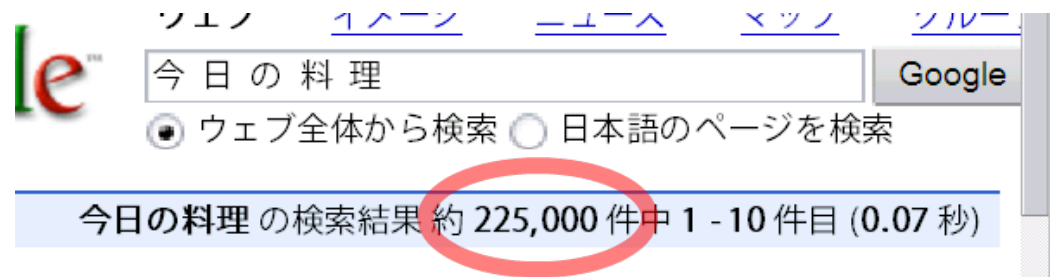


整列 (sorting)

- 問題: n 個のデータを大小順に並べる
(単純化): n 個の整数を小さい順に並べる
 - 色々な場面で使われる
 - データの「下ごしらえ」としても使われる
例: 検索を高速化するために整列しておく
- データ数は, 時として非常に大きくなる
 - 「全学生の氏名を学生証番号順に表示」
 - 「『今日の料理』という言葉を持つページを信頼度順に表示」



国語辞典で先に来るのは？

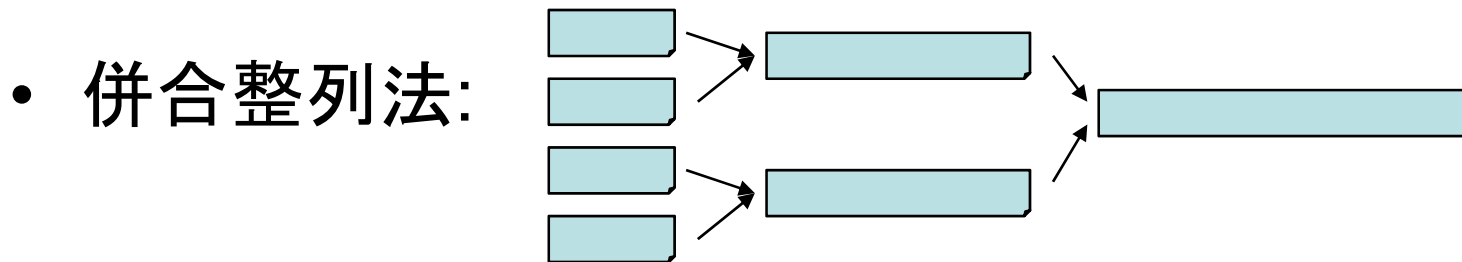
1. しょうほう
2. しょうほう
3. じょうほ
4. じょうほう
5. ショー

最初だけではなく、すべてを順序付けてみよう

整列アルゴリズム

とても沢山ある

- 単純整列法: 1番小さいものを見つけ, 2番目に小さいものを見つけ, ...

