データ構造とアルゴリズム
（第9回）

モバイルコンピューティング研究室
柴田史久

1

本日の講義内容

- 整列（２）
  - シェルソート
  - クイックソート
    - 分割統治法

2

教科書 第13章（pp.305～314）

シェルソート

3

シェルソート(1)

- シェルソート：Shell Sort
  - Donald L. Shell が考案（1959年）
  - クイックソートの考案までは最速
  - 実用性能：$O(n^{1.25})$ ～ $O(n^{1.5})$
  - 安定ではない
- 挿入ソートの改良版
  - 挿入ソートはコピー回数が多すぎる
  - 例えば最も右側に最小の要素があったらどうなる？
    - 正しい場所に収めるために，それより大きいすべての項目を右へシフトする必要がある（＝ほぼN回のコピー）
  - 小さな項目を一挙に左に移動できないか？
    - 前処理として，離れている要素同士を比較・交換しておこう

4

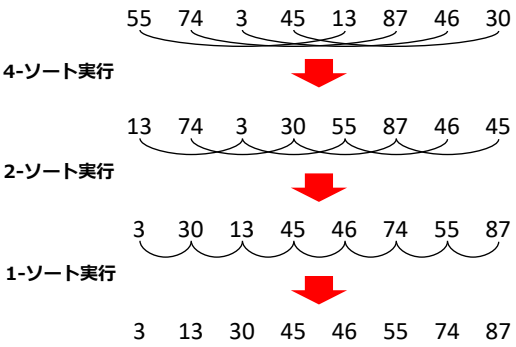
シェルソート(2)

- 歩幅 h の挿入ソートを複数回実行
- 歩幅 h を徐々に小さく

- 手順
  - 歩幅 h で挿入ソートを実行
  - 歩幅を小さくして繰り返す
  - 歩幅 h が 1 の状態での挿入ソートを実行すれば終了

5

シェルソート(3)



6

増分の選択(1)

- h-ソートについての性質
  - x>yならばx-ソート後にy-ソートすると，両方のソート済みの性質を満たす
- シェルソート：減少する数列に従ってh-ソートを行う手順
  - 数列に現れる値は互いに倍数になっていないほうがよい => 要素を混ぜ合わせながらh-ソートを実行するため

7

7

増分の選択(2)

- インターバル数列（Knuthによる解析と実験の結果）
  - 1, 4, 13, 40, 121, …
  - Knuth による数列の定義式
    - h = 3 * h + 1 → h = (h – 1) / 3
  - 計算量： $O(n^{1.25})$
- 初期値を見つけるには
  - for (h = 1; h < n / 9 ; h = h * 3 + 1)
  - 見つけた後は，歩幅 h による挿入ソートを実行し，h を順次小さくしていく

8

8

増分の選択(3) 補足

- Shellのオリジナル案
  - 最初のギャップを N/2 として，それを２で割っていく
  - 性能が悪い場合があることが判明（ $O(n^2)$ に退化する）
- 改良案
  - 2.2で割る方法
  - N=100ならば，45, 20, 9, 4, 1
  - 数列の最後を必ず 1 にするための工夫が必要
- Flamingの案

```
if (h < 5)
  h = 1 ;
else
  h = (5 * h – 1) / 11 ;
```

9

9

シェルソートの実装

詳細は教科書 p.313 List 13.1

```
public static void sort(int[] a)
{
  int n = a.length;   // 配列の要素数
  int h;

  for (h = 1; h < n / 9; h = h * 3 + 1)  // hの初期化
    ;

  for (; h > 0 ; h /= 3) {
    for (int i = h; i < n; i++) {
      int j = i;
      while (j >= h && a[j – h] > a[j]) {  // h離れた同士を比べて
        int temp = a[j];                   // 適切な位置まで
        a[j]     = a[j – h];               // 交換していく
        a[j – h] = temp;
        j -= h;                            // j = j - h
      }
    }
  }
}
```

10

10

教科書 第14章（pp.315～331）

クイックソート

11

11

クイックソート（1）

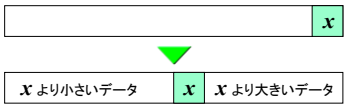
- クイックソート：Quick Sort
- Charles A. R. Hoare が考案（1962年）
- 平均計算量： $O(n \log n)$

