クイックソート

計算量とは

- アルゴリズムのよしあしを評価する基準
  - 領域計算量(space complexity)
    - 計算機のメモリーをどれだけ必要とするか
  - 時間計算量(time complexity)
    - アルゴリズムが答を出すまでにどの程度の計算時間を必要とするか
    - プログラム中のそれぞれの文が実行される回数を数えて計算量の目安をつける

計算量の評価


```
1:  found = false;
2:  for  ( i=0 ; i<n ; i++ ){
3:     if  ( x == table[i] ){
4:        found = true;
5:        break;
      }
   }
```

線形探索アルゴリズム

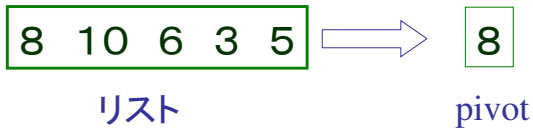
ループを回る回数がk回のとき、つまりx=table[k-1]のときの計算時間（概算）は
$T(k)=t_1+k(t_2+t_3)+t_4+t_5$
ループを回る回数の平均は約n/2回
最悪の場合(k=n)の計算時間は
$T= t_1+t_4+t_5+n(t_2+t_3)$
nが大きくなれば定数$t_1+t_4+t_5$は無視でき、Tはおよそnに比例する
$T=cn$ $(c=t_2+t_3)$

➡ O記法では$O(n)$とあらわす

クイックソートの手順

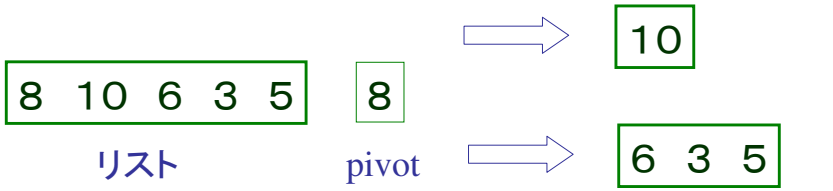
1. pivot の選択
   - ソートする範囲の中から pivot を1つ選ぶ
2. pivot による要素の分割
   - pivot を使い「より大きい要素」と「より小さい要素」に分割
   - 要素を1つずつ調べて、基準値より小さい要素と，より大きい要素を分ける
3. 分割された2つの部分に，再びクイックソートを適用
   - 「基準値より大きい要素」と「基準値より小さい要素」のそれぞれを独立にソート
   - 最後に、2つの部分と pivot をつなげる．全体がソートできたことになる

pivot の選択

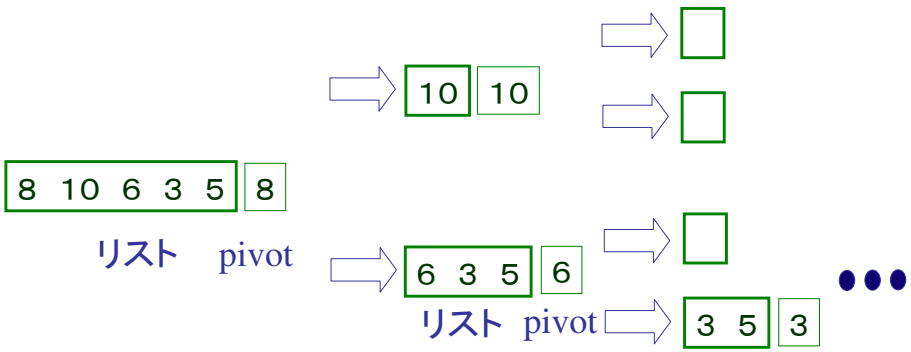


ソートする範囲の中から pivot を選ぶ
（ここでは，リストの先頭要素を pivot として
選んでいる）

pivot による要素の分割



要素を1つずつ調べて、基準値より
小さい要素と，より大きい要素を分ける



リストが空になるまで，pivot の選択と，
pivot による要素の分割を続ける

部分問題の例

- 休暇旅行の旅程（家から旅先のホテルまで）
  - 家→空港
  - 空港→旅先の空港
  - 旅先の空港→ホテル

クイックソートの部分問題

1. 「基準値より大きい要素」のソート
2. 「基準値より小さい要素」のソート

分割統治法(divide and conquer)

