アルゴリズムとデータ構造

- 第９回講義トピック：整列（続き）
  - クイックソート
  - マージソート
  - ２分木ソート
  - 各種整列法の比較

クイックソート (quick sort)

- 「分割統治」と「再帰」の考えに基づく
  - データ列中の適当な値を基準値として、それより小さいか等しい要素を左側、大きいか等しい要素を右側に来るようにデータセットを2つの部分列に分ける。
  - こうしてできた左部分列と右部分列に対しさらに再帰的に分割を行う。

基本アルゴリズム

```
quicksort(int a[], int l, int r) {
  int i;
    if (r > l) { // 終了条件
      i = partition(l, r);  //列の分割
      quicksort(a, l, i-1); //左の部分列のソート
      quicksort(a, i+1, r); //右の部分列のソート
   }
}
```

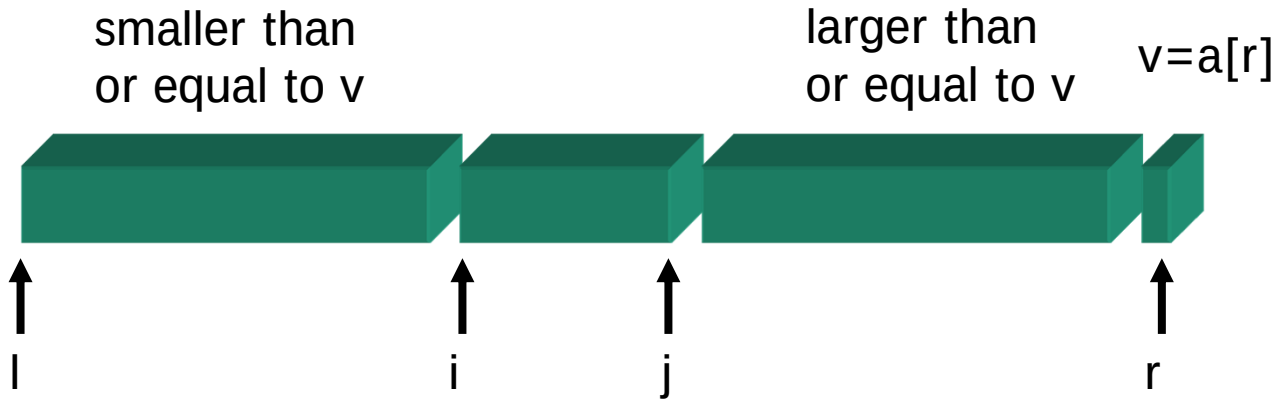
分割について

- データ列にある要素を１つ選び、分割値（ピボット、pivot）とする。
  - ピボットの選び方はいろいろあって、例えば中間要素を選ぶ方法もあるが、ここでは最後の要素a[r]を選ぶ方法を採用する。
- データ列を、分割値を境界とした前後２つの部分列に分割する。
- 分割値の最終位置をiとすると、
  - a[1],...,a[i-1]のすべての要素はa[i]より小さいか等しい。
  - a[i+1],...,a[r]のすべての要素はa[i]より大きいか等しい。