12

12

クイックソート（2）

- 手順
  - 基準値 x を選ぶ（基準値は枢軸（pivot）と呼ぶ）
  - 基準値 x より小さいデータを左側，残りを右側に分割
  - 分割された２つの部分に同様の操作を，分割部分の要素数が１になるまで繰り返す
- 再帰アルゴリズムで実装
- 分割統治法（divide and conquer）



13

13

分割統治法

- divide and conquer
- divide and rule
- 大きな問題を複数の小さな問題に分割し，各個撃破する方法
- クイックソートでは
  - 枢軸で分割した部分配列をそれぞれ別々に整列

14

14

クイックソート（3）

- アルゴリズム概略（疑似コード）

詳細は教科書 p.317 List 14.1

```
static void quickSort(int[] a, int l, int r) {

  if (整列する要素が1つのみである)
    return ;

  適当な要素 a[v] を枢軸として,
    a[v]より小さい要素をa[l]  ～a[v-1]に集め,
    a[v]より大きい要素をa[v+1]～a[r]  に集める

    quickSort(a, l, v - 1) ; // 左の部分を整列
    quickSort(a, v + 1, r) ; // 右の部分を整列
  }
}
```

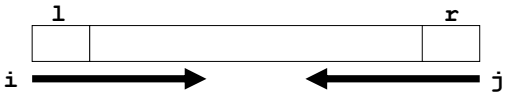
- 再帰呼び出しで左右の整列を実現

15

15

分割のアルゴリズム(1)

- 枢軸（pivot）によってデータを２つの部分に分割
- 基本的な手順
  - 左から右へ走査しpivotより大きい要素を探索：i
  - 右から左へ走査しpivotより小さい要素を探索：j
  - 両者の位置関係が i < j ならば交換
  - i >= j となるまで（＝ぶつかるまで）繰り返す

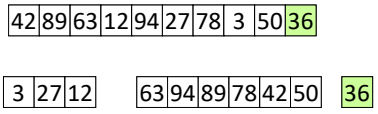


16

16

pivotの選び方(1)

- pivotは任意の値ではなくこれからソートするデータ中に存在する値とする
- pivotとして例えば分割する部分の右端のデータを選ぶ
- 分割後に，pivotとして選んだデータを左と右の部分配列の境界に挿入する
  → 最終的なソート済みの位置に収まる

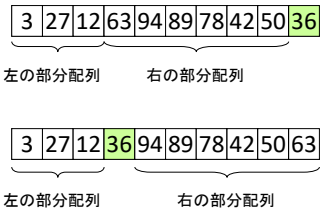


17

17

pivotの選び方(2)

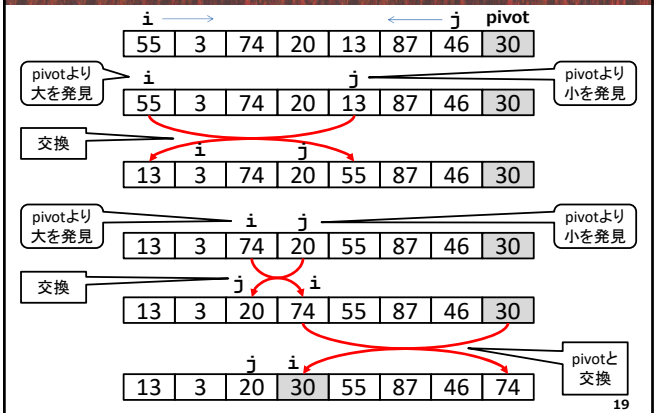
- どうやってpivotを適切な位置にはめ込む？
  - 右にシフトするのは可能だが無駄
  - 右側の部分配列の左端とpivotを入れ替えればよい



18

18

分割の流れ

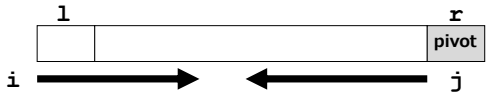


19

19

分割のアルゴリズム(2)

