



プログラミング演習 I 問3 (第3回)

担当 藤井 雅弘

概要

- 第1回 単純な整列アルゴリズム(1)
 - C言語の復習(配列, マクロ定義, 代入演算子)
 - バブルソート
- 第2回 単純な整列アルゴリズム(2)
 - C言語の復習(3項演算子)
 - 選択ソートと挿入ソート
- 第3回 高速な整列アルゴリズム
 - CSVファイルの利用
 - クイックソート
 - 課題



CSVファイルの利用

CSVファイルとは

- コンマ区切り (**Comma Separated Values**) 形式のテキストファイル
- データをコンマ(,)で区切り、改行を用いて2次元的に配列
- コンマを含むデータは二重引用符(")でくくられる

- 標準出力例

```
printf("%f, %f, %f\n", a, b, sum);
```

- 標準入力例

```
scanf("%lf, %lf, %lf", &a, &b, &sum);
```

CSVファイルへの出力

```
/* csv_ex1.c*/
#include <stdio.h>
int main(void)
{
    double a, b, sum;
    a = 2.345;
    b = 5.678;
    sum = a + b;
    printf("%f, %f, %f\n", a, b, sum);
    return 0;
}
```

コンマがなければCSVではない

fooディレクトリにいることを確認

Z:¥foo>notepad csv_ex1.c notepadで新規作成

Z:¥foo>cl csv_ex1.c コンパイル

Z:¥foo>csv_ex1 > bar.csv 実行（結果をbar.csvに出力）

ソースファイルの新規作成と編集

- **notepad** (メモ帳) 新規作成の時
 - Z:¥foo> notepad cvs_ex1.c
 - 保存先が Z:¥foo であることを確認
- **copy** (内部コマンド) コピーする時
 - Z:¥foo>copy cvs_ex1.c cvs_ex2.c
- **devenv** (開発環境) 編集する時
 - Z:¥foo>cvs_ex1.c

CSVファイルからの入力(その1)

```
/* csv_ex2.c */
#include <stdio.h>
int main(void)
{
    double a, b, sum;
    scanf("%lf, %lf, %lf", &a, &b, &sum);
    printf("a=%f, b=%f, sum=%f\n", a, b, sum);
    return 0;
}
```

コンマをとったら？

Z:¥foo>copy csv_ex1.c csv_ex2.c コピー

Z:¥foo>csv_ex2.c 編集

Z:¥foo>cl csv_ex2.c コンパイル

Z:¥foo>csv_ex2 < bar.csv 実行(bar.csvからデータ入力)

CSVファイルからの入力(その2)

```
/*csv_ex3.c */
#include <stdio.h>
int main(void)
{
    double a, b, sum;
    char s[135];
    fgets(s, sizeof(s), stdin);
    sscanf(s, "%lf, %lf, %lf", &a, &b, &sum);
    printf("a=%f, b=%f, sum=%f\n", a, b, sum);
    return 0;
}
```

- **fgets**: 一行入
- **sscanf**: 文字列sからscanf

Z:¥foo>notepad csv_ex3.c **新規作成**

Z:¥foo>cl csv_ex3.c **コンパイル**

Z:¥foo>csv_ex3 < bar.csv **実行(bar.csvからデータ入力)**

CSVファイルへの追加出力

```
/* csv_ex4.c */  
#include <stdio.h>  
int main(void)  
{  
    double a, b, sum;  
    a = 1.234; b = 2.567;  
    sum = a + b;  
    printf("%f, %f, %f\n", a, b, sum);  
    return 0;  
}
```

Z:\foo>notepad csv_ex4.c 新規作成

Z:\foo>cl csv_ex4.c コンパイル

Z:\foo>csv_ex4 >> bar.csv 実行(bar.csvにデータ追加出力)

