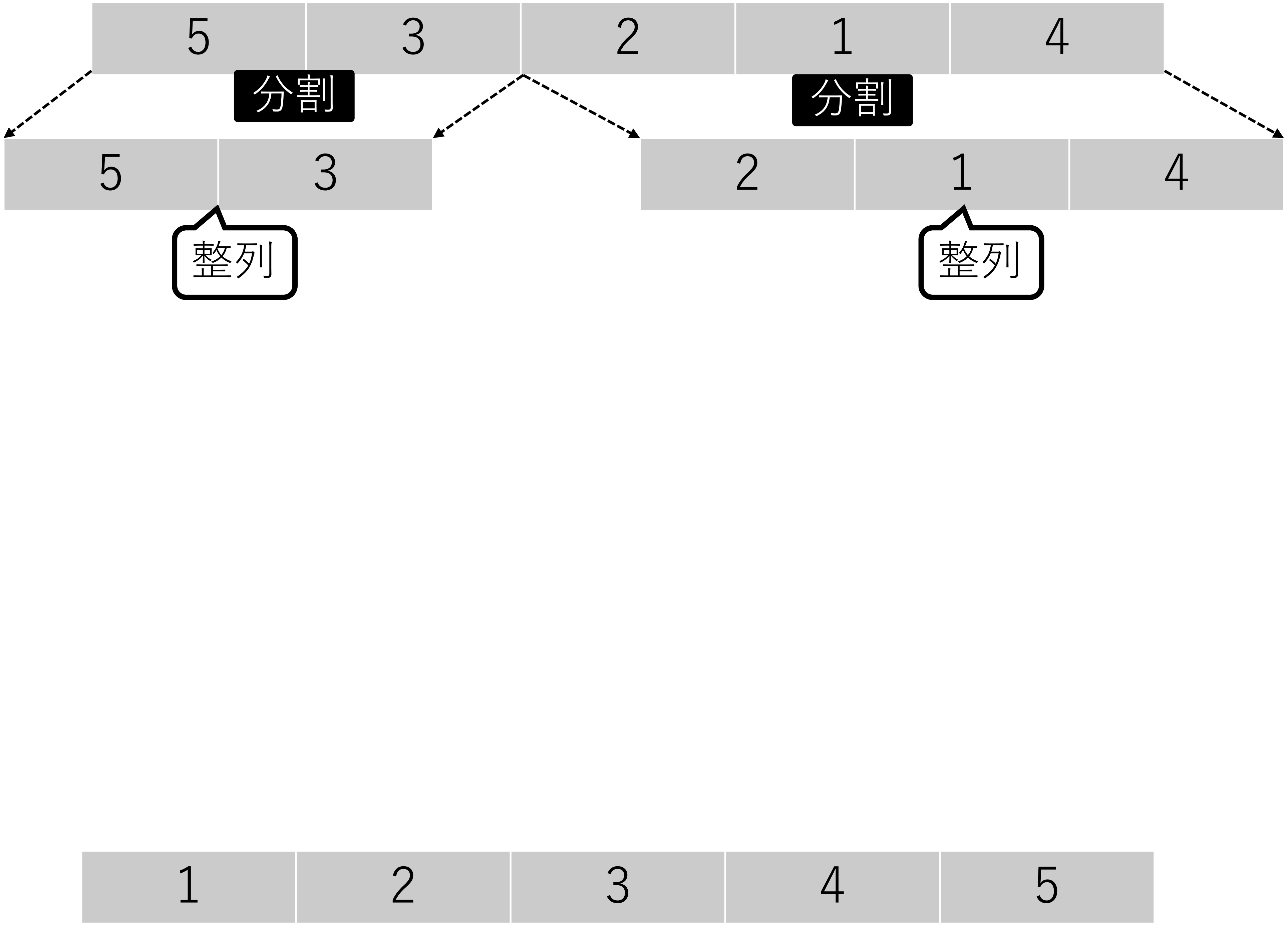
配列を前半と後半に分割して、それぞれを整列した後に併合する方法



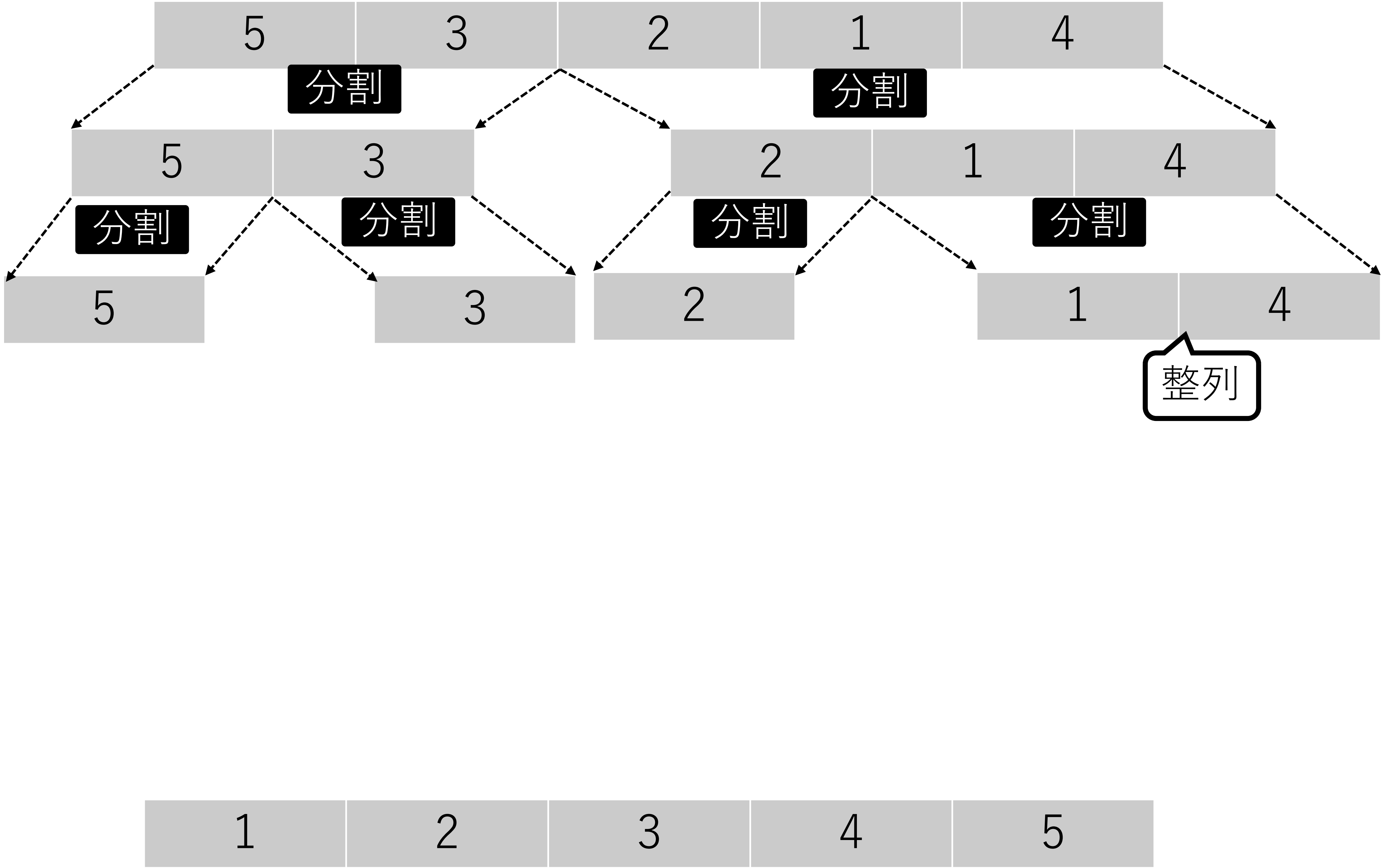
5	3	2	1	4

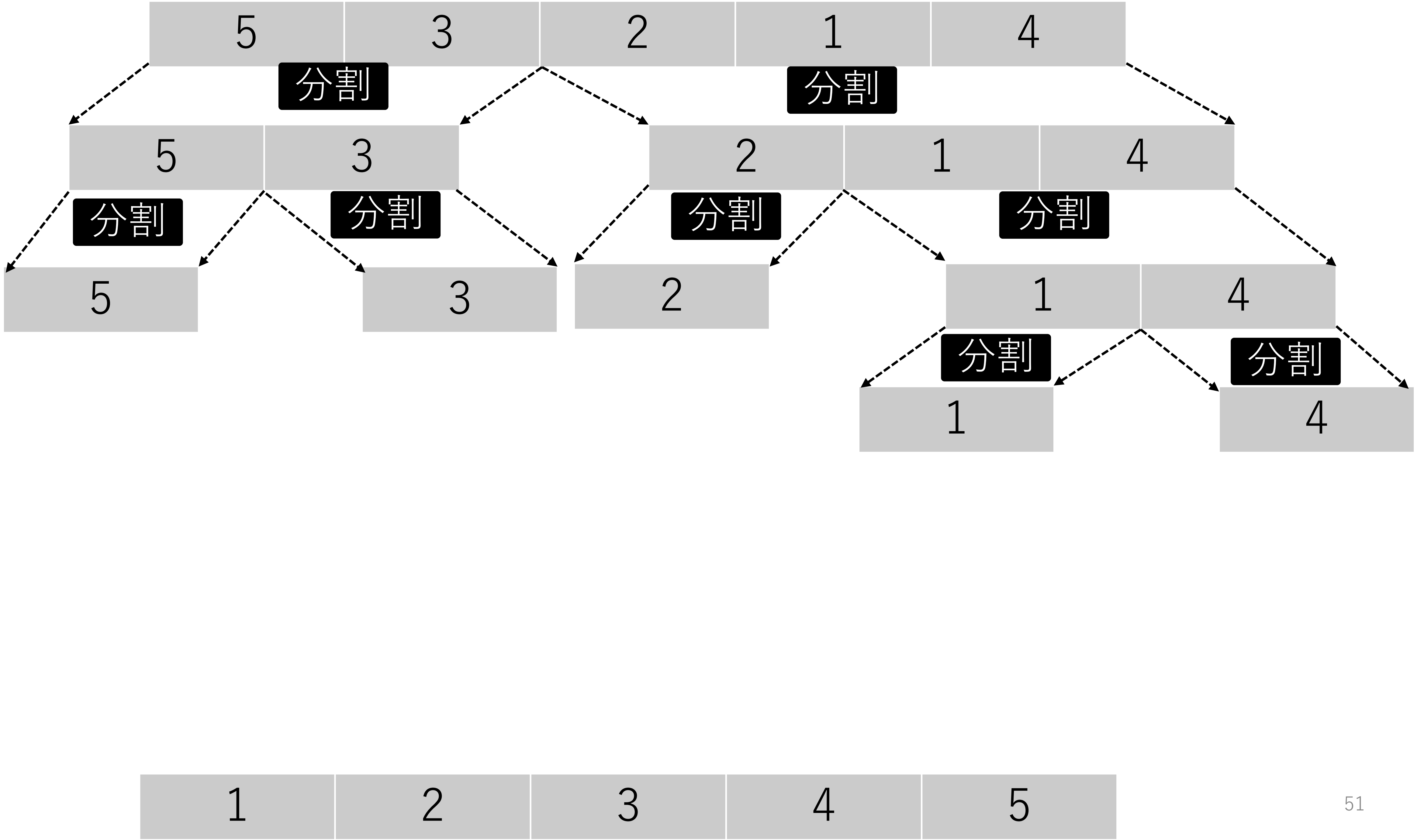