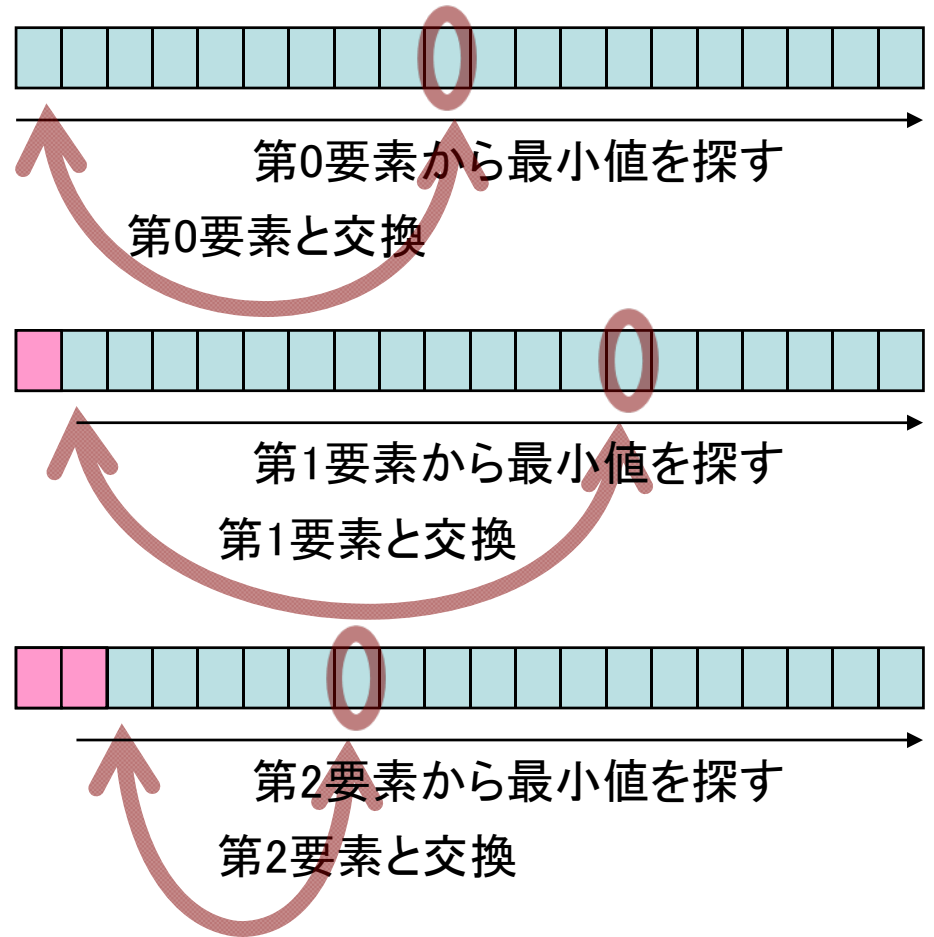


- ビン整列法
- 基数整列法
- クイック整列法

単純整列法

- 方法:

- $i=0,1,2,\dots$ について
列の中で i 番目に小さい数を見つけ, それを第 i 要素へ移動
- 「 i 番目に小さい数」とは列の第 i 要素以降の最小値



単純整列法

```
def min_index(a, i)
  min = i
  for j in i..(a.length()-1)
    if a[j] < a[min]
      min = j
    end
  end
  min
end
```

配列 a の第 i 要素から終わりまでの間での最小要素の添え字を返す

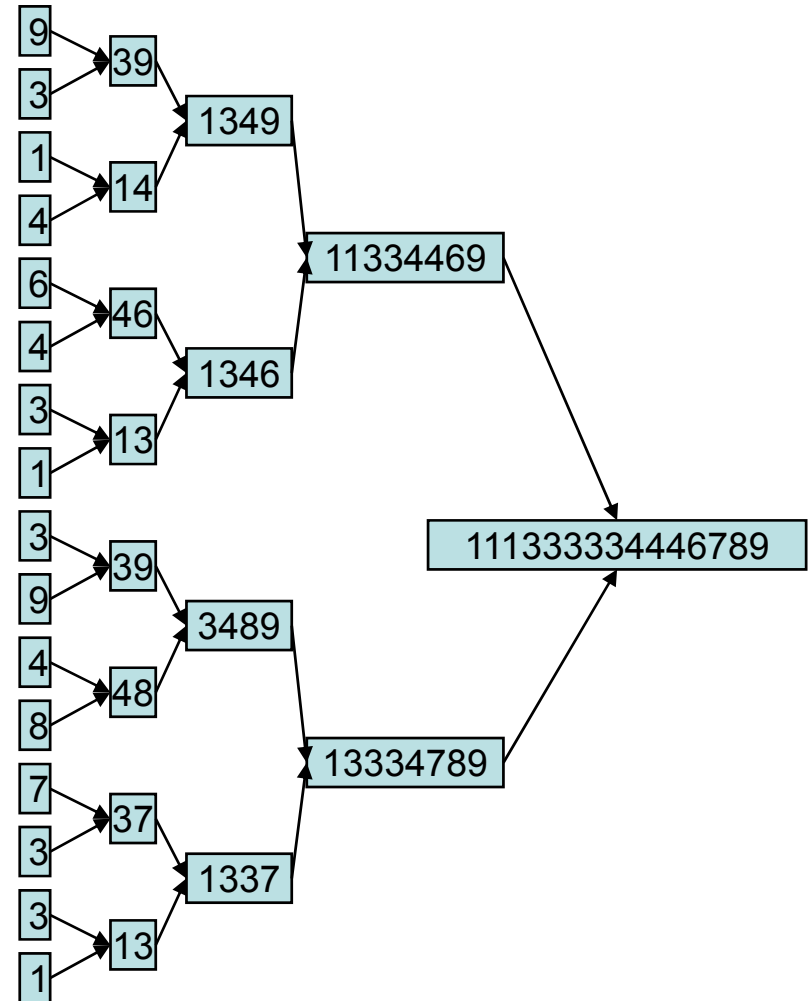
```
def simplesort(a)
  for i in 0..(a.length()-1)
    k = min_index(a, i)
    tmp = a[i]
    a[i] = a[k]
    a[k] = tmp
  end
  a
end
```