標準ファイル入出力(その1)

```
/* csv_ex5.c */
#include <stdio.h>
int main(void)
{
    double x, y, z;
    int i = 0;
    char s[135];
    while (fgets(s, sizeof(s), stdin) != NULL) {
        sscanf(s, "%lf, %lf, %lf", &x, &y, &z);
        printf("%f, %f, %f\n", x, y, z);
        i++;
    }
    printf("ファイル中のデータ(x, y, z) の数は%d 組です. \n", i);
    return 0;
}
```

読んですぐ使い捨てるので配列は不要

fgets関数の値がNULLでない(入力がCtrl-zでない)間は{}を繰り返す

Z:¥foo>notepad csv_ex5.c 新規作成

Z:¥foo>cl csv_ex5.c コンパイル

Z:¥foo>csv_ex5 < bar.csv 実行(bar.csvからデータ入力)

標準ファイル入出力(その2)

```
/* csv_ex6.c */
#include <stdio.h>
int main(void)
{
    double x, y, z, sum;
    char s[135];
    while (fgets(s, sizeof(s), stdin) != NULL) {
        sscanf(s, "%lf, %lf, %lf", &x, &y, &z);
        sum = x + y + z;
        printf("%f + %f + %f = %f\n", x, y, z, sum);
    }
    return 0;
}
```

Z:¥foo>notepad csv_ex6.c 新規作成

Z:¥foo>cl csv_ex6.c コンパイル

Z:¥foo>csv_ex6 < bar.csv > hoge.csv

実行(bar.csvからデータ入力し,
hoge.csvにデータ出力)

クイックソート

クイックソートのアルゴリズム

- 内部整列アルゴリズムの中では最高速
- 考え方
 - 適当な要素 x を配列の要素の中から1つ選ぶ
 - この x を枢軸(pivot)と呼ぶ
 - それを最終的におかれるべき位置 $a[v]$ に移動
 - x より小さい要素を配列の前半へ, x より大きい要素を配列の後半へ振り分ける
 1. $a[v]$ よりも小さい要素 $a[0] \dots a[v-1]$
 2. $a[v]$
 3. $a[v]$ よりも大きい要素 $a[v+1] \dots a[n-1]$
- 1, 3の部分配列を個別に整列して a 全体の整列完了

分割統治

クイックソートアルゴリズム

```
/*配列aのうち配列a[l]からa[r]を整列する*/
```

```
void quick_sort_core(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pivotにして,
```

```
        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
```

```
    quick_sort_core(a, l, v-1); /*前半部分を整列する*/
```

```
    quick_sort_core(a, v+1, r); /*後半部分を整列する*/
```