整列

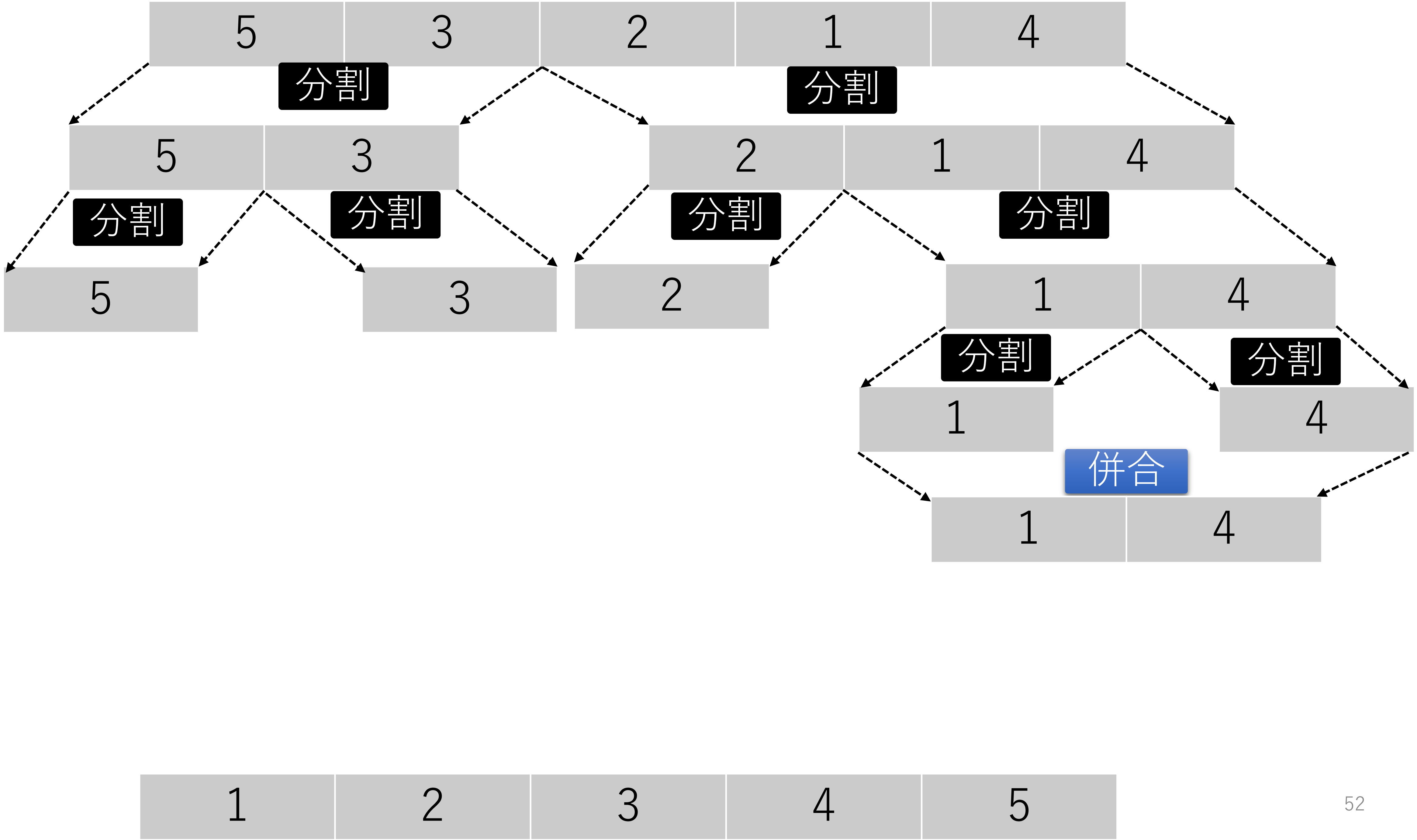
1	2	3	4	5

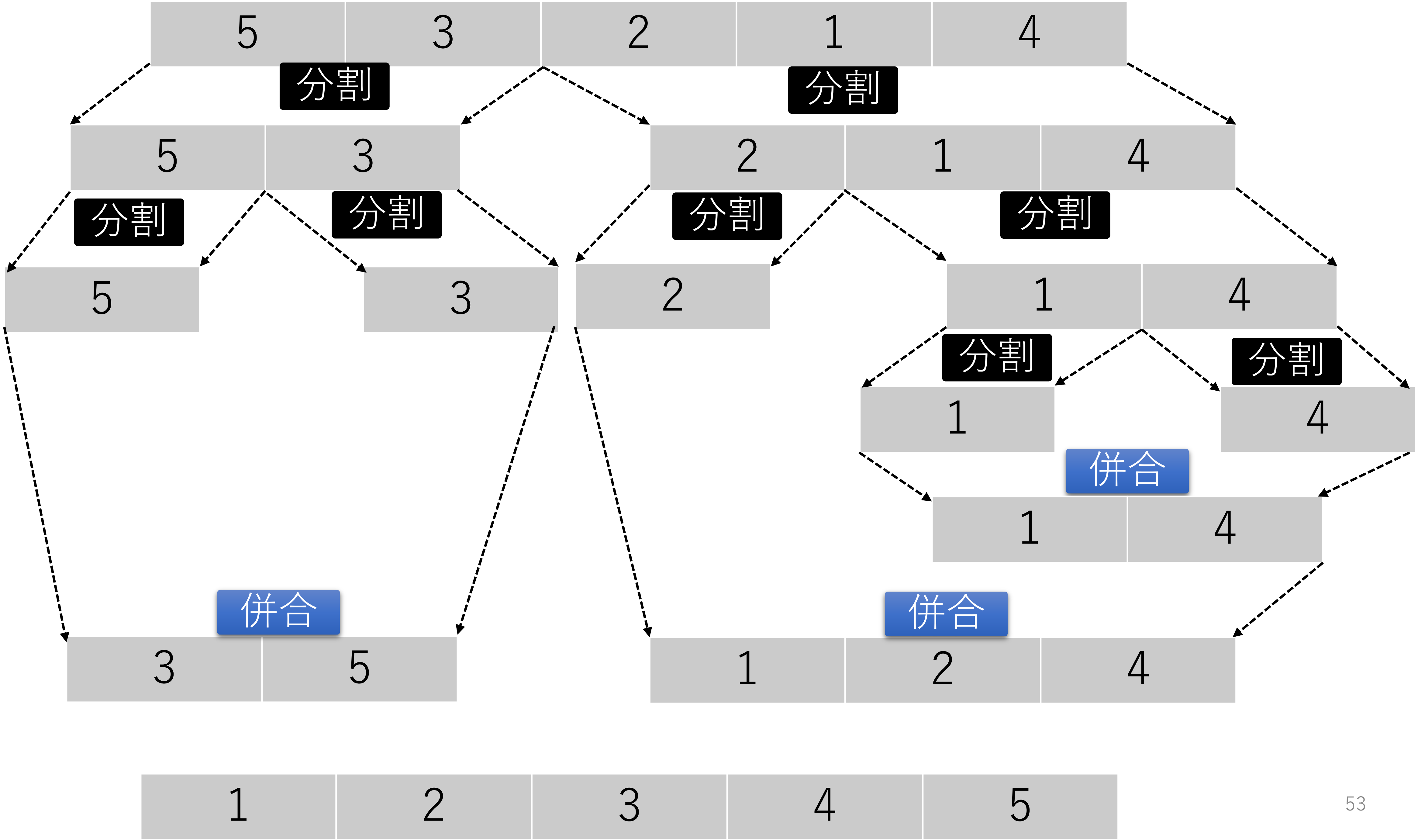


5 3 2 1 4
分割
分割
5 3
2 1 4
整列
整列
1 2 3 4 5