第0要素から順にそれ以降の最小要素を入れていく

simplesort.rb

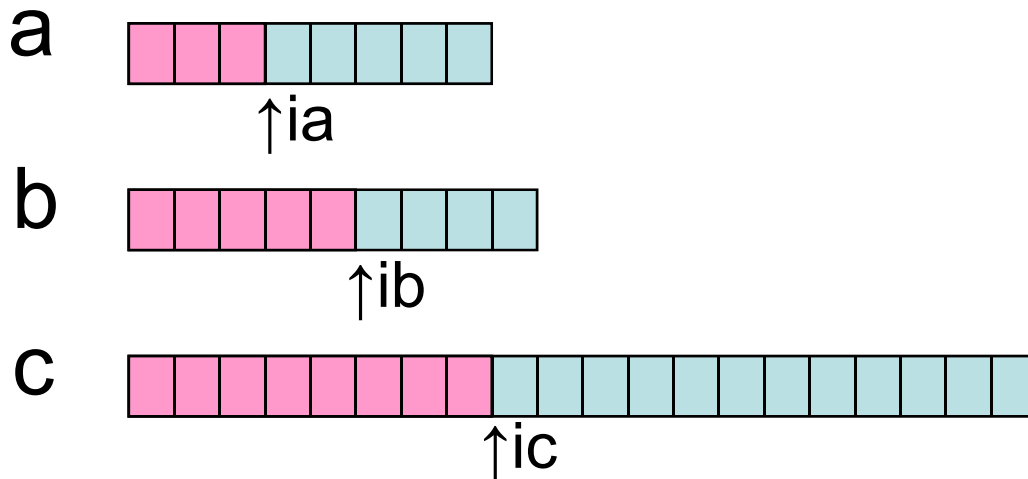
併合整列法 (merge sort)

- 前提:
 - 整列済みの列が
沢山ある
- 方法:
 - 2つの列を順序よく
くっつけて1つにする
(併合)
 - 列が1つになるまで
繰り返す



併合整列法: 整列済の列の併合

- 方法 (a, b を併合した c を作る):
 - c を作る (長さは (a の長さ) + (b の長さ))
 - 2つの列の先頭を比べ, 小さい方をコピーする
 - どちらかの最後に至るまで続ける
 - 残った列をコピーする