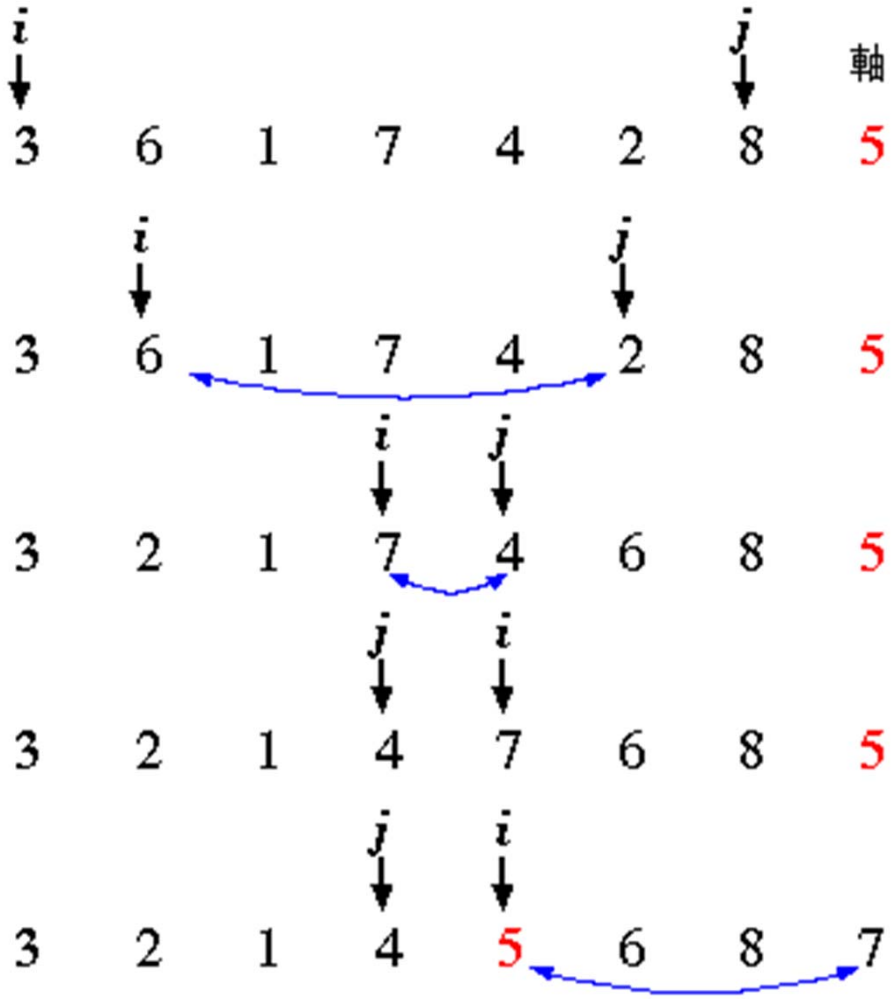
の条件を満たす。

実際の分割アルゴリズム

- 2つのポインタ(インデックス)i、jを用意し、iは左から右へ移動しながら分割値vより大きい値を見つけ、また、jは右から左へ移動しながらvより小さい値を見つける。iとjの要素を交換する。
- さらにiは右へ、jは左へ移動しながら交換する要素を見つける。
- iとjは最終的に左右交差するので、その場所の要素a[i]を分割値のa[r]と交換する。



クイックソートにおける分割の様子



クイックソートのプログラム例

```
quicksort(int a[], int l, int r) {
  int v, i, j, t;
  if (r > l) {
    v = a[r]; i = l-1; j = r;
    for (; ;) {
      while (a[++i] < v) ;
      while (a[--j] > v) ;
      if (i >= j) break;
      t=a[i]; a[i]=a[j]; a[j]=t;
    }
    t=a[i]; a[i]=a[r]; a[r]=t;
    quicksort(a, l, i-1);
    quicksort(a, i+1, r)
  }
}
```

このアルゴリズムは挿入ソートと同様にa[0]に番兵（sentinel, 配列の最小値よりも小さい値）を置く必要がある。
もし、配列の最小値が決められない場合はwhile文の中でindexのチェックをする必要があり、アルゴリズムが若干遅くなる。