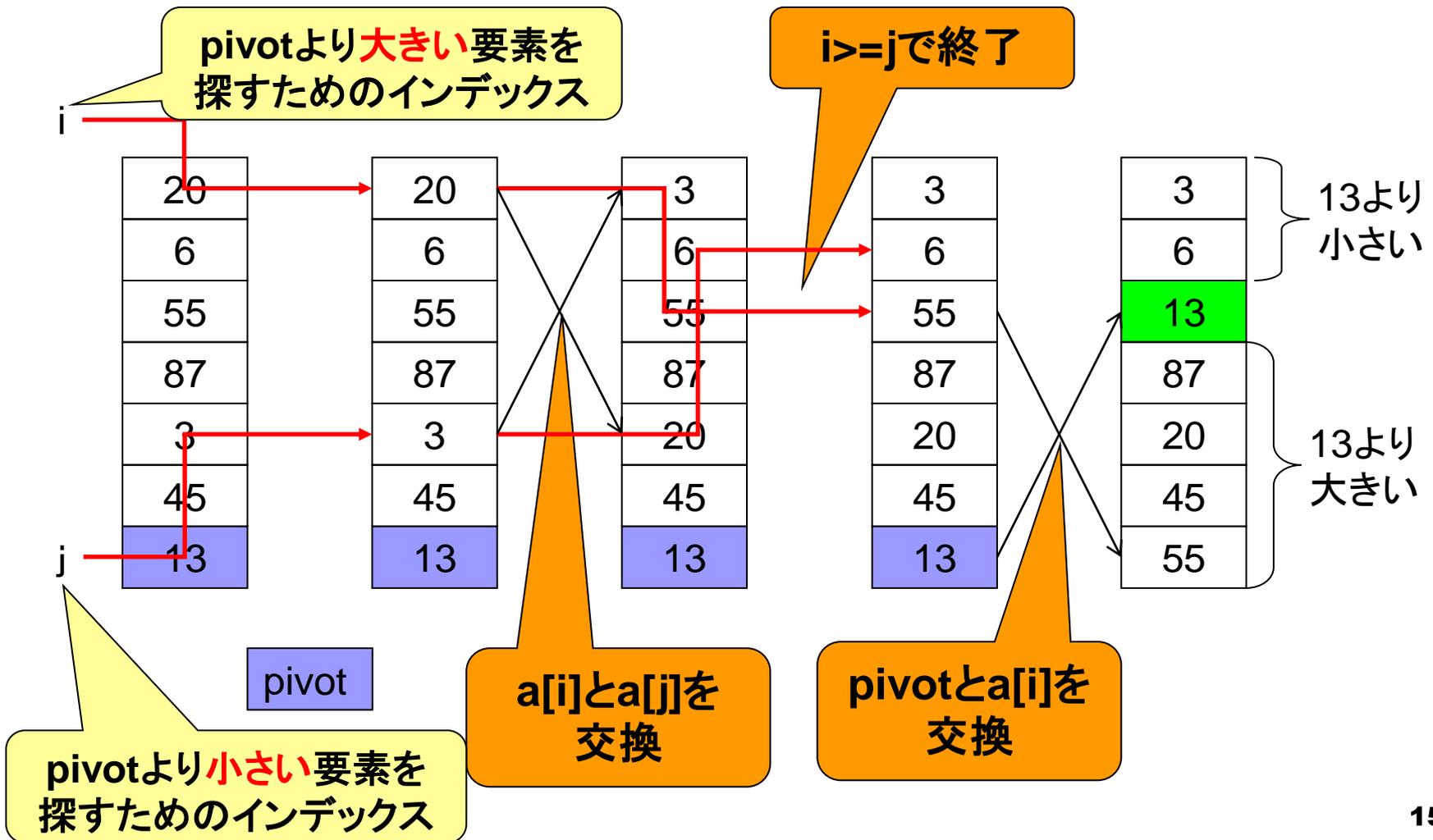
```
}
```

```
void quick_sort(int a[], int n) {
```

```
    quick_sort_core(a, 0, n-1);
```

```
}
```