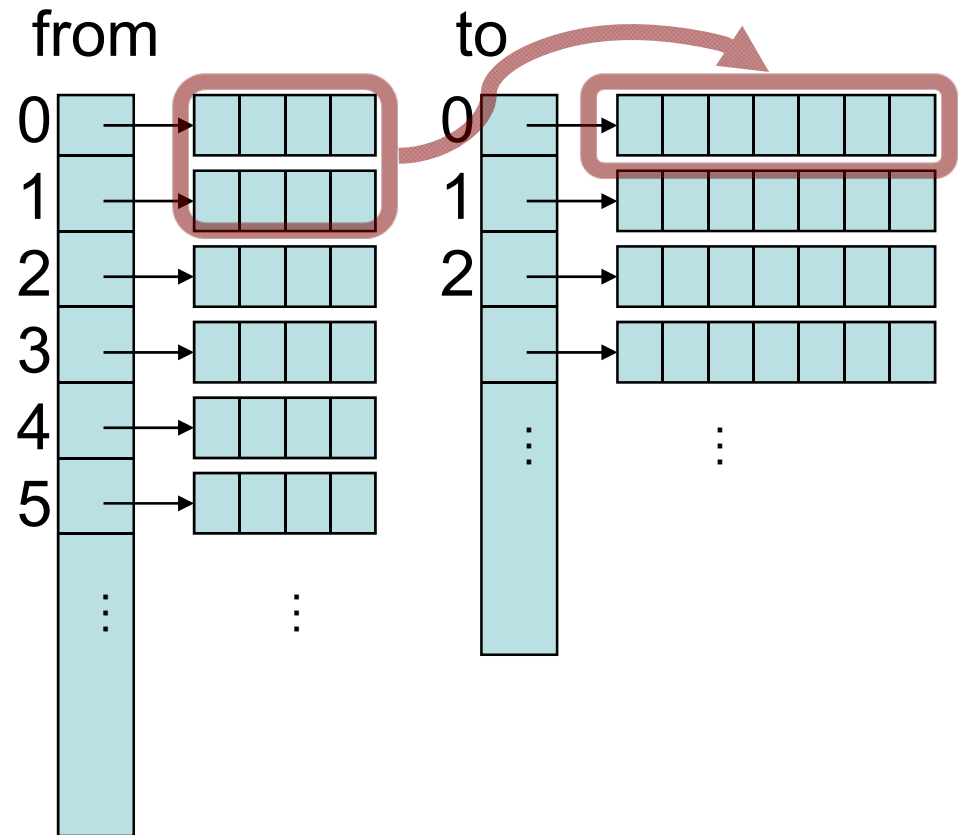
併合整列法: 併合を繰り返す

- 前提:

- 併合される列は配列の配列 from にしまわれている

- 方法:

- from が長さ1の配列になるまで繰り返す
 - from の半分の長さの配列 to を作る
 - from の第 $(2i)$ 要素と第 $(2i+1)$ 要素を併合して to の第 i 要素にしまう
 - 現在の to を新たな from にする



併合整列法

```
def is_even(n)
  n % 2 == 0
end
```

中で定義しても
loadしてもよい

fromが奇数個のときは、
最後の要素は
併合せずにコピーする

最後はfromの第0要素に
整列済みの配列が入って
いる

```
def mergesort(a)
  n = a.length()
  from = Array.new(n)
  for i in 0..(n-1)
    from[i] = [a[i]]
  end
  while n > 1
    to = Array.new((n+1)/2)
    for i in 0..(n/2-1)
      to[i] = merge(from[i*2], from[i*2+1])
    end
    if !is_even(n)
      to[(n+1)/2-1] = from[n-1]
    end
    from = to
    n = (n+1)/2
  end
  from[0]
end
```

各要素が大きさ
1の配列である
配列fromを作る

半分の長さの
配列toを作る

fromの2要素を
併合してtoにコ
ピー

toを新たなfromと
する

fromの要素が1つに
なるまで繰り返す

mergesort.rb

2つの整列済み配列の併合

```
def merge(a, b)
  c = Array.new(a.length() + b.length())
  ia = 0
  ib = 0
  ic = 0
  while ia < a.length() && ib < b.length()
    if a[ia] < b[ib]
      c[ic] = a[ia]
      ia = ia + 1
      ic = ic + 1
    else
      c[ic] = b[ib]
      ib = ib + 1
      ic = ic + 1
    end
  end
end
```

a,b,c の現在の先頭を示す添え字

a, b の先頭から小さいほうを c にコピー

```
if ia < a.length()
  for i in ia..(a.length() - 1)
    c[ic] = a[i]
    ic = ic + 1
  end
else
  if ib < b.length()
    for i in ib..(b.length() - 1)
      c[ic] = b[i]
      ic = ic + 1
    end
  end
end
end
c
end
```

灰色の if 文は、片方の配列が空になったときの処理 (a または b の残りを c にコピー)

mergsort.rb

アニメ化はこの部分に対して行なう(前のページの部分はそのまま)

併合整列法の再帰版

```
def mergesort(a)
  submergesort(a,0,a.length())
end

def submergesort(a,i,n)
  if n<2
    if n==0
      []
    else
      [a[i]]
    end
  else
    merge(submergesort(a,i,n/2),submergesort(a,i+n/2,n-n/2))
  end
end
```

繰り返し版よりわかりやすい！

配列 a の i 要素から n 個を整列した結果を配列として返す

要素のない配列(長さ 0)

$a[i]$ という一つの要素から成る配列

演習: これのアニメはどうなるか?

mergesort_rec.rb

課題: アルゴリズムによる速度差の実測

- 平方根・整列の複数のアルゴリズムに対応したプログラムを作り, 速度の違いを調べる
 - 平方根の場合: x/δ の大きさによって時間がどう変化するかをグラフを描いてみよ, 1秒間に何回計算できるかで比べよ
 - 整列の場合: ランダムな列を作り, それを整列させてみよ. 列の長さで時間がどう変化するかをグラフを描いてみよ
- グラフから, 実行時間を予測する式を推定せよ

