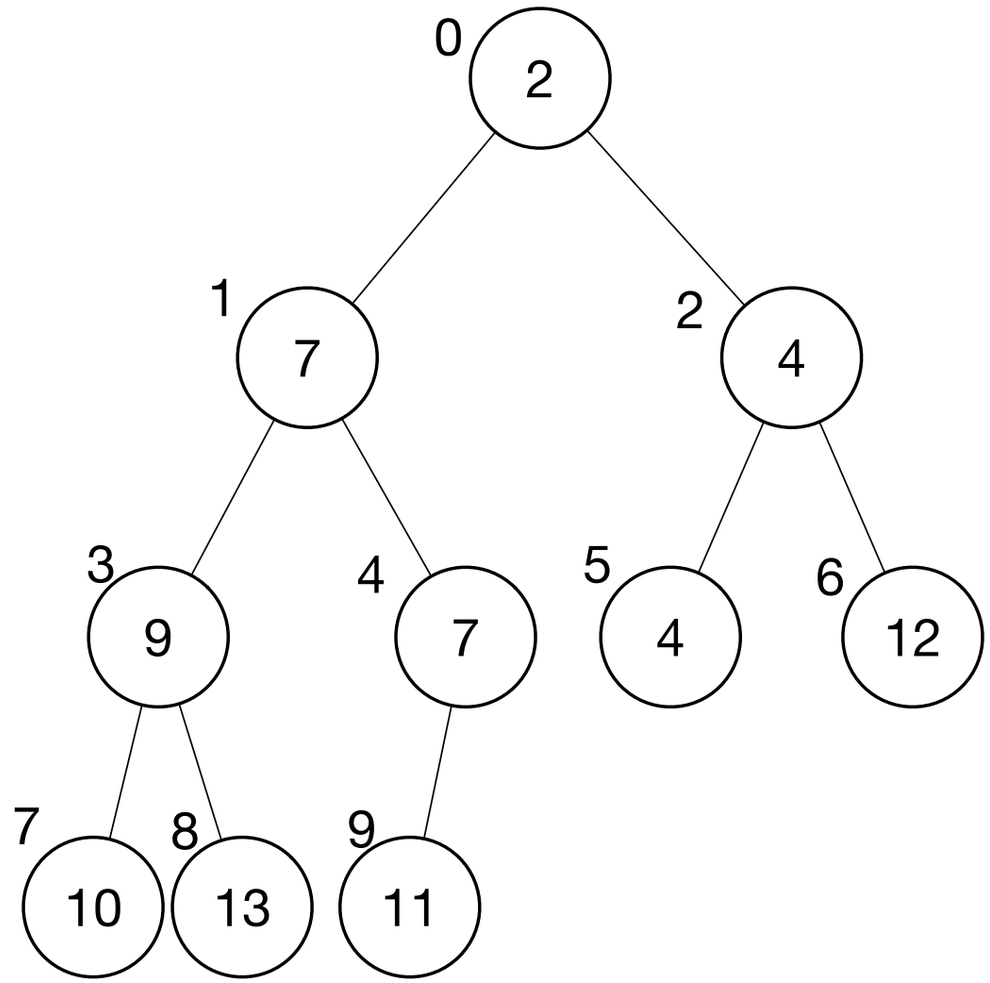
データ構造とアルゴリズム

第６回　整列(2)

小池 英樹 (*koike@c.titech.ac.jp*)

HEAPSORT

- ヒープの特徴（復習）
  - 親節点の値 ≦ 子節点の値
  - ルートが最小値
  - deletemin()で最小値を削除した後もヒープ条件は保存される



0	1	2	3	4	5	6	7	8	9	10
2	7	4	9	7	4	12	10	13	11	-

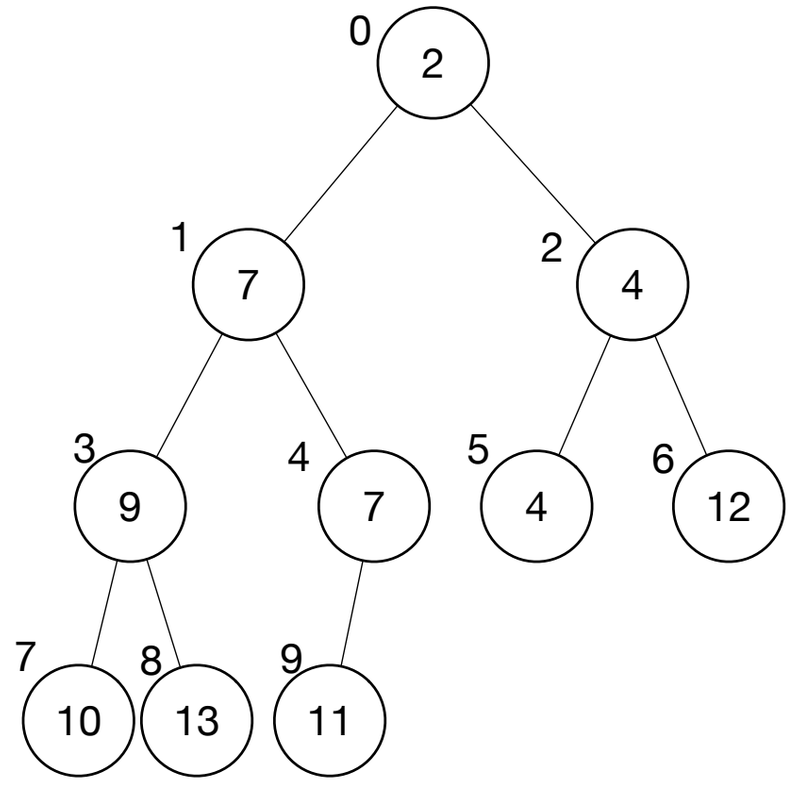
配列による実現(n=10, N=11)