- サイズNの問題を解くのに、サイズが約N/2の部分問題2つに分けて、それぞれを再帰的に解き、その後でその2つの解を合わせて目的の解を得る

クイックソート

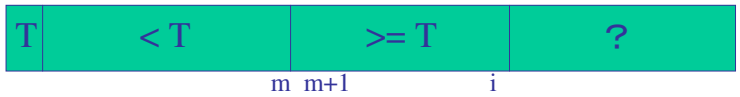
- 分割統治法に分類されるソート法
- 平均して O(n*log* n) の計算量でソートを行なうアルゴリズム
  - ヒープソートの計算量も O(n*log* n) だが、ヒープソートに比べて基本操作が簡単なので、実際の計算量はクイックソートの方が有利と言われる
  - 多くの問題に対して、最高速のソートアルゴリズムであると言われているそうですが、
- 最悪の場合には，$O(n^2)$ の計算量を要することが知られている

クイックソートの手順

- ソートする範囲の中から適当な値を1つ選ぶ
  - この値を基準値と呼ぶ
- 次に配列中の要素を1つずつ調べて、基準値より小さいデータを配列の左側、大きなデータを右側に集める
  - この操作を分割と呼ぶ
- 分割が終わったら、基準値より小さい部分と大きい部分に，それぞれに再びクイックソートを適用
  - これは再帰呼び出しで実現
- 分割された2つの部分を独立にソートすれば全体もソートされたことになる
  - 基準値より小さい部分に含まれる値は基準値より大きい部分に含まれるどの値よりも小さいから

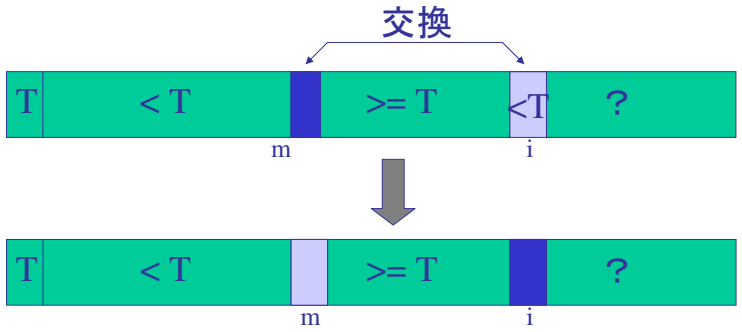
基本的な分割法

- ソートすべき範囲を left〜right とする
- まず基準値Tを選び、この値を left に移しておく
  - left の値は今まで基準値があった場所に移す
- 次に、left+1〜right の範囲を分割する



<u>ある時点 i まで分割が終わった状態</u>
left+1〜m の範囲に T 未満の値、
m+1〜i までの範囲に T 以上の値が残る

- i を1増やした後の i 番目の要素は，次の2通り
  1) i 番目の値が T以上：
     何もしない
  2) i 番目の値が T より小さい
     m を 1 増やし、小さい要素のための新たな場所を指すようにして、次にそこにある要素と i 番目の要素を交換する



- これを i が right にいたるまで繰り返す

クイックソートの平均計算量

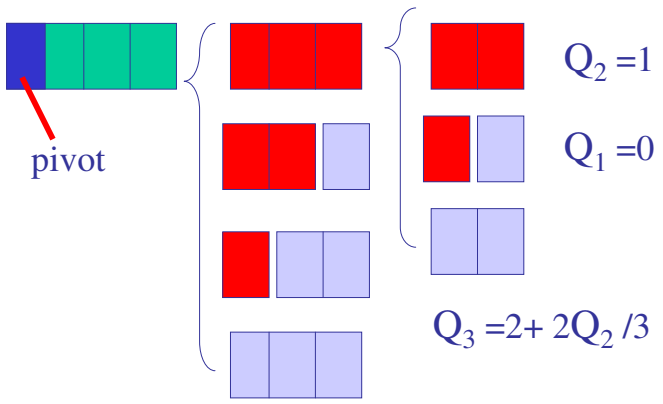
- ソートする範囲の要素数をnとする
- 1回の分割での比較の回数： n-1
- 長さnのクイックソートに要する計算量：

  $Q_n = n-1+ Q_a+Q_b$
  - ここで a, b は分割によって生じた左右の部分の長さ
  - a+b = n-1 が成立
  - a と b の組は分割の回数によって決まる
  - 平均をとると， $Q_n = n-1+ 2/n(Q_0+\ldots+Q_{n-1})$
  - これを計算すると