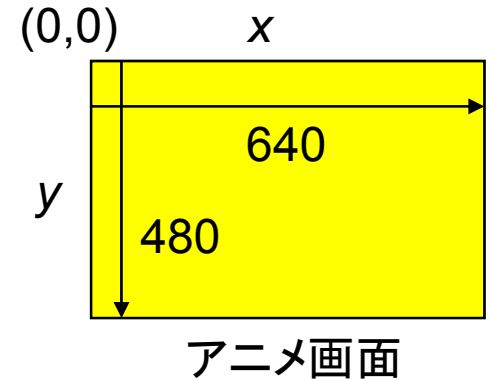
```
load("./randoms.rb ") # randoms(id,size,max)
load("./bench.rb")    # run(function_name, x, v)
load("./simplesort.rb") # simplesort(a)
load("./mergesort.rb") # mergesort(a)
```

```
def compare_sort(max, step)
  for i in 1..(max/step)
    x=i*step
    a=randoms(i,x,1)
    run("simplesort", x, a)
    a=randoms(i,x,1)
    run("mergesort", x, a)
  end
end
```

ソートプログラムの動きを 自分で可視化して理解しよう

- 数値配列を背の高さの違う人形の並びとして表現
 - 人形同士には身長を比較する不等号演算が定義されている
 $>$ $>=$ $<=$ $<$
- ソートのときに行なわれる要素の入れ替えを人形の入れ替えのアニメで表示
- プログラムの動きの理解に重要な人形をその都度フラッシュさせる(適当な時間, 色をつけて目立つようにする)
- これらを行なうためにはisrbを使用し, プログラムの先頭に
`include(Komaba::MiniStar)`
を入れる

人形の並びの生成



```
def index_to_x(i)  
  i * 30 + 60  
end
```

第 i 要素の人形の x 座標

```
def index_to_y(i)  
  400  
end
```

第 i 要素の人形の y 座標 (定数)

```
a = Array.new(16)
```

デフォルトの画面には16個程度が最適

```
for i in 0..(a.length() - 1)  
  a[i] = Person.new()  
  a[i].x = index_to_x(i)  
  a[i].y = index_to_y(i)  
  a[i].height = rand(60) + 40  
end
```

- Person.new()で人形を生成
- 人形の x 座標と y 座標を各々 .x .y と記述 (文字列の長さを .length() と書くのと同様), 代入も可能
- 人形の身長は .height と記述

0以上60未満の乱数

()があつたりなかったりするが、オブジェクト指向の回に詳しく説明

人形の入れ替え

```
tmp = a[i]
```

```
a[i] = a[k]
```

```
a[k].move(index_to_x(i), index_to_y(i))
```

```
a[k] = tmp
```

```
tmp.move(index_to_x(k), index_to_y(k))
```

a[k]にあった人形を第 i 要素の位置へ動かす

.move
に注意

tmpに一時的に保存してあった人形を第 k 要素の
位置へ動かす

本来のプログラムとうまく対応していることに注意

人形のフラッシュ

- 人形は画面では黒地に白で表示される
- これをふわっと一瞬色をつけるには, これまでに使ってきたRGB配列を使って

人形.flash(秒, RGB配列)

と書く (秒は実数)

- 色のデフォルト (色を指定しなかったときの標準色) は [1, 0, 0] つまり赤色, たとえば

a[k].flush(0.5)

- 自分でいろいろいじって見ると面白い

```
include(Komaba::MiniStar)

def simplesort(a)
  for i in 0..(a.length()-1)
    k = min_index(a, i)
    a[k].flash(0.5)
    tmp = a[i]
    a[i] = a[k]
    a[k].move(index_to_x(i), index_to_y(i))
    a[k] = tmp
    tmp.move(index_to_x(k), index_to_y(k))
  end
end
a
end
```

アニメを開始し, 終了させる

```
def index_to_x(i)
  i * 30 + 60
end

def index_to_y(i)
  400
end

a = Array.new(16)

for i in 0..(a.length() - 1)
  a[i] = Person.new()
  a[i].x = index_to_x(i)
  a[i].y = index_to_y(i)
  a[i].height = rand(60) + 40
end

start()
simplesort(a)
halt()
```

animated_simplesort.rb

単純整列法のアニメーション

```
include(Komaba::MiniStar)
```

```
def merge(a, b)
```

```
  for i in 0..(a.length() - 1)
    a[i].flash(0.5, [1, 0, 0])
  end
```

```
  for i in 0..(b.length() - 1)
    b[i].flash(0.5, [0, 0, 1])
  end
```

```
  c = Array.new(a.length() + b.length())
```

```
  ia = 0
```

```
  ib = 0
```

```
  ic = 0
```

```
  while ia < a.length() && ib < b.length()
```

```
    if a[ia] < b[ib]
```

```
      c[ic] = a[ia]
```

```
      move(c[ic], ic - ia)
```

```
      ia = ia + 1
```

```
      ic = ic + 1
```

```
    else
```

```
      c[ic] = b[ib]
```

```
      move(c[ic], ic - (a.length() + ib))
```

```
      ib = ib + 1
```

```
      ic = ic + 1
```

```
    end
```

```
  end
```