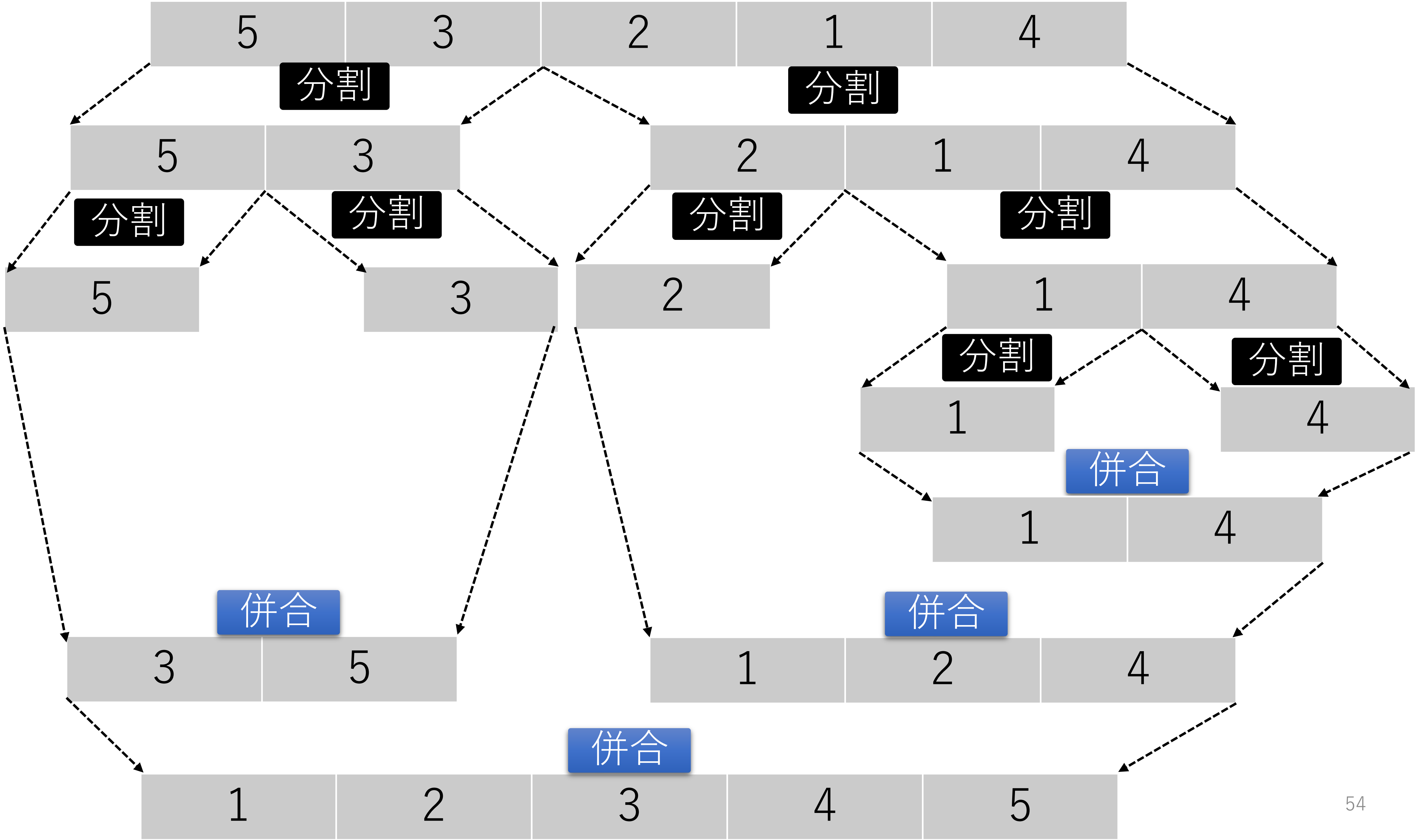

5 3 2 1 4
分割
分割
5 3
2 1 4
分割
分割
分割
分割
5
3
2
1 4
整列
1 2 3 4 5


5 3 2 1 4
分割
分割
5 3
2 1 4
分割
分割
分割
分割
5
3
2
1 4
分割
分割
1
4
1 2 3 4 5


5 3 2 1 4
分割
分割
5 3
2 1 4
分割
分割
分割
分割
5
3
2
1 4
分割
分割
1
4
併合
1 4
1 2 3 4 5



5 3 2 1 4
分割
分割
5 3
2 1 4
分割
分割
分割
分割
5
3
2
1 4
分割
分割
1
4
併合
1 4
併合
3 5
併合
1 2 4
1 2 3 4 5


