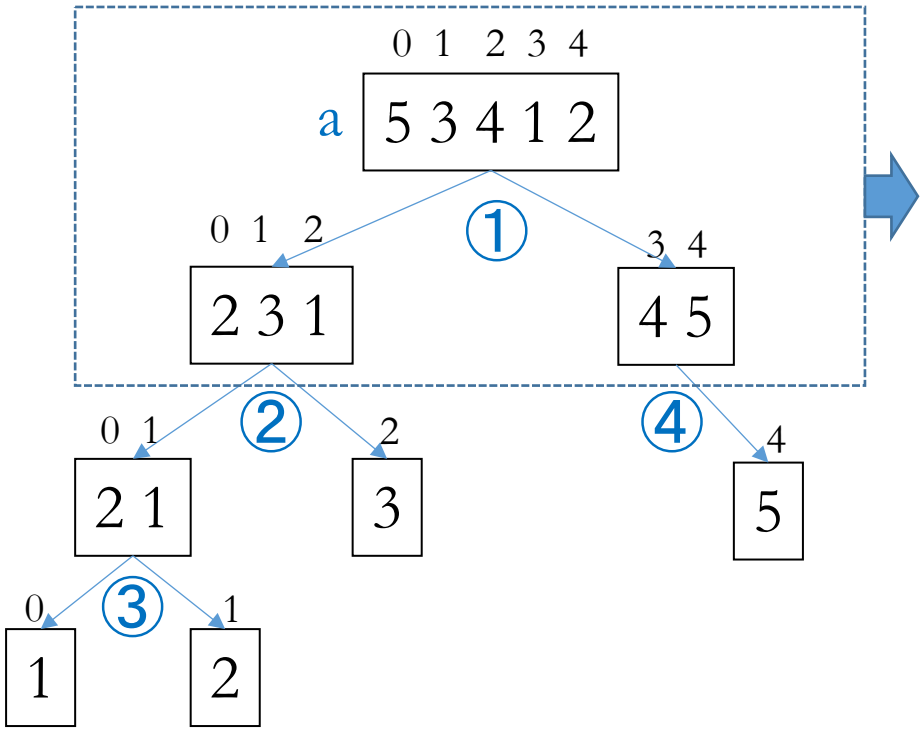
# クイックソート

# 考え方

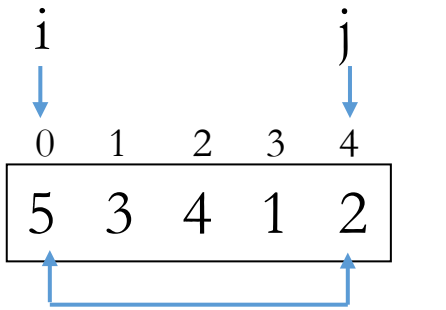
- ある基準値をもとにして、配列の要素を1つ1つ調べ、それより小さい値のデータを左側にし、大きい値のデータを右側にして、分割する。分割ができれば、その結果の左右の区間に同じ処理を行う。以上の処理を繰り返すことにより、整列を行う