HEAPSORT

- 考え方
  - 与えられた数の集合を順にヒープに追加(insert)
  - ヒープから最小要素を次々に削除(deletemin)して並べる
- アルゴリズム

```
/* 全要素をヒープに挿入. Hはヒープ */
for (i=0; i<n; i++)
    insert(A[i], H[i]);

/* 最小要素を削除して並べる */
for (i=0; i<n; i++)
    A[i] = deletemin(H);
```



HEAPSORTの計算時間

- 計算時間
  - insert()がO(log n)なので，最初のループがO(n log n)
  - deletemin()がO(log n)なので，次のループがO(n log n)
  - 結局，$O(n \log n) + O(n \log n) = O(n \log n)$

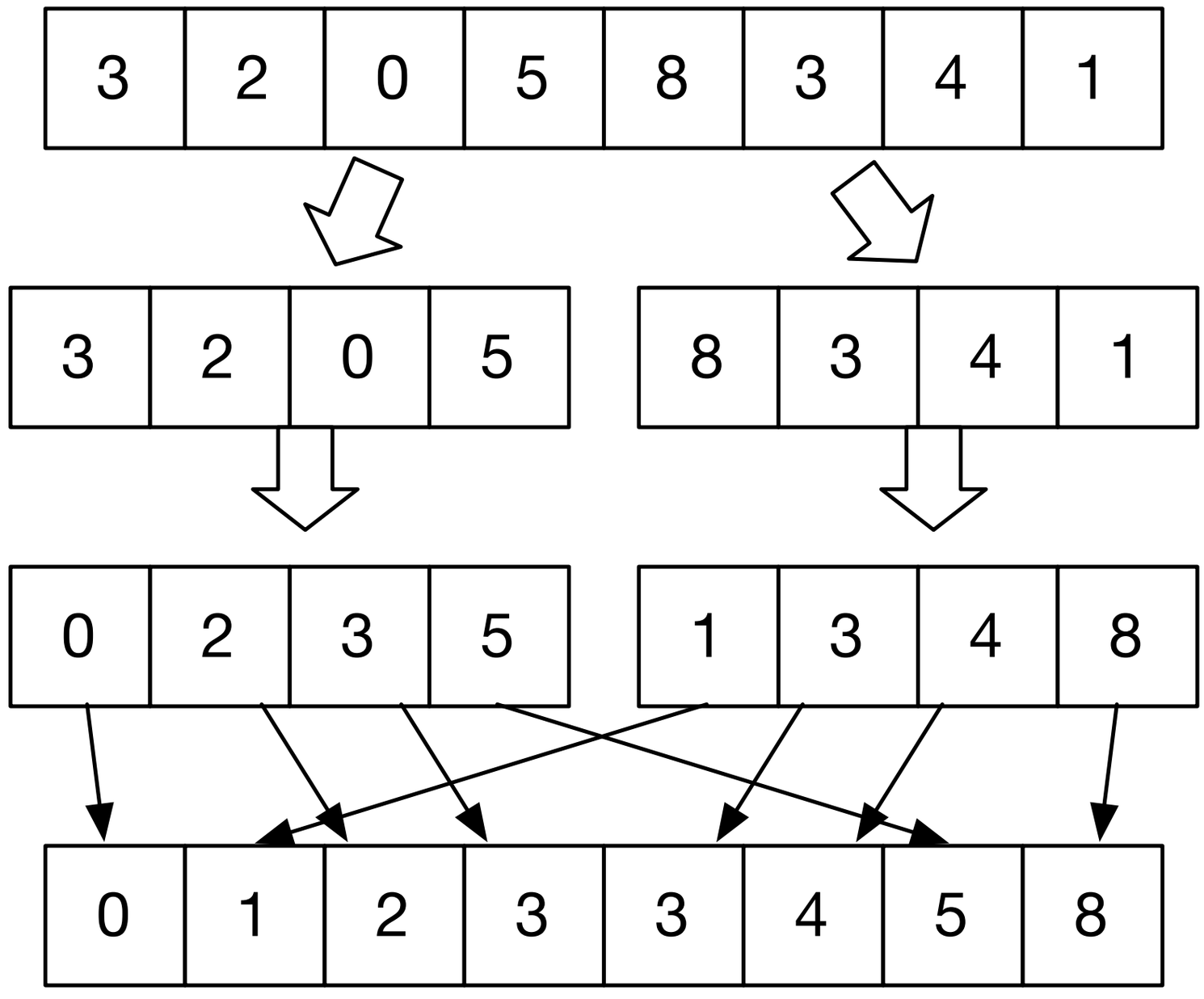
MERGE SORT

- 基本的な考え方

配列を２等分

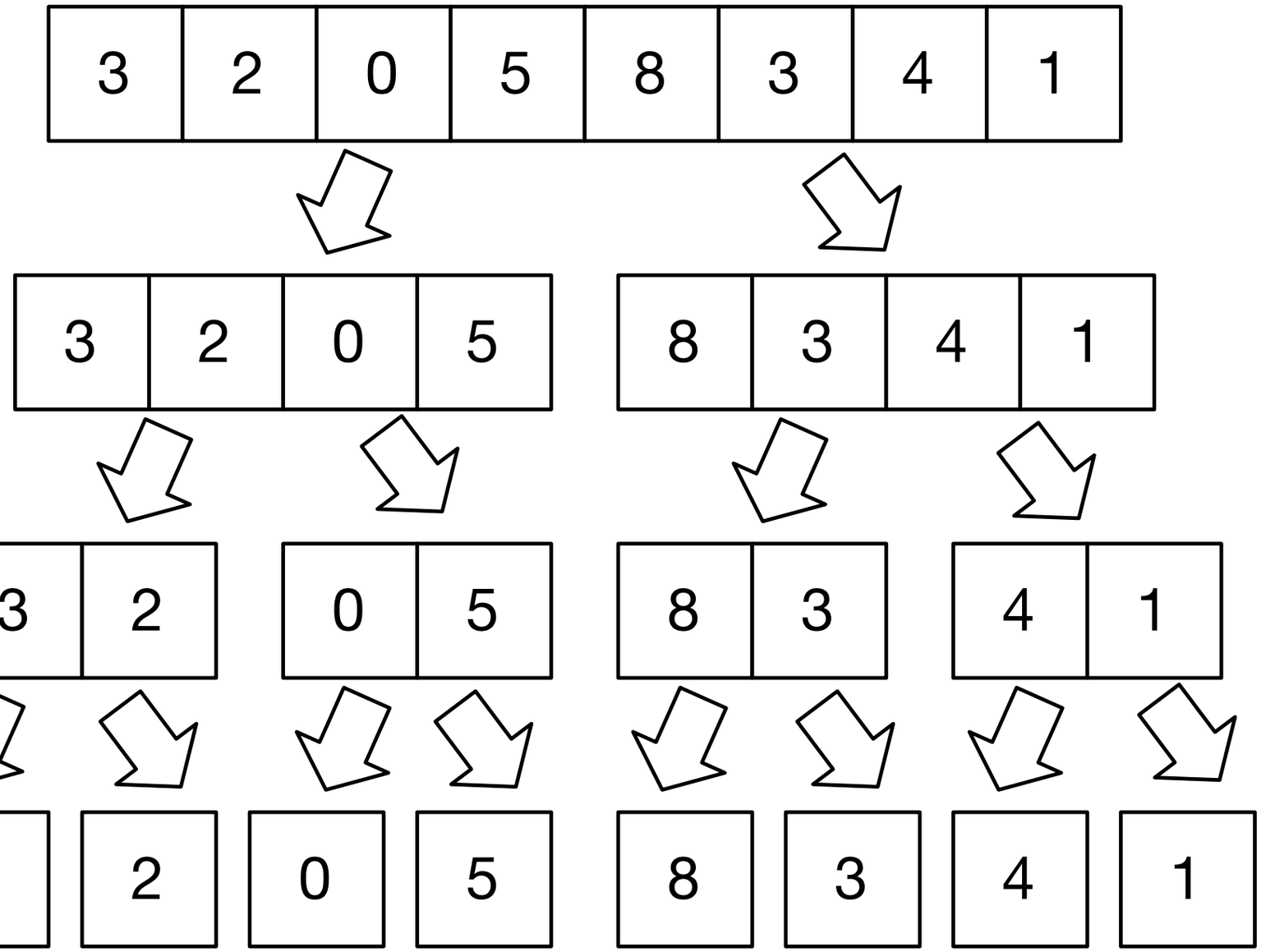
それぞれをソート

２つの配列をマージ



MERGE SORTの動作（前半）