    $Q_n = 2(n+1)(1+1/2+\ldots+1/(n+1) - 2) +2$

    $Q_n \fallingdotseq 2n\log n$

n=4のときの計算量



$Q_4 = 3+(2Q_3 +2Q_2)/4$

最悪の場合

- 最悪の場合とは、分割した結果、一方の組に n-1 個の要素が残り、もう一方の組が空になる場合
- 基準値として最大または最小の値をとった場合に起こる
- 分割を行うたびにこのような状態になったとすると、比較の回数は

$$Q_n=(n-1)+(n-2)+\ldots+1 = (n(n-1))/2$$

基準値の選び方

- 「最悪の場合」から分かるように、基準値の選び方が，性能の鍵
- 最悪の計算量になる場合とは逆に，分割させる範囲をなるべく同じ長さの２つの部分に分けるような値を選ぶ
- しかし、分割のたびに中央値を求めるのでは、そのための手間がかかりすぎる
- 実用的な方法は，ソートする範囲の中からいくつかの値をサンプルとして選び、それらの中央値を基準とする方法です。
  - サンプルとしてとる値の個数は多いほど分割を均等化するためにはよい
  - 実際には3個とれば十分だとされている

簡単なソート法の併用

- ソートすべき範囲が短い場合には、クイックソートよりも単純な方法(挿入法など)と差が無い
- 高級なアルゴリズムは、単純な方法より計算量のオーダーが低い
  - 対象とするデータが多い場合には明らかに速い
  - データが少ない場合にはオーダーの問題とはならないことがありえる
  - １０～２０の範囲内であれば，どれを選んでも大差はないといわれている

例題　クイックソート

- 次の２つのクイックソートプログラムを作成し，性能を比較する（平均の比較回数は6 / 7程度に減り、実際の実行時間は5%ほど減ると言われている）
  - 基準値を適当な値にとるプログラム
  - ソートする範囲から3つのサンプルを選び，その中央値を基準値として使うプログラム
- ソート対象のデータは rand関数等を用いて生成すること

```
#include<stdio.h>
FILE *infile,*outfile;
int data[10000];
main()
{
    int i, n, in[20], out[20];
    printf("Input InFilename :"); /*入力データのファイル名を入力*/
    scanf("%s", in);
    if ((infile=fopen(in,"r"))==NULL) {
        printf("can't open file %s¥n", in);
        exit();
    }
    printf("Input OutFilename :");
    scanf("%s", out);
    if ((outfile=fopen(out,"w"))==NULL) {
        printf("can't open file %s¥n",out);
        exit();
    }
    n=0;
    while(fscanf(infile, "%d", &(data[n])) != EOF) {
        n++;
    }
    Quick(0,n-1);           /*0からn-1の範囲でクイックソート*/
    for (i=0; i<n; i++) {
        fprintf(outfile,"%d¥n",data[i]);
    }
    fclose(outfile);
    fclose(infile);
}
```

```
Quick(int left,int right)
{           /*クイックソートを行う*/
    int i,t,m;
    t=Choice(left, right);          /*基準値を選ぶ*/
    swap(&data[left], &data[t]);
    m=left;
    for (i=left; i<right; i++){
        if (data[i+1]<data[left]) {    /*基準値よりも小さい場合*/
            swap(&data[i+1], &data[m+1]);
            m++;
        }
    }
    swap(&data[left], &data[m]);       /*基準値をmに挿入*/
    if (m-left>1) {
        Quick(left, m-1);
    }
    if (right-m>1) {
        Quick(m+1, right);
    }
}
```

```
int Choice(int left,int right)
{
 /*基準値を選ぶ*/
    if ((right-left)<2) {
        return(left);    /*要素数が3つないとき*/
    }
    else {              /*要素数が3つ以上のとき*/
        if((data[left]<=data[left+1] & data[left+1]<=data[left+2])
        | (data[left+2]<=data[left+1] & data[left+1]<=data[left+2])) {
            return(left+1);       /*中央値を返す*/
        }
        else if((data[left+1]<=data[left] & data[left]<=data[left+2])
        | (data[left+2]<=data[left] & data[left]<=data[left+1])) {
            return(left);
        }
        else {
            return(left+2);
        }
    }
}
```

```
swap(int *a, int *b){
   /*値の交換*/
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
```