# データ分割の例

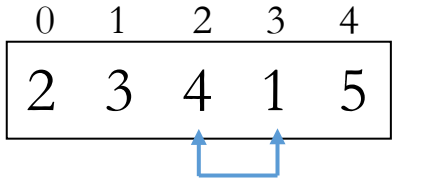



①の分割：f=0, t=4

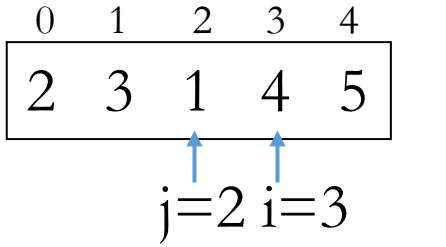
基準値：4




i=0 交換 j=4




i=2 交換 j=3

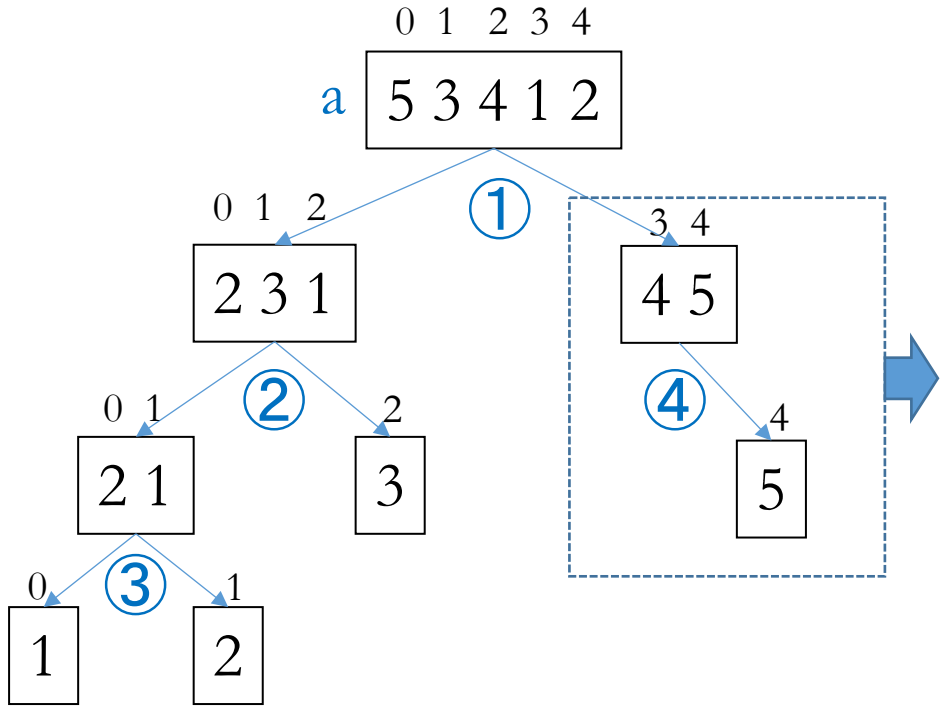



i>jのため分割完了

f～j：0～2の(2, 3, 1)が左②

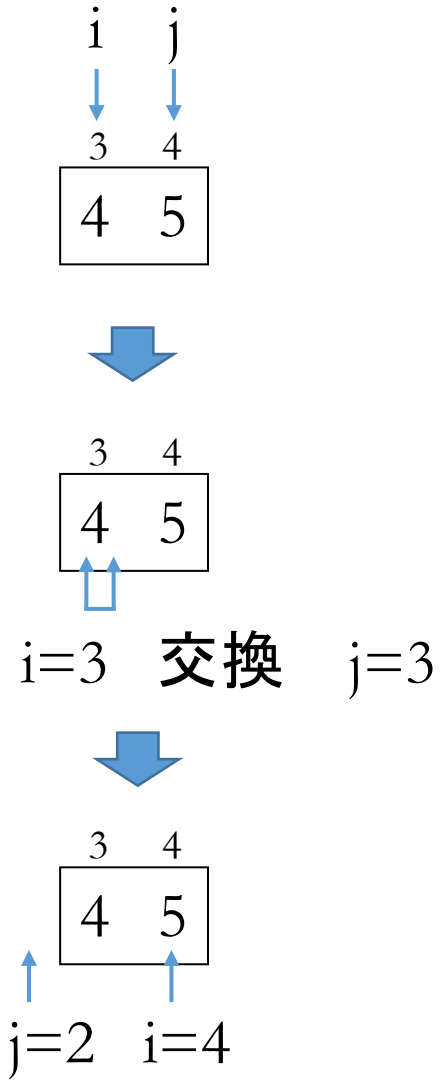
i～t: 3～4の(4, 5)が右④

# データ分割の例




④の分割：f=3, t=4

基準値：4




f～j:3～2はあり得ないから左はなし
i～t:4～4の(5)が右

i>jのため分割完了

# データ分割の方法

配列a[f], a[f+1], …, a[t]