5 3 2 1 4
分割
分割
5 3
2 1 4
分割
分割
分割
分割
5
3
2
1 4
分割
分割
1
4
併合
1 4
併合
3 5
併合
1 2 4
併合
1 2 3 4 5

マージソート

Algorithm: マージソート Merge_Sort_Entire_List
Input: 配列 $A = [a_0, a_1, \ldots, a_{n-1}]$
Output: $A(a_0 \leq a_1 \leq \cdots \leq a_{n-1})$

1. Merge_Sort$(A, 0, n-1)$

配列全体の整列

部分配列（前半あるいは後半）の整列

Algorithm: 部分配列のマージソート Merge_Sort
Input: 配列 $A = [a_0, a_1, \ldots, a_{n-1}]$，先頭の添え字 $start$, 最後の添え字 end
Output: 部分配列 $[a_{start}, \ldots, a_{end}]$が昇順に整列した$A$

1. **if** $start < end$ **then**
2. $\quad middle \leftarrow \left\lfloor \frac{start+end}{2} \right\rfloor$ {$\lfloor x \rfloor$はxの整数部分}
3. $\quad$Merge_Sort$(A, start, middle)$ {前半部分配列の整列}
4. $\quad$Merge_Sort$(A, middle+1, end)$ {後半部分配列の整列}
5. $\quad$Merge_Sublists$(A, start, middle, end)$
6. **end if**

マージソート

Algorithm: マージソート Merge_Sort_Entire_List
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$
Output: $A(a_0 \leq a_1 \leq \cdots \leq a_{n-1})$

1. Merge_Sort($A, 0, n - 1$)

配列全体を部分配列とみなして整列

Algorithm: 部分配列のマージソート Merge_Sort
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$，先頭の添え字 $start$, 最後の添え字 end
Output: 部分配列 $[a_{start}, \dots, a_{end}]$が昇順に整列した$A$

部分配列の要素数が1かどうかチェック

1. **if** $start < end$ **then**
2. $middle \leftarrow \left\lfloor \frac{start+end}{2} \right\rfloor$ {$\lfloor x \rfloor$はxの整数部分}
3. Merge_Sort($A, start, middle$) {前半部分配列の整列}
4. Merge_Sort($A, middle + 1, end$) {後半部分配列の整列}
5. Merge_Sublists($A, start, middle, end$)
6. **end if**

中央の添え字を計算

配列を分割して
再帰的に整列

整列した部分配列を併合

部分配列の併合関数

Algorithm: 前半部分配列と後半部分配列を併合する Merge_Sublists
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$， 前半部分配列の先頭の添え字 $start$,前半部分配列の最後の添え字 $middle$, 後半部分配列の最後の添え字 end
Output: 前半部分配列と後半部分配列の要素の順序を変えずに併合したA

```
W = [w_0, …, w_{end−start}] ← [a_start, …, a_end]
i ← 0
j ← middle − start + 1
for k ← start to end do
   if (end − start) < j or (i ≤ (middle − start) and w_i ≤ w_j) then
      a_k ← w_i
      i ← i + 1
   else
      a_k ← w_j
      j ← j + 1
   end if
end for
```

部分配列の併合関数

Algorithm: 前半部分配列と後半部分配列を併合する Merge_Sublists
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$， 前半部分配列の先頭の添え字 $start$,前半部分配列の最後の添え字 $middle$, 後半部分配列の最後の添え字 end
Output: 前半部分配列と後半部分配列の要素の順序を変えずに併合したA

1. $W = [w_0, \dots, w_{end-start}] \leftarrow [a_{start}, \dots, a_{end}]$
2. $i \leftarrow 0$
3. $j \leftarrow middle - start + 1$
4. **for** $k \leftarrow start$ **to** end **do**
5. **if** $(\mathrm{end} - start) < j$ **or** $(i \leq (middle - start)$ **and** $w_i \leq w_j)$ **then**
6. $a_k \leftarrow w_i$
7. $i \leftarrow i + 1$
8. **else**
9. $a_k \leftarrow w_j$
10. $j \leftarrow j + 1$
11. **end if**
12. **end for**

併合によりAを書き換えるため、
Wに元の部分配列を一時的に格納

$[w_0, \dots, w_{middle-start}]$ と
$[w_{middle-start+1}, \dots, w_{end}]$
の併合

部分配列の併合関数

Algorithm: 前半部分配列と後半部分配列を併合する Merge_Sublists
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$， 前半部分配列の先頭の添え字 $start$,前半部分配列の最後の添え字 $middle$, 後半部分配列の最後の添え字 end
Output: 前半部分配列と後半部分配列の要素の順序を変えずに併合したA

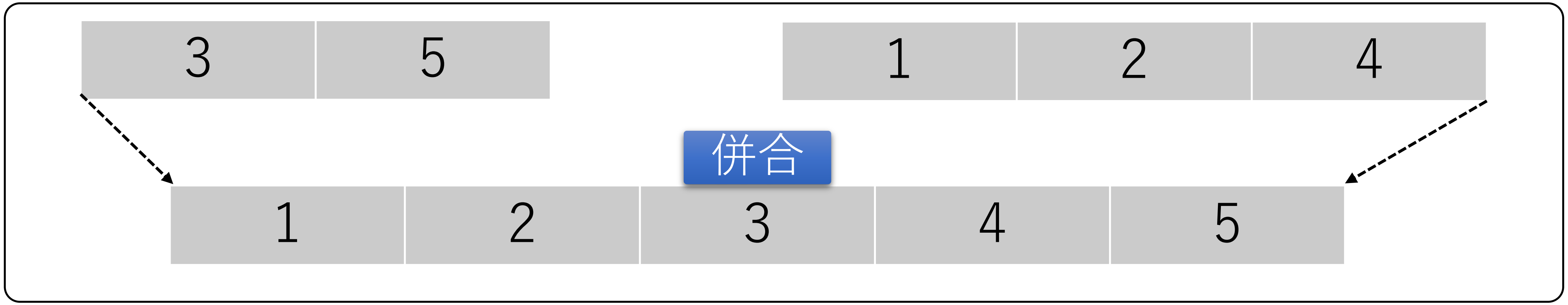
$W = [w_0, \dots, w_{end-start}] \leftarrow [a_{start},$
　$i \leftarrow 0$
　$j \leftarrow middle - start + 1$
　for $k \leftarrow start$ **to** end **do**
　　if $(\mathrm{end} - start) < j$ **or** $(i \le (middle - start)$ **and** $w_i \le w_j)$ **then**
　　　$a_k \leftarrow w_i$
　　　$i \leftarrow i + 1$
　　else
　　　$a_k \leftarrow w_j$
　　　$j \leftarrow j + 1$
　　end if
　end for