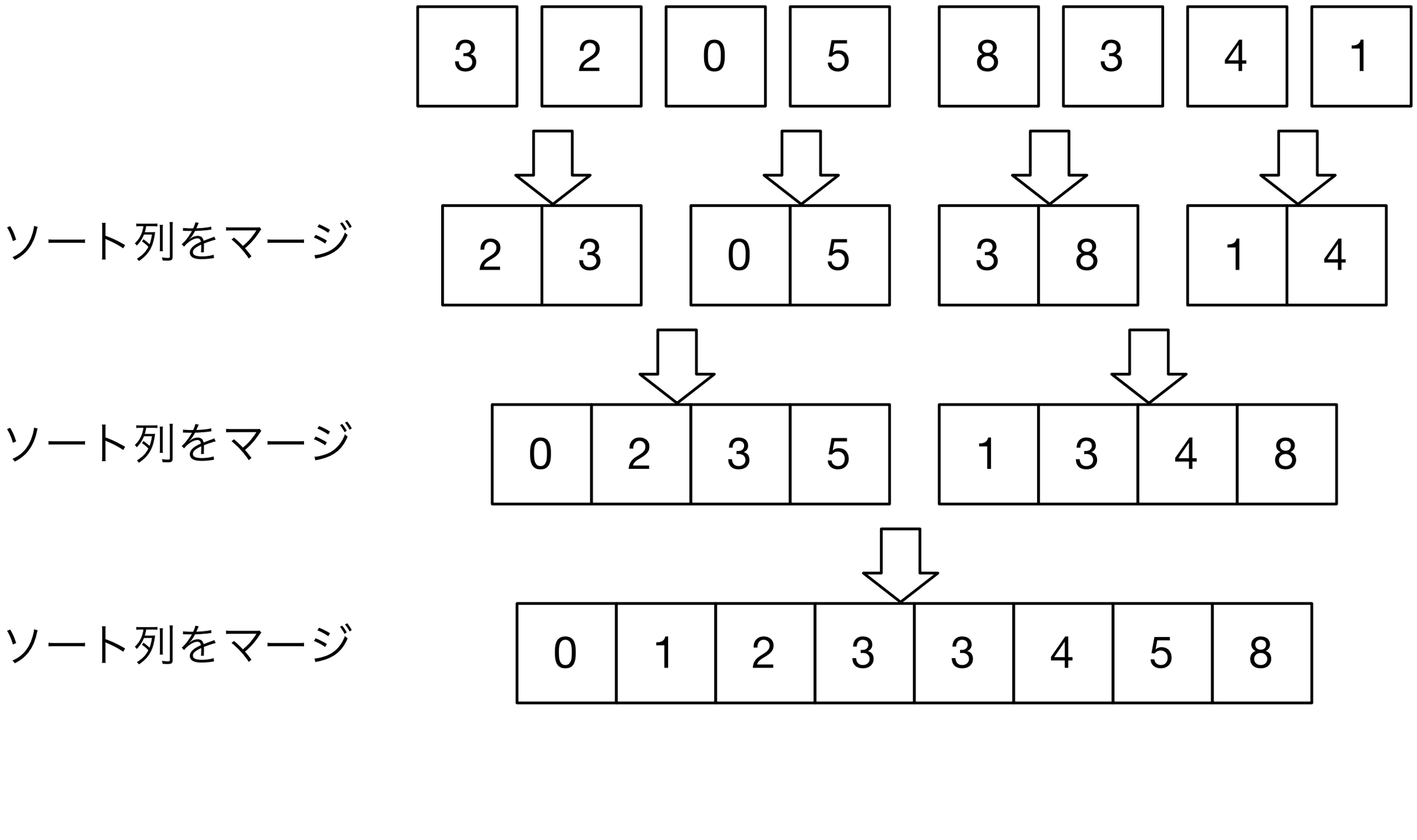
配列を２等分



配列を２等分

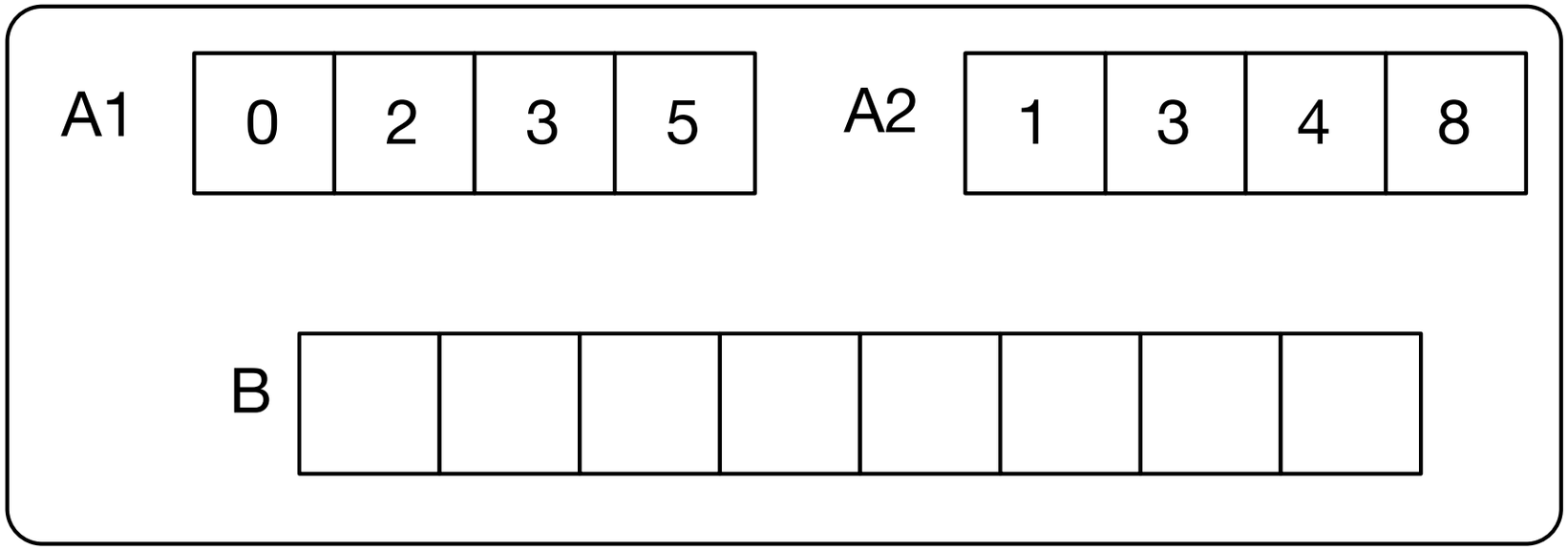
配列を２等分

MERGE SORTの動作（後半）

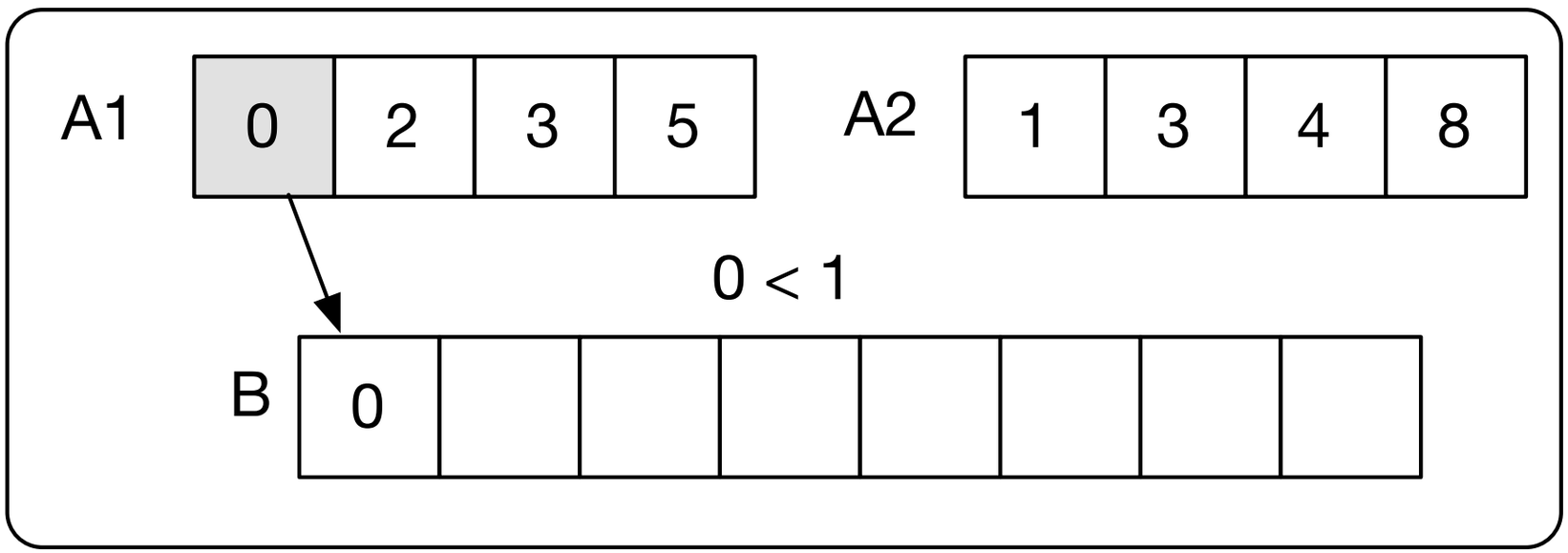


ソート列のマージ

結果を格納する配列
Bを準備

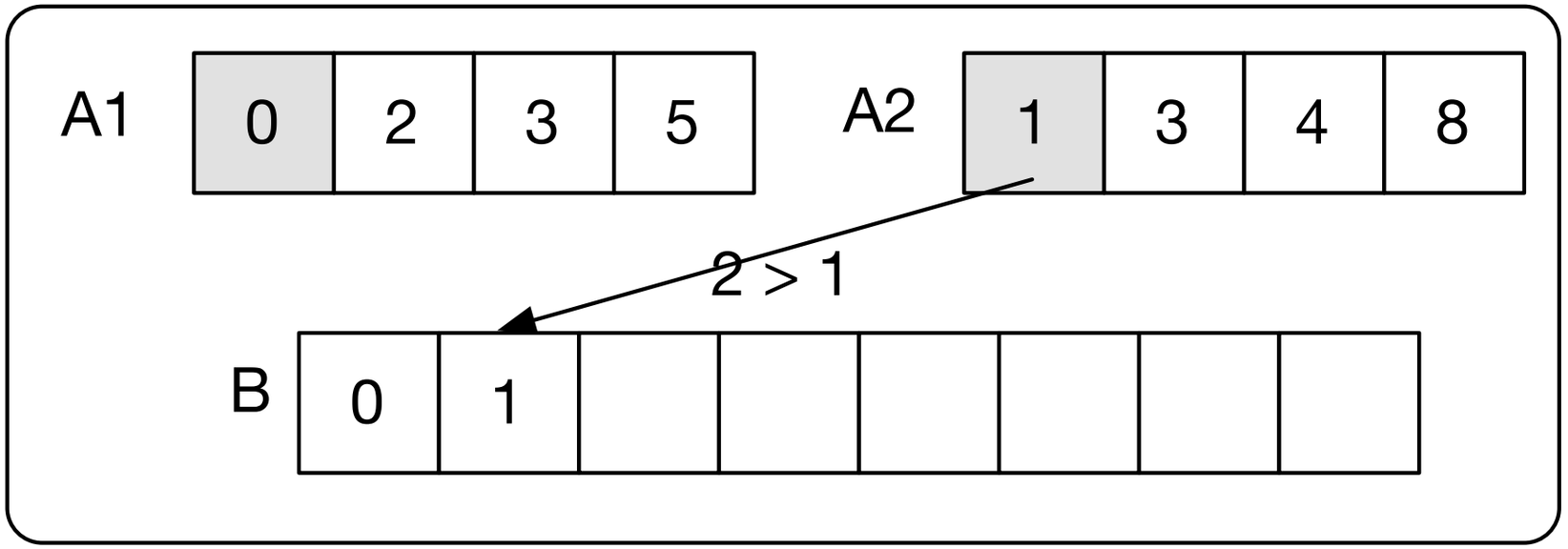


A1とA2の先頭を比較し，小さい方をBの空欄の先頭に