1. 基準値pを決める
   - 一番簡単でよく使われるのは、配列中央のデータを基準値とする。つまり、p=a[(f+t)/2]
2. 基準値より大きい値を右に、小さい値を左に配置し、配列を分割する
   - i=f, j=tとする
   - i<=jが成立する間に、以下を繰り返す
     I. a[i]<pが成立する間にi++を行う （都合の悪いa[i]>=pのiを探し出す。この場所で止まる）
     II. a[j]>pが成立する間にj--を行う （都合の悪いa[j]<=pのjを探し出す。この場所で止まる）
     III. i<=jならa[i]とa[j]を交換し、i++, j--を行う

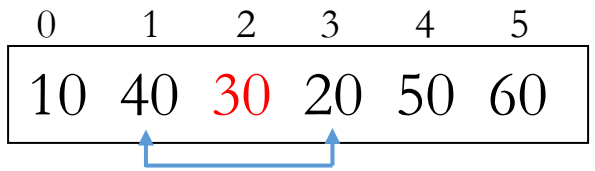
結果：配列aが左側f～jのa[f],…,a[j], 右側i～tのa[i], …, a[t]に、分割される

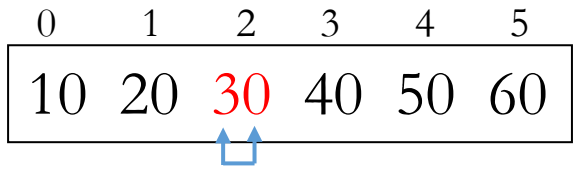
# Question

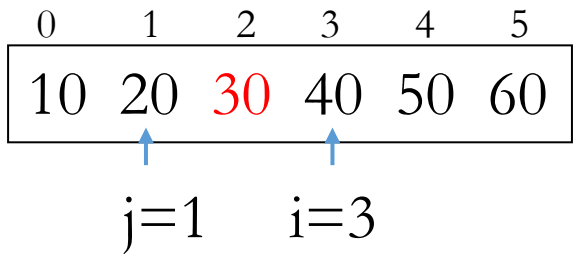
- (20, 60, 40, 50, 30, 10)の分割結果を求めなさい

# 補足：i-j>1がありうる

- 分割結果i-j>1の例は前にもあった（p3）
- 以下もi-j>1の例




f～j：0～1の(10, 20)が左
i～t: 3～5の（40, 50, 60）が右

30というデータは処理対象から除外される

# データ分割のアルゴリズム

入力：配列a[f], … , a[t]

出力：分割結果　（左）a[f], … , a[j], （右）a[i], … , a[t]

f: 部分配列の左端の要素番号

t: 部分配列の右端の要素番号

補助変数：

i：配列を左(f)から右へみていくときの位置

j：配列を右(t)から左へみていくときの位置

p：基準値

# データ分割のアルゴリズム

1. {中央にあるデータを基準値pとする}p=?
2. {i, jの値を初期化する} i=?, j=?
3. {配列a[f],...,a[t]をa[f],...,a[j]とa[i],...,a[t]に分割する}
   i<=jが成立する間に、以下を繰り返す
   ?

# クイックソートの手順

- 基準値を決め、基準値以下の値のデータを左側へ、基準値以上の値のデータを右側に移し、データを左右に分割する
- 左に分割されたデータに対し、このクイックソートの手順を実行する
- 右に分割されたデータに対し、このクイックソートの手順を実行する

**再帰！**

# 再帰とは

- 上記の、「クイックソートの手順」の中に「クイックソートの手順」を呼び出す、というような、自分自身を呼び出す処理を再帰処理という
- プログラム的に書くと、以下のようなもの

```
void proc(){
        ⋮
    proc();
        ⋮
}
```

# 簡単なプログラム例1

- 1+2+...+nをもとめ結果を返す関数を書くとすれば、for文を用いて以下のように簡単に作れる

```
int sum(int n)
{
  int i, s=0;
  for(i=1; i<=n; i++)
     s += i;
  return s;
}
```

# Question

- 繰り返し処理をfor, while文を使わずにどう実現する？

- 再帰を使えばよい

# 簡単なプログラム例1

再帰プログラム

```
int sum(int i)
{
  if(i==1) return i;
  return i+sum(i-1);
}
```

級数f(n)=1+2+...+nは再帰的に
f(n)=n+f(n-1), f(1)=1…漸化式
と書ける

再帰展開(n=3の場合…sum(3)を呼び出す場合)…処理過程

```
sum(3)
{
    3+sum(2)を返す
        {
            2+sum(1)を返す
                {
                    1を返す
                }
        }
}
```