- iは、前半部分配列で次に併合する要素の添え字
- jは、後半部分配列で次に併合する要素の添え字
- kは、次に併合した要素を格納するAの添え字

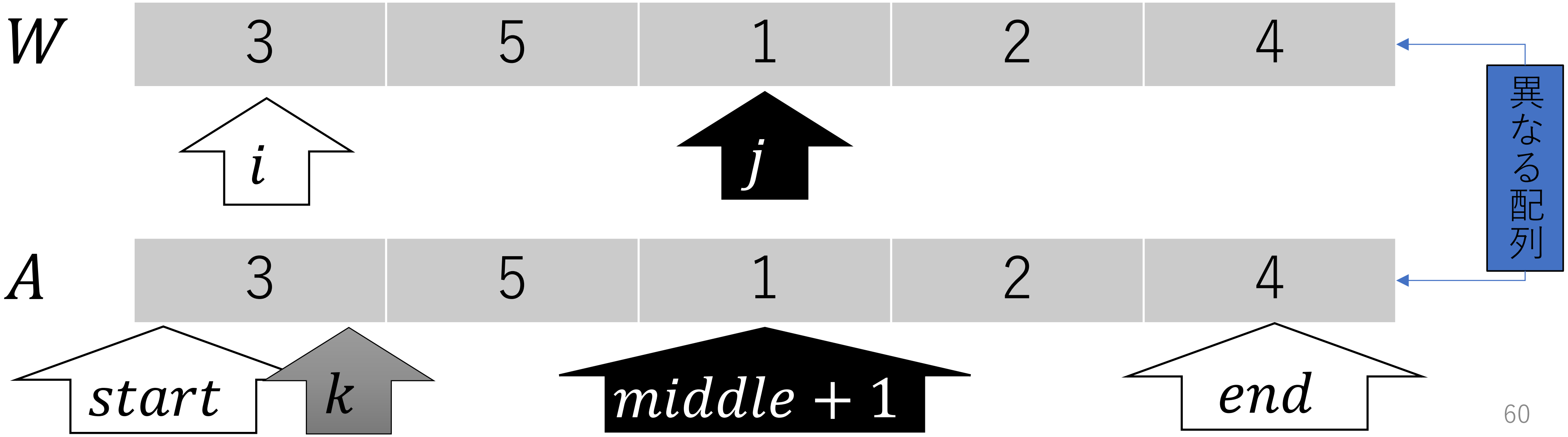
前半部分配列で次に併合する要素 ≤ 後半部分配列で次に併合する要素ならば、前半部分配列の要素を併合