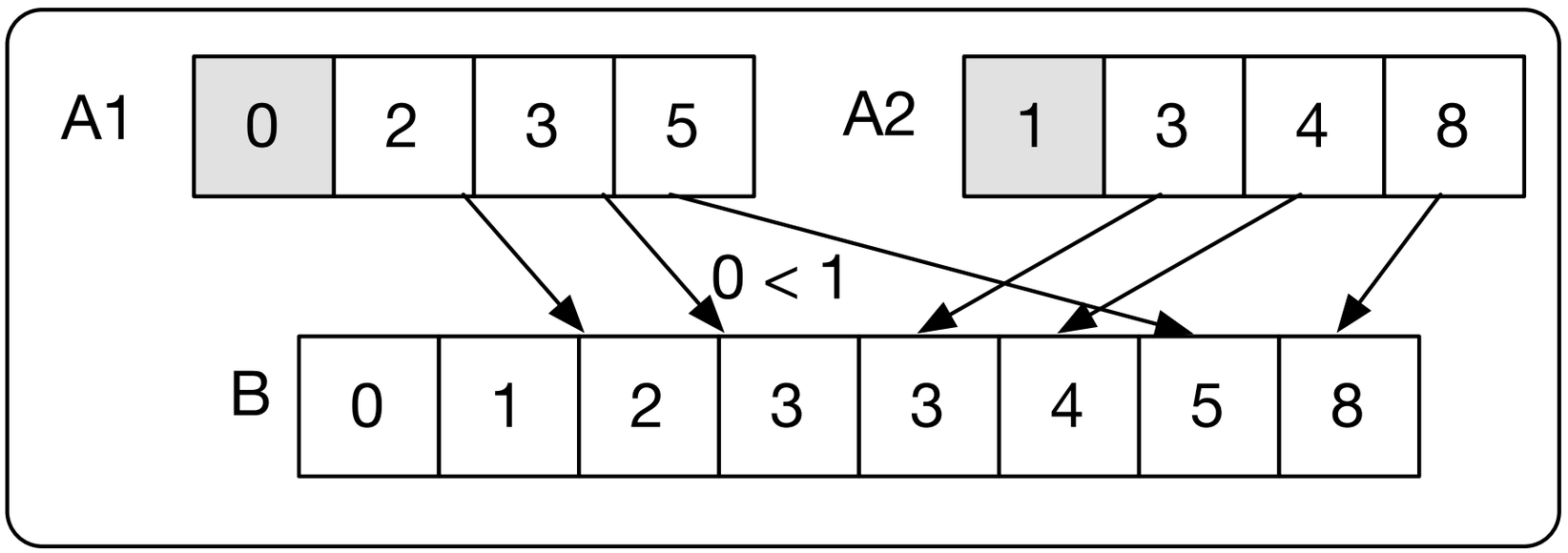


ソート列のマージ

A1とA2の先頭を比較し，小さい方をBの空欄の先頭に



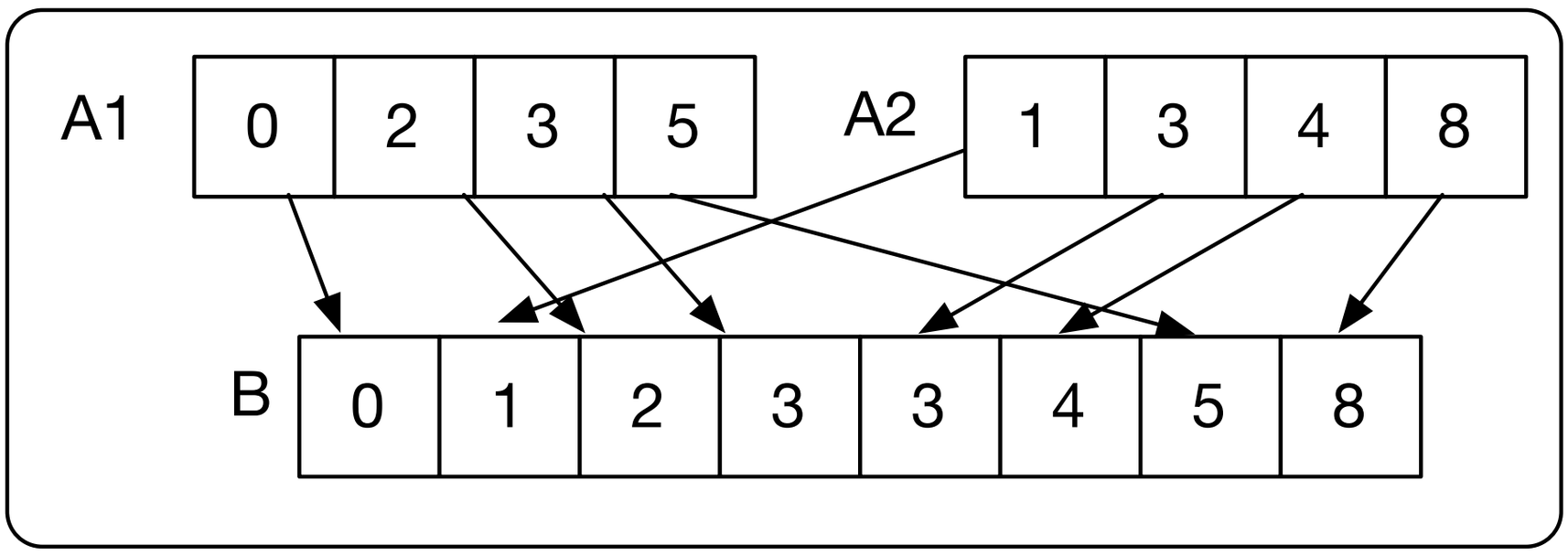
この作業を繰り返す



MERGE SORTの計算時間

- 計算時間
  - n1 = A1の要素数，n2 = A2の要素数
- A1とA2の各要素に対して
  - 他の要素との比較
  - Bに要素を書き込む

} 定数時間 c



c(n1+n2)時間

MERGE SORTの計算時間

- T(n)＝n個の要素のソート時間 (n は2のべき乗)