分割のアルゴリズム



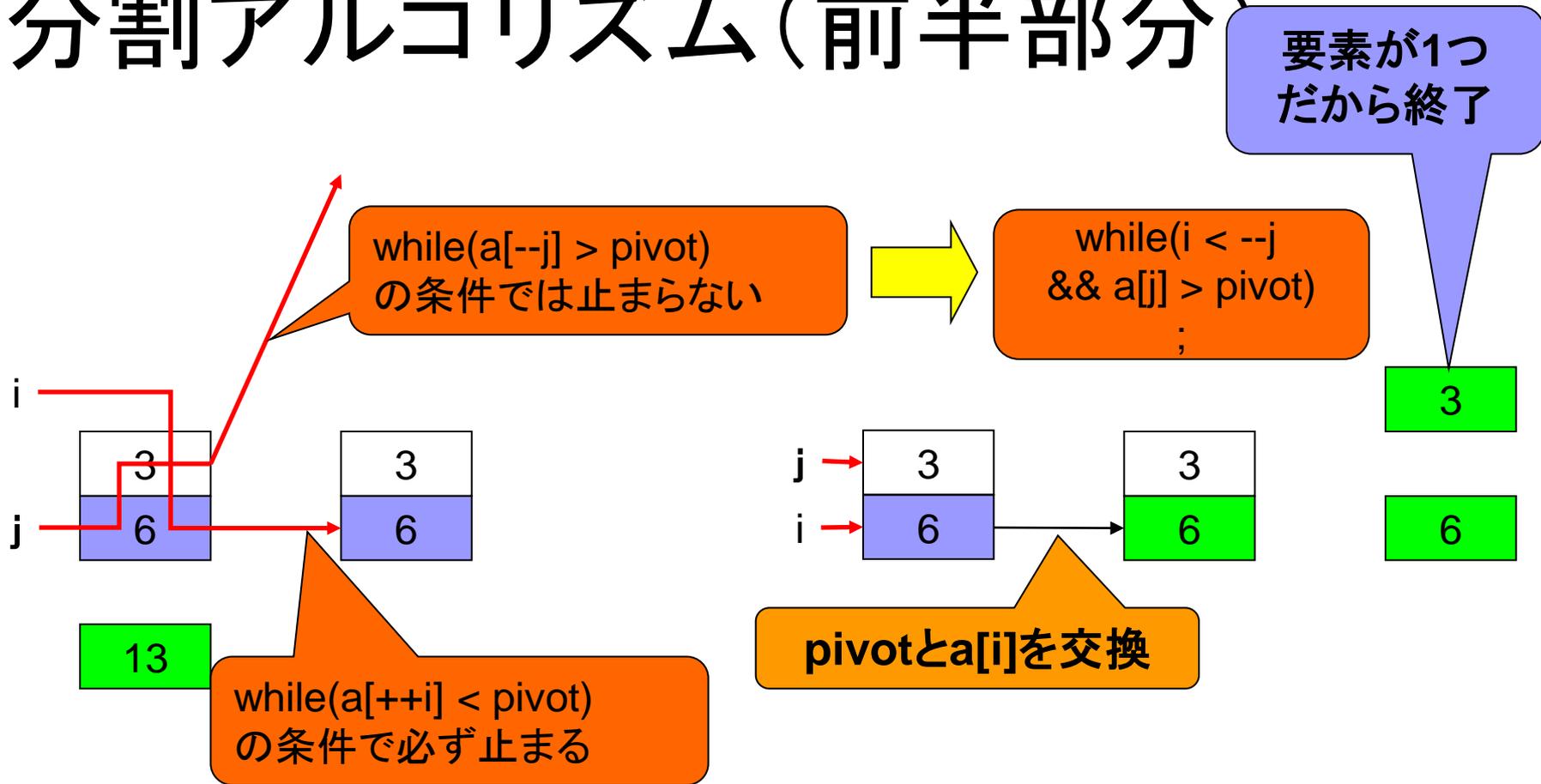
分割アルゴリズム

1. pivotより**大きな**要素が見つかるまでiを進める
2. pivotより**小さい**要素が見つかるまでjを戻す
3. iとjの指す要素を交換する
4. 1から3までをiとjが**ぶつかるまで**繰り返す.
5. 最後にiが指す要素(必ずpivotより大きいとか等しい)をpivotと交換する