後半部分配列の要素を併合

併合操作の例

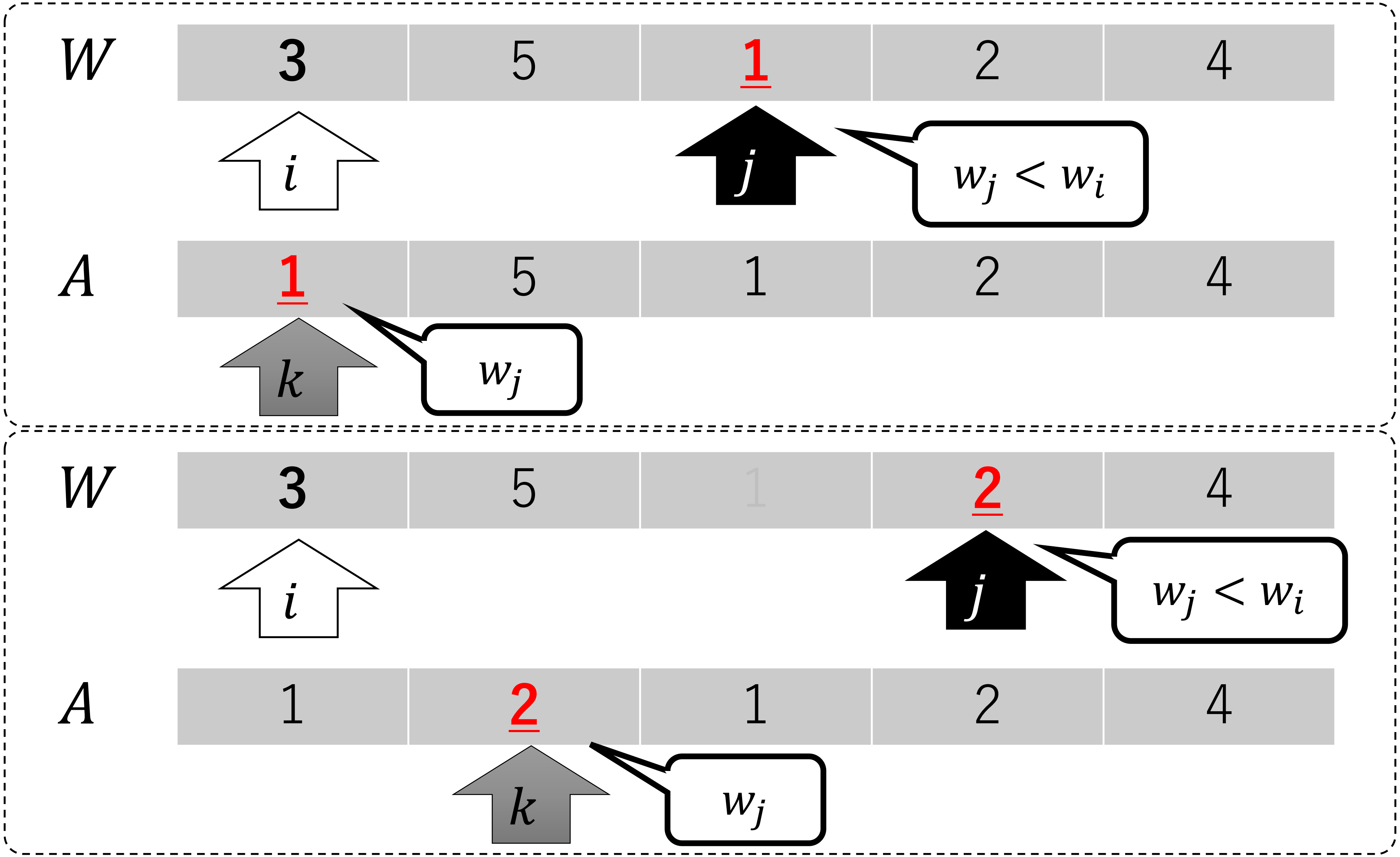


併合開始前



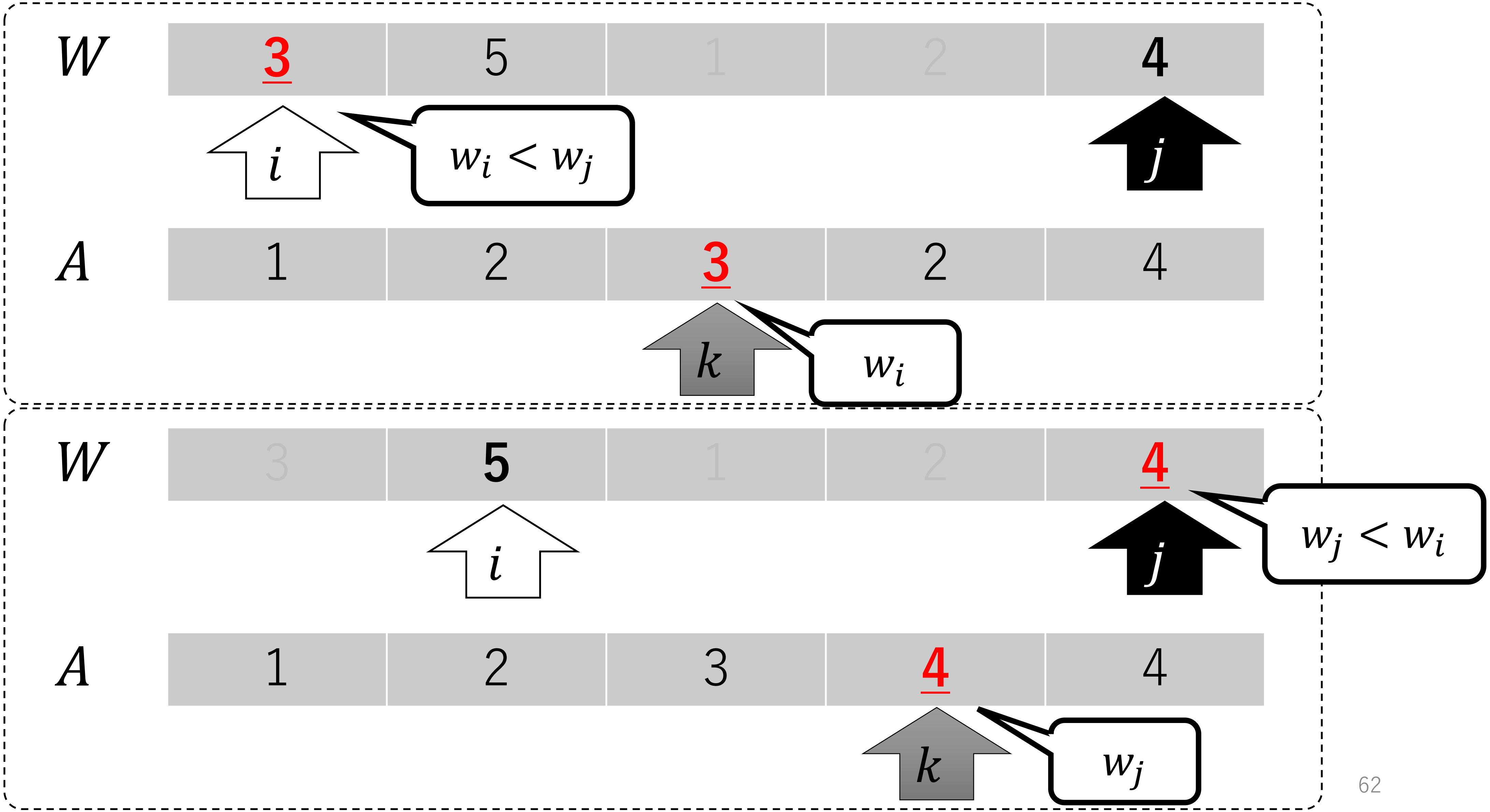
併合操作の例

併合中

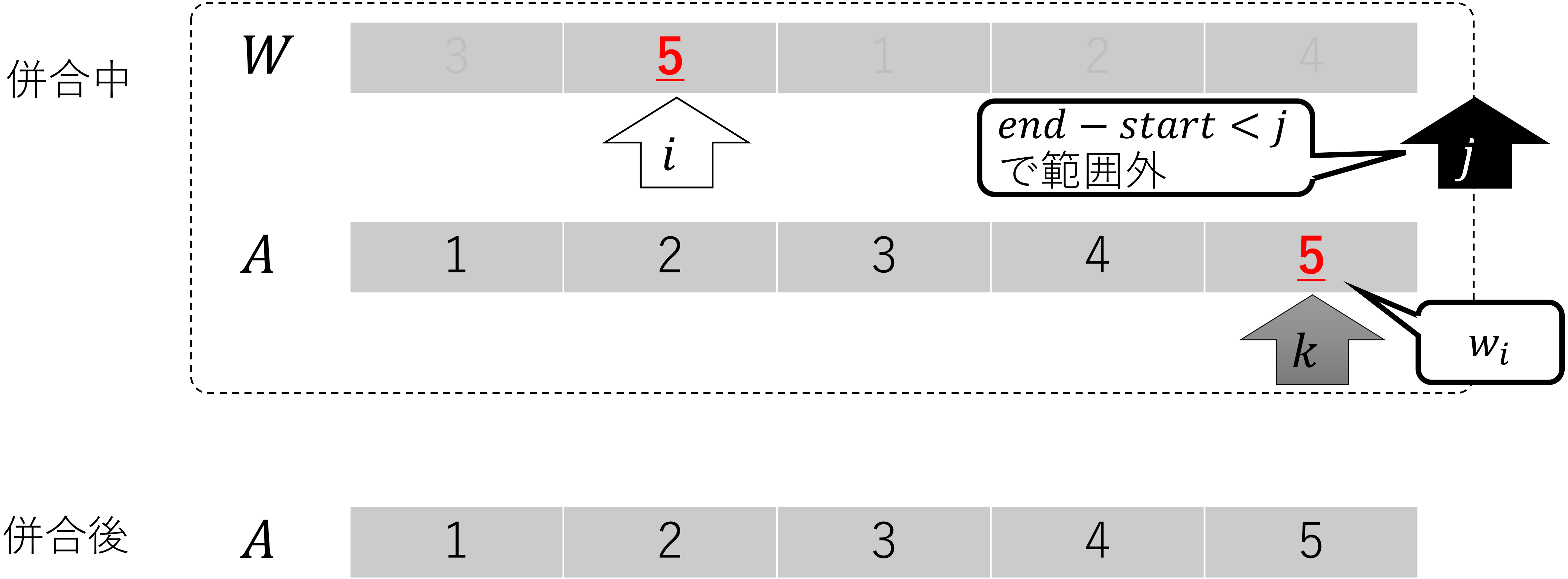


併合操作の例

併合中



併合操作の例



部分配列が整列済みであるという事実を利用して併合を進める

マージソート

Algorithm: マージソート Merge_Sort_Entire_List
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$
Output: $A(a_0 \le a_1 \le \cdots \le a_{n-1})$

1. Merge_Sort$(A, 0, n - 1)$

Algorithm: 部分配列のマージソート Merge_Sort
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$, 先頭の添え字 $start$
end
Output: 部分配列 $[a_{start}, \dots, a_{end}]$が昇順に整列した$A$

1. **if** $start < end$ **then**
2. $middle \leftarrow \left\lfloor \frac{start+end}{2} \right\rfloor$ {$\lfloor x \rfloor$はxの整数部分}
3. Merge_Sort$(A, start, middle)$ {前半部分配列の
4. Merge_Sort$(A, middle + 1, end)$ {後半部分配
5. Merge_Sublists$(A, start, middle, end)$
6. **end if**

merge_sort.py

```python
def merge_sort_entire_list(seq):
    merge_sort(seq, 0, len(seq) - 1)

def merge_sort(seq, start, end):
    if start < end:
        middle = (start + end) // 2
        merge_sort(seq, start, middle)
        merge_sort(seq, middle + 1, end)
        merge_sublists(seq, start, middle, end)

a = [5, 3, 2, 1, 4]
merge_sort_entire_list(a)
print(a)
```

```
▶ merge_sort.py
[1, 2, 3, 4, 5]
```

部分配列の併合関数

Algorithm: 前半部分配列と後半部分配列を併合する Merge_Sublists
Input: 配列 $A = [a_0, a_1, \dots, a_{n-1}]$, 前半部分配列の先頭の添え字 $start$,前半部分配列の最後の添え字 $middle$, 後半部分配列の最後の添え字 end