- 配列を２分割

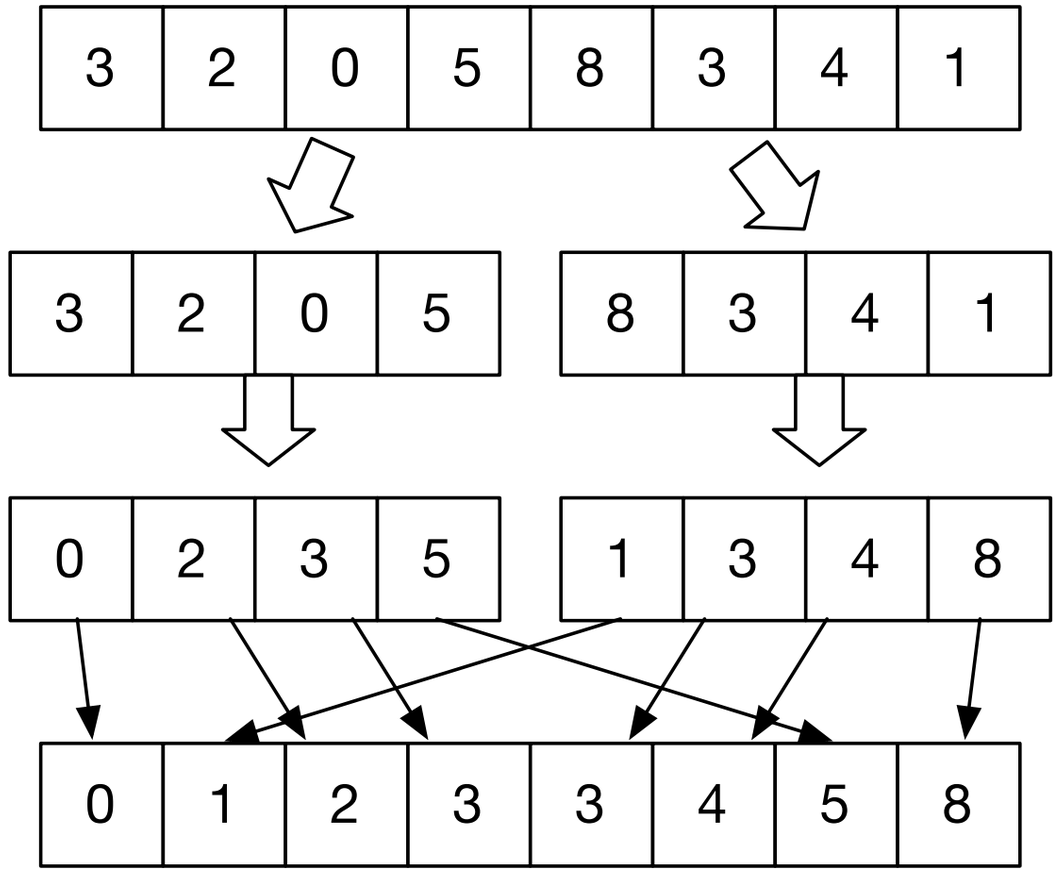
  c’ n 時間

- 2分割された配列を再帰的ソート

  T(n/2) x 2 時間

- ソートされた２つの配列をマージ

  c n 時間



MERGE SORTの計算時間

- T(n) = n個の要素のソート時間

- $T(n) = 2\,T(n/2) + (c+c')\,n$
- これを解くと

$$T(n) = (c+c')\,n \log n = O(n \log n)$$

QUICKSORT