分割アルゴリズムの実装

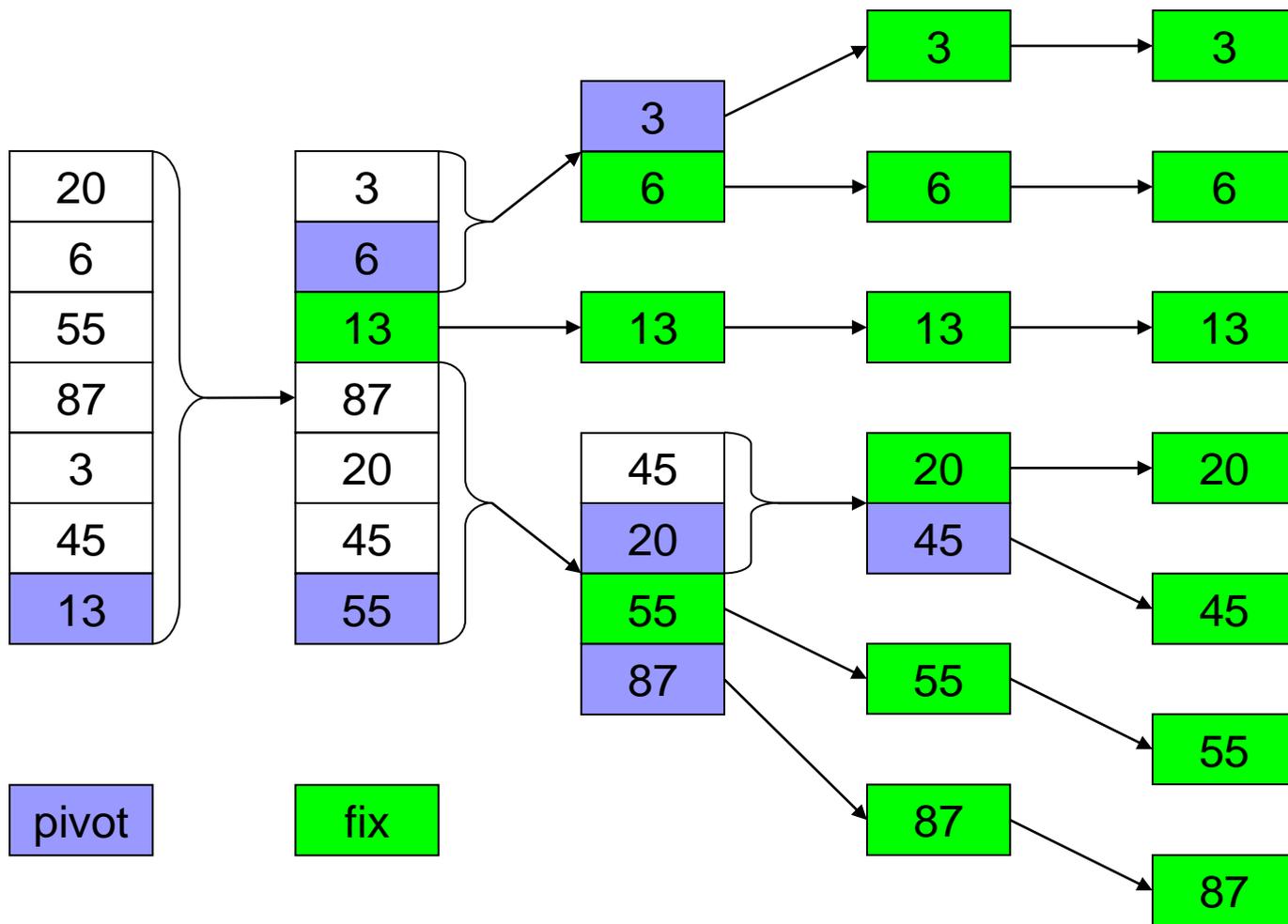
```
/*部分配列aのうち配列a[l]からa[r]を分割する*/
i = l-1; /*部分配列の先頭の要素番号-1*/
j = r; /*部分配列の最後の要素番号*/
pivot = a[r]; /*部分配列aの最後の要素をpivotにする*/
for(;;){ /*無限ループ*/
    /*iを進めてpivotより大きい要素番号を見つける*/
    while(a[++i] < pivot)
        ;
    /*jを戻してpivotより小さい要素番号を見つける*/
    while(a[--j] > pivot)
        ;
    /*iとjがぶつかったら無限ループをぬける*/
    if(i >=j)
        break;
    /*iの指す要素とjの指す要素を交換する*/
    t = a[i]; a[i] = a[j]; a[j] = t;
}
/*a[i]とpivot(a[r])を交換する*/
t = a[i]; a[i] = a[r]; a[r] = t;
```

分割アルゴリズム(前半部分)



