

ソートアルゴリズム

(2016/8/10)

アルゴリズムの定番である「データの並び替え（ソートアルゴリズム）」です。今回は入力と出力の配列を分けることでCSアンプラグド風に「1個の配列で処理することにこだわらない」ようにしてみました。

CSアンプラグドのソーティングアルゴリズム

- 男の子が選択ソートを、女の子がクイックソートを実演しています。
- 配列のような「1列に並んだ決まった場所に置く」のではなく、「机の上のわかりやすい場所に置く」ことで、「グループからいちばん重いものを探す」「1個を基準に軽いものと重いものに分ける」ようなアルゴリズムの考え方の本質をわかりやすく示しています。

選択ソート

- 元の配列in[]から一番小さい要素を選び、結果の配列out[]に移していきます。

```
//in[]配列！53728614作る。
in=配列！作る。in[]乱数(50)書く！20 繰り返す。
ラベルin[]作る。
out=配列！作る。
[]
[]min=in[1] 読む。pos=1[]
[]n[]
[]v=in[pos]読む。
[]v<min[]ならmin=vpos=pos+1実行。
[]in[]要素数？) 繰り返す。
out[pos]=min[]書く。in[pos]位置で消す。
[]in[]要素数？) 繰り返す。
ラベルout[]作る 次の行。
```

挿入ソート

- 空の配列out[]を用意して、元の配列in[]の要素を1個ずつoutの適切な場所に入れていきます。

```
//in[]配列！53728614作る。
in=配列！作る。in[]乱数(50)書く！20 繰り返す。
ラベルin[]作る。
out=配列！作る。
[]i[]
[]v=in[i]読む。
[]n=0[]
[]j[]
```

```

v2=out[j]読む。
v1>v2ならn=j実行。
out要素数? ) 繰り返す。
out[n+1]v1挿入。
in要素数? ) 繰り返す。
ラベルout作る 次の行。

```

クイックソート

- quick関数は、渡された配列aの最後の要素を基準pにして、それ以外の要素を「pより小さい値の配列left」と「p以上の値の配列right」に分けます。
- そしてleftをソートした結果」と「p」と「rightをソートした結果」を連結して返します。
- quick関数の冒頭の「 | ... | 」の部分では、引数aに続いて、「;」の後でローカル変数ret, n, p, left, right, vを定義しています。

```

quick=a ; ret n p left right v
ret=a
n=a要素数?。
n<1なら「
p=a[n] 読む。
left=配列! 作る。
right=配列! 作る。
i
v=a[i]読む。
v<pならleft[v]書く」そうでなければright[v]書く」実行。
n-1繰り返す。
ret=配列! 作る ( left quick p right quick 連結。
  ) 実行。
ret

```

```

//in配列! 53728614 作る。
in=配列! 作る。 in乱数(50) 書く」! 2 0 繰り返す。
ラベルin作る。
out=in quick
ラベルout作る 次の行。

```

From:

<https://dolittle.eplang.jp/> - プログラミング言語「ドリトル」

Permanent link:

https://dolittle.eplang.jp/tips_sorting?rev=1515325943

Last update: **2018/01/07 20:52**