- 内部ソート法としては最も効率の良い方法.
  - 平均：O(n log n)，ただし最悪$O(n^2)$
- 考え方：
  - 整列する範囲の中から基準値(pivot)を 1 つ決める
  - 基準値より小さいデータは左，大きいデータは右に集める (分割：partition)
  - 左部分と右部分のそれぞれにクイックソートを適用

QUICKSORT

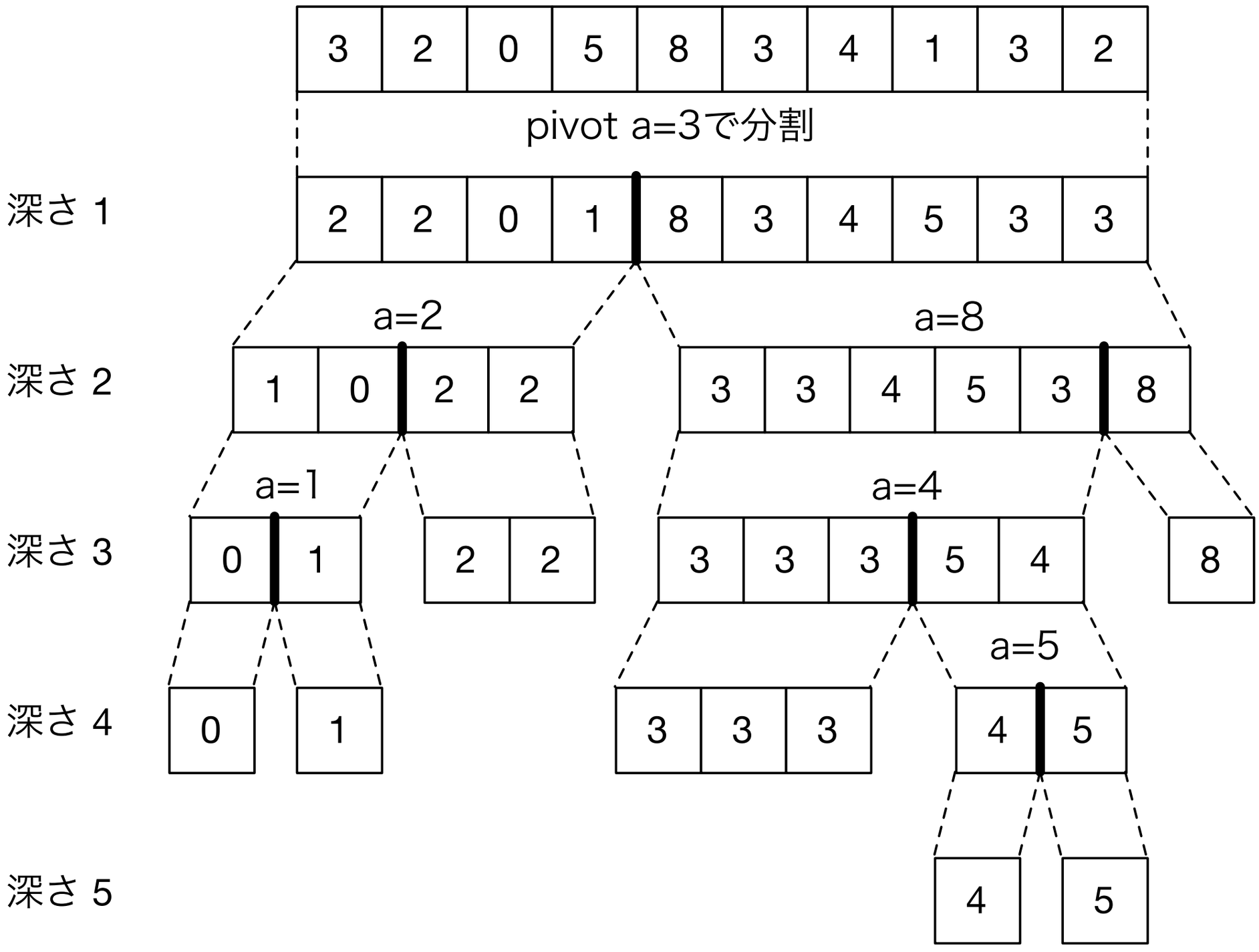
- 従来の整列法はそれぞれの要素が収まる位置を１つずつ決めていたが，quicksortは，位置決めは後回しにして，全体を２分する．