- 枢軸（pivot）によってデータを２つの部分に分割
- 配列の右端をpivotにする
- 手順
  1. pivotより大きい要素が見つかるまでポインタ i を右へ
  2. pivotより小さい要素が見つかるまでポインタ j を左へ
  3. ポインタ i と j の指す要素を交換
  4. 1〜3 をポインタ i と j がぶつかるまで繰り返す
  5. ポインタ i が指している要素とpivotを交換



20

20

分割の実装

詳細は教科書 p.321〜322 List 14.2

```
private static int partition(int[] a, int l, int r) {
  int i = l - 1 ;      // i を初期化
  int j = r ;          // j を初期化
  int pivot = a[r] ;   // 枢軸を右端の要素に設定
  while (true) {       // i と j がぶつかるまで繰り返す
    while (a[++i] < pivot)  // 右端に pivot があるので必ず脱出できる
      ;  // NOP 何もしない. ただ探しているだけ
    while (i < --j && pivot < a[j])  // いつかは i にぶつかる
      ;  // NOP 何もしない. ただ探しているだけ
    if (i >= j)
      break ;  // ぶつかったら終わり. ループを抜ける
    int temp = a[i] ; // ぶつかっていないので i と j の指す要素を交換
    a[i]     = a[j] ;
    a[j]     = temp ;
  }
  int temp = a[i] ; // a[i] と枢軸を交換
  a[i] = a[r] ;
  a[r] = temp
  return i
}
```

21

21

クイックソートの実装

詳細は教科書 p.321〜322 List 14.2

```
private static void quickSort(int[] a, int l, int r) {
  if (l >= r)  // 整列する要素が1つなら何もしないで戻る
    return ;

  int v = partition(a, l, r) ;  // 枢軸a[v]を基準に分割

  quickSort(a, l, v - 1) ;      // 左部分配列a[l]〜a[v-1]を整列

  quickSort(a, v + 1, r) ;      // 右部分配列a[v+1]〜a[r]を整列
}

public static void sort(int[] a) {

  quickSort(a, 0, a.length - 1) ;  // quickSortを呼び出して整列

}
```

22

22

クイックソートの計算量

- 平均計算量： $O(n \log n)$
- 最悪計算量： $O(n^2)$
  - 一方のグループに $n-1$ 個のデータが偏る場合
  - 右端をpivotとした場合，逆順ソートされたデータは最悪
- 最悪ケースの場合，再帰呼び出しに必要なスタック領域の大きさが $O(n)$
- 平均ではスタック領域は $O(\log n)$

23

23

クイックソートの改善(1)

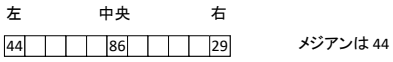
- 再帰を利用しない実装
  - List 14.3，List 14.4
- 整列する範囲をスタックによって管理
  - List 14.3
- 左右の部分配列のうち，短いほうを優先して処理
  - List 14.4
- 詳細説明は割愛

24

24

クイックソートの改善(2)

- 理想的にはpivotはメジアン（中央値）であるべき
  - メジアンを求めるのに時間がかかるので事実上不可能
- 3のメジアン（median-of-three）を使う
  - 最初，最後，中央の3項のメジアン



- 選んだメジアンと右端の要素を入れ替えるコードを追加する

25

25

その他の工夫

- クイックソートは定数係数が大きい
- 部分配列がある程度短くなったら挿入ソートを利用
- 短さの程度は実際のデータで実験すべし

26

26

まとめ

- シェルソート
- クイックソート
  - 分割統治法

27

27

参考文献

- 定本 Javaプログラマのための
  アルゴリズムとデータ構造（近藤嘉雪）
- 新・明解 Javaで学ぶ
  アルゴリズムとデータ構造（柴田望洋）
- 岩波講座ソフトウェア科学 3
  アルゴリズムとデータ構造（石畑清）
- Javaで学ぶアルゴリズムとデータ構造
  Robert Lafore（著）・岩谷 宏（翻訳）
- Java アルゴリズム+データ構造完全制覇
  オングス (著)・杉山 貴章・後藤 大地 (監修)

28

28