# Question

- プログラムの実行結果を示しなさい。

```
#include <stdio.h>
int sum(int i)
{
  int ai, r;

  if(i==1){
    printf("%d¥n", i);
    return 1;
  }
  ai=2*i-1;
  printf("%d+", ai);
  r=ai+sum(i-1);
  return r;
}

int main(void)
{
   int n=4;
   printf("sum: %d¥n", sum(n));
   return 0;
}
```

# 簡単なプログラム例2

- int型変数iに値1を100回足してその結果を返す関数を書くとすれば、for文を用いて以下のように簡単に作れる

```
int cal(int i)
{
   int j;
   for(j=0; j<100; j++)
      i++;
   return i;
}
```

# 簡単なプログラム例2

```
int j=0; // グローバール変数

int cal(int i)
{
   i++; j++;
   if(j<100)
     i=cal(i);
   return i;
 }
```

```
int cal(int i)
{
   static int j=0;
   i++; j++;
   if(j<100)
     i=cal(i);
   return i;
}
```

再帰プログラムを書くときの注意点：
必ず条件を正しくつけること！！
そうでないと、プログラムが無限ループになってしまう

# 再帰過程展開

```
int j=0;

int cal(int i)
{
  i++; j++;
  if(j<3)
    i=cal(i);
  return i;
}

int main(void)
{
  int i;
  i=cal(0);
  return 0;
}
```

展開

```
j=0
i=cal(0)
{
  i=1, j=1
  j<3なので、
    i=cal(1)
    {
      i=2, j=2
      j<3なので
        i=cal(2)
        {
          i=3, j=3
          j<3成立しないので、calを呼び出さない
          return 3→i=3
        }
      return 3→i=3
    }
  return 3→i=3
}
```

# Question

- プログラムの実行結果を示しなさい。

```
int j=0;

int cal(int i)
{
  i++; j++;
  if(j<100)
    i=cal(i);
  return i;
 }
```

```
int main(void)
{
  int i;
  i=cal(50);
  printf("i:%d¥n",i);
  return 0;
}
```