クイックソートの例

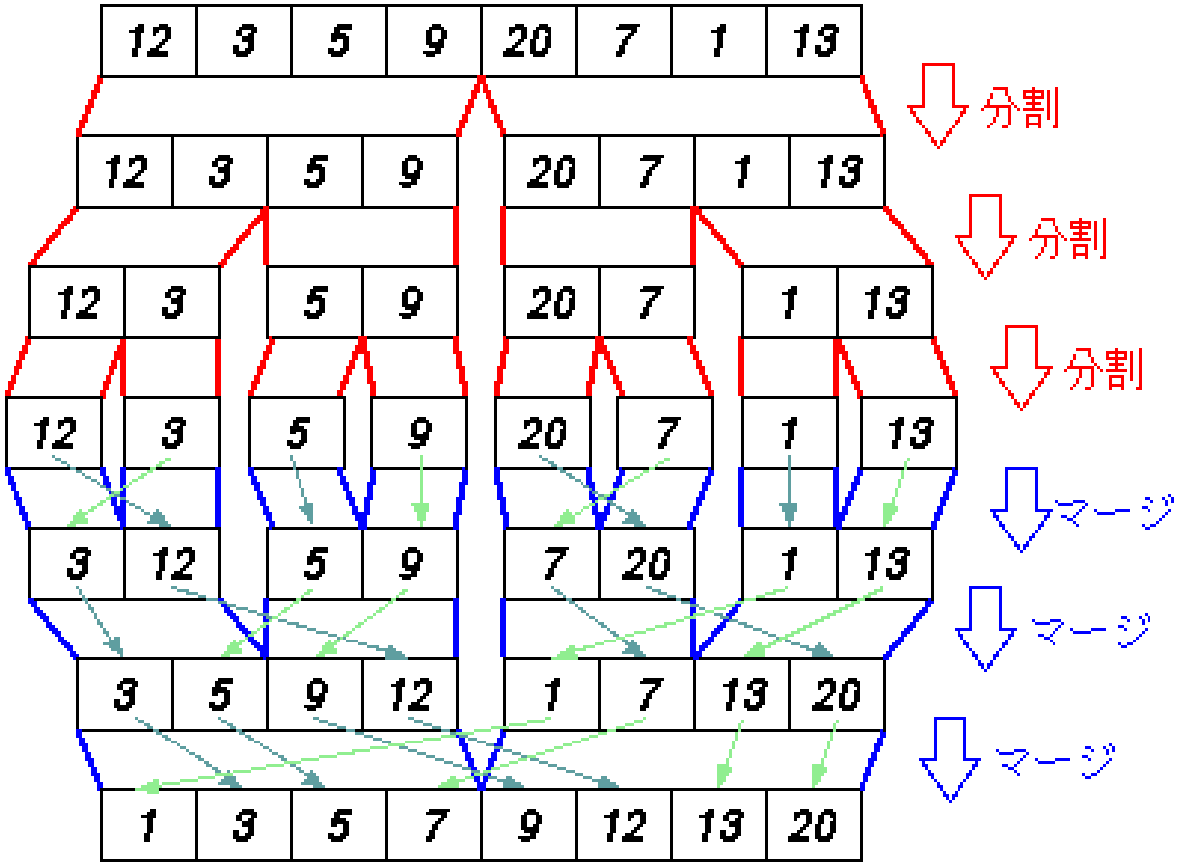
- 以下の文字列がクイックソートによって整列される様子

  ＡＳＯＲＴＩＮＧＥＸＡＭＰＬＥ

マージソート (merge sort)

- 基本的考え方:
  - クイックソートが要素を選択して2つの部分列に分割するのに対して、マージソートは2つの部分列に分割し、それぞれを整列してから併合して1つの整列したデータ列を作る考え方である。
    この2つの方法は選択と併合という相補的な考え方であるといえる。
  - 2つの整列した部分列から1つの統合したデータ列を作り出す方法を2ウェイ併合(two-way merge)という。
- 方法
  - 再帰的に配列を2つの部分配列に分割し、それぞれを整列する。
    ここでいう整列とは再帰的に自分自身を呼び出すことで、最終的には要素1つの配列となり、自然的に整列していく。
  - 整列された2つの部分列を併合し、1つの整列した配列を作る。

マージソートの例


12 3 5 9 20 7 1 13
分割
12 3 5 9 20 7 1 13
分割
12 3 5 9 20 7 1 13
分割
12 3 5 9 20 7 1 13
マージ
3 12 5 9 7 20 1 13
マージ
3 5 9 12 1 7 13 20
マージ
1 3 5 7 9 12 13 20

マージソートのプログラム

```
mergesort(int a[], int l, int r) {
   int i, j, k, m;
   if (r > l) {
     m = (r+l)/2;
     mergesort(a, l, m);
     mergesort(a, m+1, r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m; j < r; j++) b[r+m-j] = a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
   }
}
```