戻る

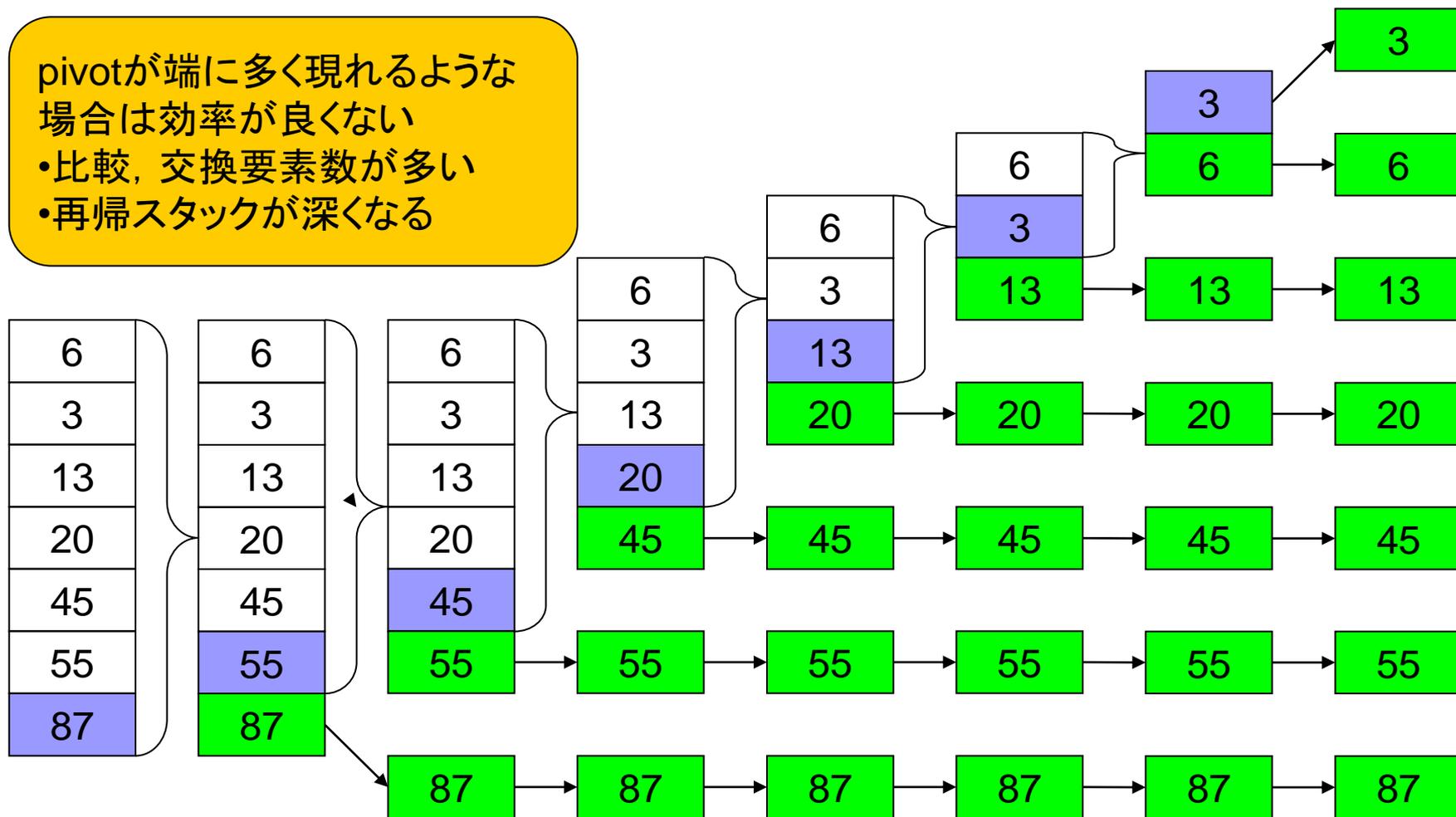
クイックソートの動き



クイックソートの問題点

pivotが端に多く現れるような場合は効率が良くない

- 比較, 交換要素数が多い
- 再帰スタックが深くなる



練習問題(1)

- クイックソートを行う関数void quick_sort_core(int a[], int l, int r)を実装せよ