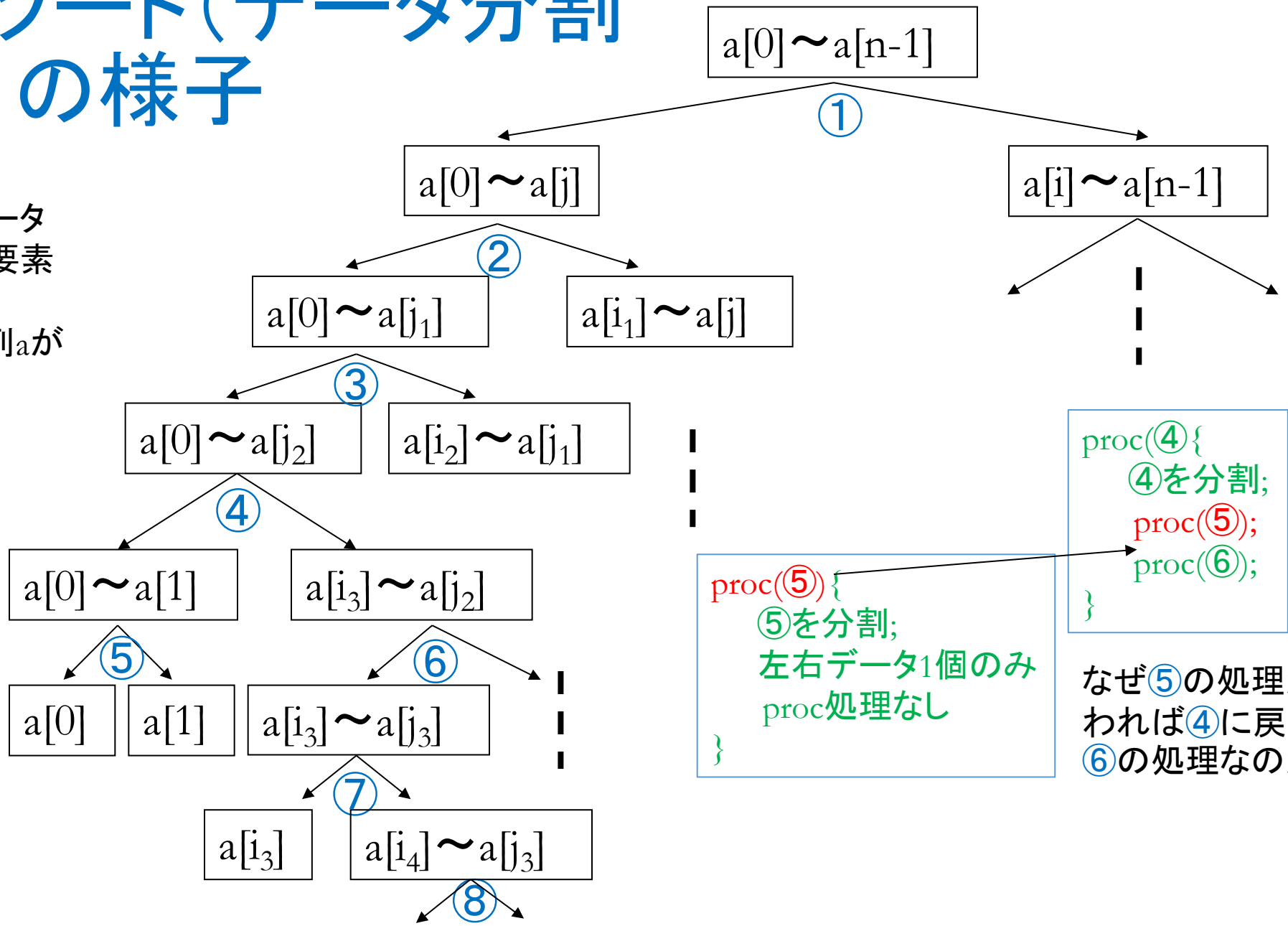
参考

# （C言語）変数の記憶クラスについて

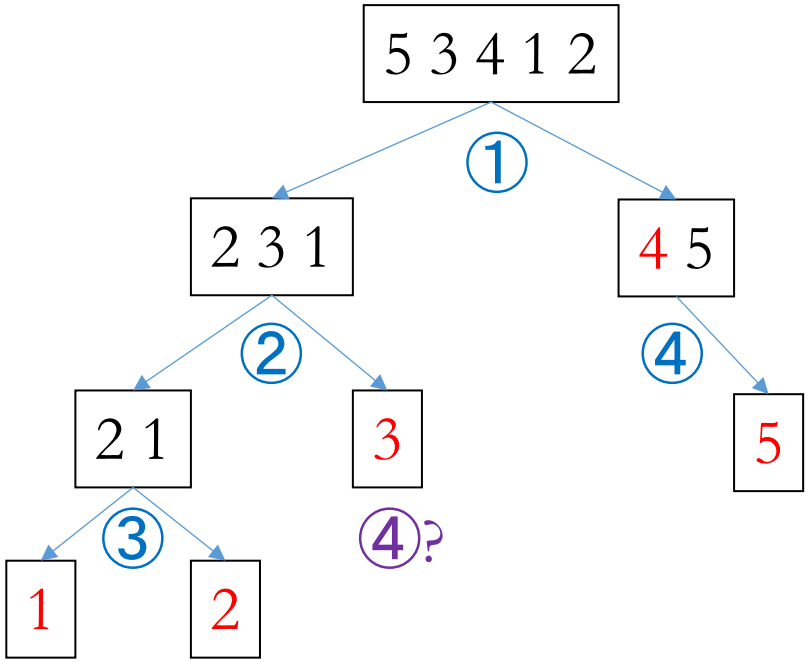
| 記憶クラスの種類 | 記憶場所 | 意味 | 宣言の例 |
|---|---|---|---|
| auto<br>通常の変数 | スタック<br>（メモリ上の一時的な記憶領域） | 宣言された実行単位内のみ有効でその実行単位を抜けた後は、その値は保存されない | auto int a;<br>（通常、種類の記述は省略される） |
| static<br>静的変数 | 通常のメモリ | 宣言された実行単位内のみで有効だが、その値は実行単位を抜けた後も保存される | static int a; |