＊このアルゴリズムのマージ部分は2つの部分列を背中合わせにすることによって、それぞれの最大要素と最小要素が番兵の役割を果たす。

非再帰マージソート

- ボトムアップのアプローチ
  - まず、隣り合う2つの要素をマージして2要素の部分列を作る。
  - 次に、隣り合う2つの2要素列をマージして4つの要素の部分列を作る。
  - さらに次に8つの要素の部分列を作る。
  - 以上の繰り返しで、最終的にデータ列全体を整列する。

マージソートのプログラム（非再帰）

```
struct node *mergesort(struct node *c) {
  int i, N; struct node *a, *b, *head; *todo, *t;
  head = (struct node *) malloc(sizeof *head);
  head->next = c; a =z;
  for (N = 1; a != head->next; N = N+N) {
    todo = head->next; c = head;
    while (todo != z) { t = todo; a = t;
      for (i = 1; i < N; i++)  t = t->next;
      b = t->next; t->next = z; t = b;
      for (i = 1; i < N; i++)  t = t->next;
      todo = t->next; t->next = z; c->next = merge(a,
  b);
      for (i = 1; i <= N + N; i++)  c = c->next;
} } return head->next;  }
```

マージソートの例

- マージソートによって以下の文字列が整列される様子

  Ａ Ｓ Ｏ Ｒ Ｔ Ｉ Ｎ Ｇ Ｅ Ｘ Ａ Ｍ Ｐ Ｌ Ｅ

2分木ソート (binary tree sort)

- データから2分木を作成する
  1. まず、最初の要素を木のルートとする。
  2. 次の要素はルートからリーフまで、親より小さい場合は左に、大きいか等しい場合は右に要素の挿入位置を決める。
  3. 要素を木の子節点として新たに作成する。
  4. 手順2、3をすべての要素に対して繰り返す
- 中央順走査による整列

  2分木は中央順走査することで、キーの小さい順に節を走査するので、走査した節のデータを順番に出力すれば、整列した結果を得ることができる。

２分木ソートの例

- ２分木ソートによって以下の文字列が整列される様子

  ＡＳＥＡＲＣＨＩＮＧＥＸＡＭＰＬＥ

その他の整列法

- ラディックスソート（基数整列、radix sort）
  - 数の離散的な性質を利用した整列法
  - 特徴：キー全体を比較しないで、キーの一部分を参照したり、比較したりする。
- ヒープソート (Heapsort)
  - 選択ソートの改良版
  - 能率のよい整列法
- 外部整列

各種の整列法の特徴

- 選択ソート
  - 数列から最小（最大）要素を探すことを繰り返す。
  - 比較回数は多いが交換回数が少ない。
- 挿入ソート
  - 整列された部分数列に対し該当要素を適切な位置に挿入することを繰り返す。
- バブルソート
  - 隣接する2つの要素を逐次交換する。
  - 交換回数が多い。
- シェルソート
  - 数列をとびとびのいくつかの部分数列にわけ、それぞれを挿入整列法でソートする。

各種の整列法の特徴（つづき）

- クイックソート
  - ある要素を軸に、全体を選別して2つの部分数列に分割し、それぞれの数列を整列する。
- マージソート
  - 全体の数列を2つの部分数列に分割し、それぞれを整列してから、併合して、1つの整列した数列を作る。
- 2分木ソート
  - データから2分木を作成し、中央順操作を行う。

各種整列法の性能

- 選択ソート
  - 約$N^2/2$回の比較、N回の交換が必要になる。
- 挿入ソート
  - 平均の場合、約$N^2/4$回の比較、$N^2/8$回の交換が必要になる。
  - 最悪の場合はその2倍の比較と交換が必要になる。
- バブルソート
  - 約$N^2/2$回の比較、$N^2/2$回の交換が必要になる。
- シェルソート
  - 平均で約$N^{1.25}$回の比較が必要になる。

各種整列法の性能（つづき）

- クイックソート
  - 平均で約2NlnN回の比較が必要になる。入力データの並びによっては最大$O(N^2)$に退化する可能性がある。
- マージソート
  - 平均約NlnN回の比較が必要になる。
- ２分木ソート
  - 平均約2NlnN回の操作が必要になる。
    挿入時の探査に所要する計算量がメーンとなる。