- quicksort，mergesortのように，問題をいくつかの部分に分解して解き，その結果を組み合わせて全体の解を得る方式を分割統治法(divide and conquer)と言う．

QUICKSORT: PIVOTの選び方

- 良い選び方
  - 配列をほぼ２等分する
  - pivotの選択に時間をかけない
- pivotの選び方
  - ランダムに１つ選ぶ
  - 左端，右端，真中の位置の３要素の中央値を選ぶ
  - 左から見て最初に得られた２つの異なる値の大きい方を選ぶ

QUICKSORTの動作



QUICKSORT (MAIN PROGRAM)

```
void quicksort(int i, int j, int *A) {
    int a, pv, k;

    pv = pivot(i, j, A);              /* iからjの範囲でpivotを選ぶ */
    if (pv != -1) {
        a = A[pv];                    /* pivotの位置の値 */
        k = partition(i, j, a, A); /* iからjを基準値 aで分割．kは分割位置 */
        quicksort(i, k-1, A);         /* iからk-1をソート */
        quicksort(k, j, A);           /* kからjをソート */
    }
    return;
}
```

QUICKSORT: PIVOTの選択

- A[i], …, A[j]からA[pv]を選び出力
- A[pv]はA[i]と最初に異なるA[k]と比べ大きい方．全て同じならpv=-1

```
int pivot(int i, int j, int *A) {
    int pv, k;

    k = i+1;
    while (k <= j && A[i] == A[k]) /* 異なる値を探す */
        k = k+1;
    if (k > j)                     /* 見つからない */
        pv = -1;
    else if (A[i] >= A[k])         /* A[i]の方が等しいか大きい */
        pv = i;
    else                           /* A[k]の方が大きい */
        pv = k;
    return(pv);
}
```

QUICKSORT: グループの分割(PARTITION)

- 前半はA[i], …, A[k-1] < a, 後半はA[k], …, A[j] ≧ a

```
int partition(int i, int j, int a, int *A) {
    int l, r, k;

    l = i; r = j;
    while (1) {
        while (A[l] < a) /* pivot aより大きい値を探す */
            l = l+1;
        while (A[r] >= a) /* pivot aより小さい値を探す */
            r = r-1;
        if (l <= r) {     /* A[l]とA[r]を交換 */
            swap(l, r, A);
            l = l+1;
            r = r-1;
        } else            /* l>rでループ終了 */
            break;
    }
    k = l;
    return(k);
}
```

```
void swap(int i, int j, int *A) {
    int tmp;

    tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
    return;
}
```

QUICKSORTの計算時間

- T(n) = n個の要素のソート時間とする
- 一般の場合：k個とn-k個に分割される (1<k<n)
  - $T(n) \leqq T(k) + T(n-k) + c\,n$
    - 帰納法により，解は

      $$T(n) \leqq 2c\,n^2 = O(n^2)$$

- 実際的には，数列がほぼ半分に分割される
  - 実用上の計算時間は$O(n \log n)$，平均時間も$O(n \log n)$

比較

- 実際的な計算速度はquicksort > merge sort ≒ heap sort
- 実際に整列を必要とする問題では，quicksortをまず考えるべき.
- heap sort, merge sortは平均も最悪もO(n log n)で，計算量の面で安全.
- merge sortの欠点は整列する配列と同じ作業領域を必要とすること．（主に２次記憶に蓄えられた大きなデータの整列）

参考：ALGORITHM ANIMATION

https://visualgo.net/en/sorting

https://www.toptal.com/developers/sorting-algorithms

ソートのまとめ

	best	average	worst
bubblesort	$O(n)$	$O(n^2)$	$O(n^2)$
insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$