# クイックソート（データ分割の再帰）の様子

各層において
1. 左のデータ≤ 右のデータ
2. 左の要素番号<右の要素番号（たとえばj<i）
3. 処理が終わったら配列aが昇順になっている



# クイックソート（データ分割の再帰）の様子（実例）




③の処理で1, 2の結果が得られた後、②に戻って、右の矢印に沿って④？を処理しようとするが、データが1個しかなく、処理済みとみなす。これで②の処理が終わって①に戻ってその右の矢印に沿って次は④の処理を行う

# クイックソートのポイント（ここまでのまとめ）

1. データ分割
   - 「左データの値 ≤ 右データの値」を満たすように
2. データ分割の再帰
   - 選択ソート・バブルソートでは、処理対象データの決定は左・右からデータを確定しながら、データの範囲を縮小していくループで実現する
   - クイックソートでは、処理対象データを決めるのは、データの分割であり、データ分割の繰り返しは通常のループ（for文）で実現するのが困難。再帰が必要
     - for文が何重必要かが決まらない。for文の範囲も決まらない
   - 再帰の代わりに、スタックというデータ構造を利用する方法もある

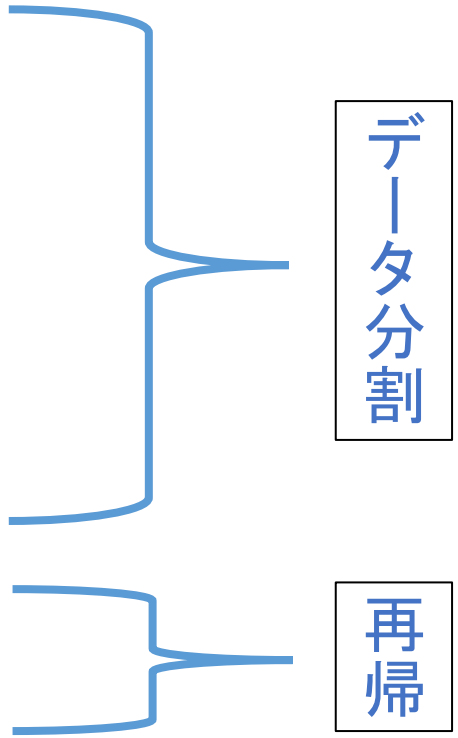
# クイックソートアルゴリズム

入力：配列a[0], ... , a[n-1]
出力：ソート済みの配列a[0], ..., a[n-1]
　　1. QSort(a, 0, n-1)を実行

アルゴリズム　QSort(a, f, t)
　f: 部分配列の左端の要素番号
　t: 部分配列の右端の要素番号
補助変数：
　　i：配列を左(f)から右へみていくときの位置
　　j：配列を右(t)から左へみていくときの位置
　　p：基準値

# クイックソートアルゴリズム

1. {中央にあるデータを基準値pとする}p=?
2. {i, jの値を初期化する} i=?, j=?
3. {配列a[f],...,a[t]をa[f],...,a[j]とa[i],...,a[t]に分割する}
   i<=jが成立する間に、以下を繰り返す
   ？
4. f<jであれば、QSort(a, f, j)を実行
5. i<tであれば、QSort(a, i, t)を実行