Output: 前半部分配列と後半部分配列の要素の順序

$W = [w_0, \dots, w_{end-start}] \leftarrow [a_{start}, \dots, a_{end}]$
$i \leftarrow 0$
$j \leftarrow middle - start + 1$
for $k \leftarrow start$ **to** end **do**
  if $(\mathrm{end} - start) < j$ **or** $(i \leq (middle - start$
    $a_k \leftarrow w_i$
    $i \leftarrow i + 1$
  else
    $a_k \leftarrow w_j$
    $j \leftarrow j + 1$
  end if
end for

m

```python
def merge_sublists(seq, start, middle, end):
    w = seq[start: end + 1]
    i = 0
    j = middle - start + 1
    for k in range(start, end + 1):
        if (end - start) < j or (i <= (middle - start) a
            seq[k] = w[i]
            i = i + 1
        else:
            seq[k] = w[j]
            j = j + 1
```

分割統治法

- 大きな問題を効率的に解く手法
- 問題全体を同じ構造の小さな問題に再帰的に分割していき、簡単に解けるサイズにした上で解いていく方式

- どちらも配列を部分配列に分割していく方式
  - マージソート：前半と後半で単純に分割
  - クイックソート：ピボットとの大小関係をもとに分割

- クイックソートはピボットの選び方に任意性がある
  - 最後の要素でなくともよい
  - 選び方によって効率に影響がある（影響の具合は元の配列に依存する）

クイックソートとマージソート

- クイックソート　＝　分割ソート
- マージソート　＝　併合ソート

- 分割ソート
  - 分割するときがポイント
- 併合ソート
  - 併合するときがポイント

- 時間計算量は同じ（$n \log n$）
- 領域計算量はクイックソートが有利

まとめ

- クイックソート
- マージソート

演習問題

- 問題1：つぎの整数列を**クイックソート**せよ。ただし、図を用いて、要素の**移動・交換**を明示すること。
  - ２　７　４　５　３　６　１　４　５　３
- 問題2：つぎの整数列を**マージソート**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　３　５　７　１　８　６　２

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
  - 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

- 提出方法：LETUS
- 締め切り：202/5/30 10:40まで