マージされる
2つの配列を
それぞれ赤と
青にフラッシュ

ちょっと考える
必要

```
if ia < a.length()
```

```
  for i in ia..(a.length() - 1)
```

```
    c[ic] = a[i]
```

```
    move(c[ic], ic - i)
```

```
    ic = ic + 1
```

```
  end
```

```
else
```

```
  if ib < b.length()
```

```
    for i in ib..(b.length() - 1)
```

```
      c[ic] = b[i]
```

```
      move(c[ic], ic - (a.length() + i))
```

```
      ic = ic + 1
```

```
    end
```

```
  end
```

```
end
```

```
c
```

```
end
```

ちょっと考える
必要

ここにmergesort.rbの
mergesortの定義を入れる

animated_mergesort.rb

併合整列法のアニメーション

```
def index_to_x(i)
  i * 30 + 60
end
```

```
def next_y(person)
  if person.y == 400
    200
  else
    400
  end
end
```

マージをするたびに
人形の y 座標を
上と下で交代させる

```
def move(person, dx)
  person.move(person.x + dx * 30, next_y(person))
end
```

人形の移動の記述を短くするために関数化

```
a = Array.new(16)

for i in 0..(a.length() - 1)
  a[i] = Person.new()
  a[i].x = index_to_x(i)
  a[i].y = 400
  a[i].height = rand(60) + 40
end

start()
result = mergesort(a)
halt()
```

animated_mergesort.rb

クイック整列法のアニメーション

クイックソート (quick sort)

- 配列の途中の適当な要素(ピボット)を選ぶ
- 左から順にピボットより大きい要素を探し, 右側からピボットより小さい要素を探し, それらを入れ替えていく
- 両側の探索がぶつかったところの左右をみると, 左側のどれもが, 右側のどれよりも小さくなっている
- この左右の部分配列それぞれについて再帰的に同じことをする(配列は小さくなっていく)
- 配列要素が1個になれば再帰の終端

以下のプログラムアニメーションでそれを実感しよう

クイックソート

```
def qsort(array, l, r)
  if l == r
    return
  end
  pivot = (l + r) / 2 + 1
  start_l = l
  start_r = r
  median = array[pivot]
```

終端

第l要素から第r要素
までのあいだを
クイックソート

とりあえずピ
ボットを真ん中
あたりから選ぶ

左から順にピボットより大きい要素を探し、右側からピボットより小さい要素を探し、それらを入れ替えていく

```
while true
  while array[l] < median
    l += 1
  end
  while median < array[r]
    r -= 1
  end
  if l >= r
    break
  end
  tmp = array[l]
  array[l] = array[r]
  array[r] = tmp
  l += 1
  r -= 1
end
qsort(array, start_l, l - 1)
qsort(array, l, start_r)
end
```

繰り返しから
抜ける

```
include(Komaba::MiniStar)
```

```
def qsort(array, l, r)  
  if l == r  
    return  
  end  
  pivot = (l + r) / 2 + 1  
  array[pivot].flash(1.5)  
  start_l = l  
  start_r = r  
  median = array[pivot]
```

簡単かつ素直に
アニメ化できている

```
while true  
  while array[l] < median  
    l += 1  
  end  
  while median < array[r]  
    r -= 1  
  end  
  if l >= r  
    break  
  end  
  tmp = array[l]  
  array[l] = array[r]  
  array[r].move(index_to_x(l), index_to_y(l))  
  array[r] = tmp  
  tmp.move(index_to_x(r), index_to_y(r))  
  l += 1  
  r -= 1  
end  
qsort(array, start_l, l - 1)  
qsort(array, l, start_r)  
end
```

animated_qsort.rb

```
def index_to_x(i)
  i * 30 + 60
end

def index_to_y(j)
  400
end

a = Array.new(16)

for i in 0..(a.length() - 1)
  a[i] = Person.new()
  a[i].x = index_to_x(i)
  a[i].y = index_to_y(i)
  a[i].height = rand(60) + 40
end

start()
result = qsort(a, 0, 15)
halt()
```

この部分で、単純ソートと異なるのはここだけ

animated_qsort.rb