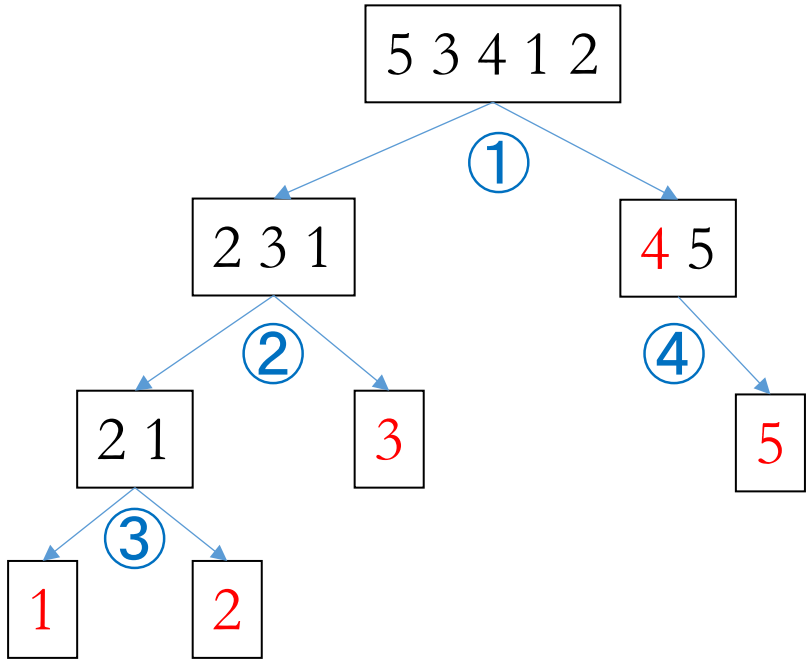


# Question

- QSort(a, 0, 2)を配列a=(20, 10, 30)に適用し、その再帰を最後まで具体的に展開し、ソート結果a=(10, 20, 30)が得られることを理解してみること。また、データ分割の再帰の様子（木構造）も示してみること

# プログラムの実行結果で再帰過程をみてみると

クイックソート（データ分割の再帰）の様子




プログラムの実行結果

処理の対象データ
5 3 4 1 2 ①
処理の対象データ
2 3 1 ②
処理の対象データ
2 1 ③
処理の対象データ
4 5 ④
ソート結果
1 2 3 4 5

# 計算量

- 分割のよさによって変わる

- 分割がすべて（ほぼ）均等な場合
  - n個のデータが左右$\frac{n}{2}$個に分割
  - データが1個まで分割する回数を$k$とすると
    $\frac{n}{2^k} = 1$から$k = \log_2 n$が得られる
  - n個のデータの1回分割に必要な比較・交換回数はn以下なので、計算量は$O(n \log_2 n)$となる

# 計算量

- 分割がすべて偏る場合
  - n個のデータがn-1個と1個に分割
  - データ1個までの分割回数がn-1となる
  - i個のデータの分割にi回の比較回数なので、合計比較回数は$n + (n-1) + \cdots + 2 = (n+2)(n-1)/2$ となり、計算量は$O(n^2)$となる

- 実用上は計算量を$O(n \log_2 n)$と考えてよく、本授業で紹介されていないほかのソート法も含め一番速いソート法と考えてよい

# 第4回演習課題

1. 配列a[f],...,a[t]を、その配列の中央データを基準値とし、左<=右となるように、左：a[f],...,a[j]と右：a[i],...,a[t]に分割する関数void part(int a[], int f, int t)を作成し、その動作を確認するプログラム（ex04-part.c）を作成しなさい。もちろん、関数内に出力文があってもよい。

実行例

```
./a.out
Input the data (end with Ctrl-D)
40 50 30 10 20 1
left part
1 20 10
right part
30 50 40
```

# 第4回演習課題

2. 課題１で作成した関数の中身をクイックソートアルゴリズムに手順1～3として組み込み（課題１の関数を呼び出す、という方法ではなく）、クイックソートアルゴリズム（再帰）を実現する関数void QSort(int a[], int f, int t)を作成し、その動作を確認するプログラム（ex04-qsort.c）を作成しなさい。ただし、QSort()が呼び出される度にその処理の対象となる配列の中身を出力すること。また、その出力に対し、説明ができるようにすること。

実行例

| | |
|---|---|
| ./a.out<br>10 30 20 50 40 100<br>target data<br>10 30 20 50 40 100<br>target data<br>10 20<br>target data | 30 50 40 100<br>target data<br>30 40<br>target data<br>50 100<br>Sorted data<br>10 20 30 40 50 100 |