アルゴリズムとデータ構造

コンピュータサイエンスコース
知能コンピューティングコース

第3回

バケットソート，基数ソート

塩浦昭義

情報科学研究科 准教授

shioura@dais.is.tohoku.ac.jp

http://www.dais.is.tohoku.ac.jp/~shioura/teaching

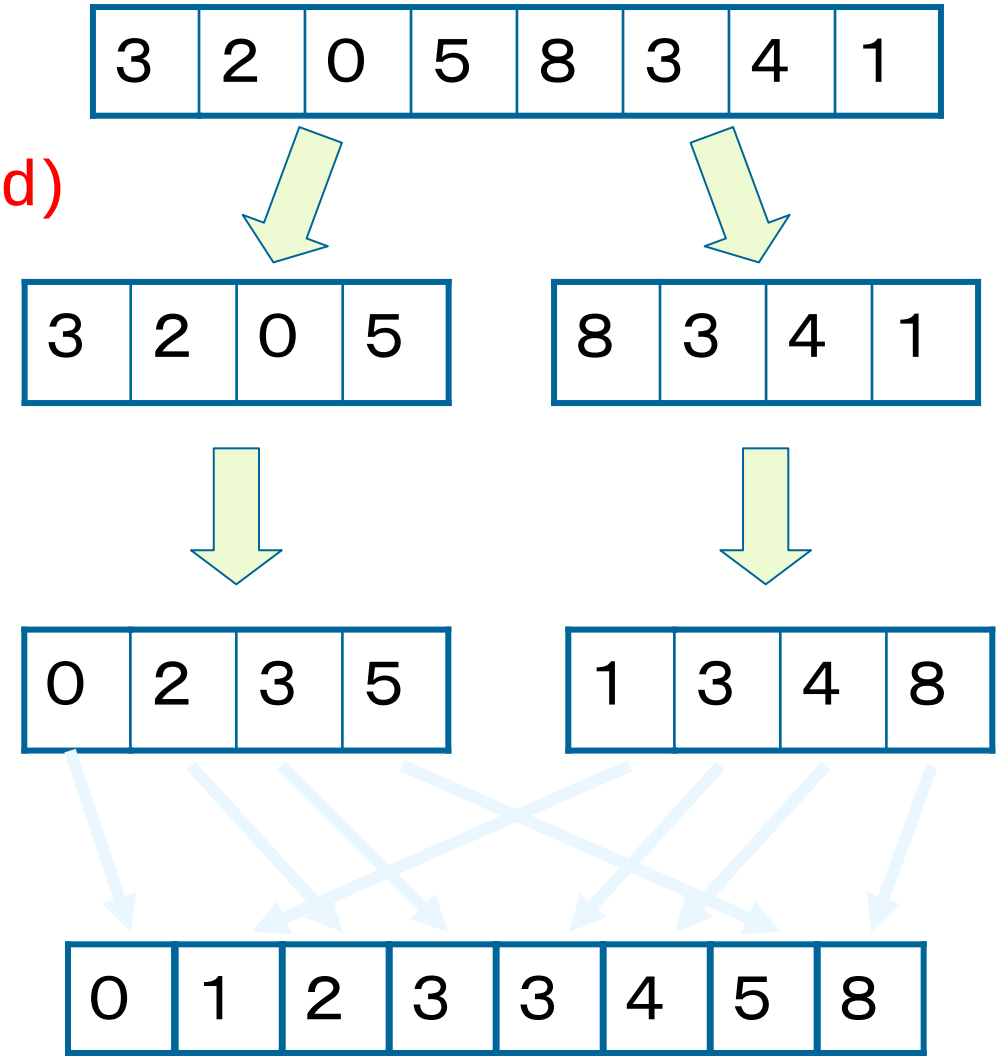
マージソートのアイディア
Idea of Merge Sort

- アイディア：分割統治法
(divide-and-conquer method)

①与えられた配列を2分割

②2分割された配列をそれぞれ再帰的にソート

③ソートされた2つの配列をマージ（統治）



マージソートの擬似コード
Pseudo-Code for Merge Sort

- 手続き（関数）mergesort(i, j, A)
  - 配列Aの要素A[i], A[i+1],…,A[j]をマージソートによりソートする

```
mergesort(i, j, A)
{
  mid = i,i+1,…,j の中央値;
  mergesort(i, mid, A);
  mergesort(mid+1, j, A);
  A[i],…,A[mid]とA[mid+1],…,A[j]をマージ;
}
```

手続きmergesortの中で
自分自身を呼び出す
---再帰呼び出し(recursive call)

クイックソートのアイディア
Idea of Quick Sort

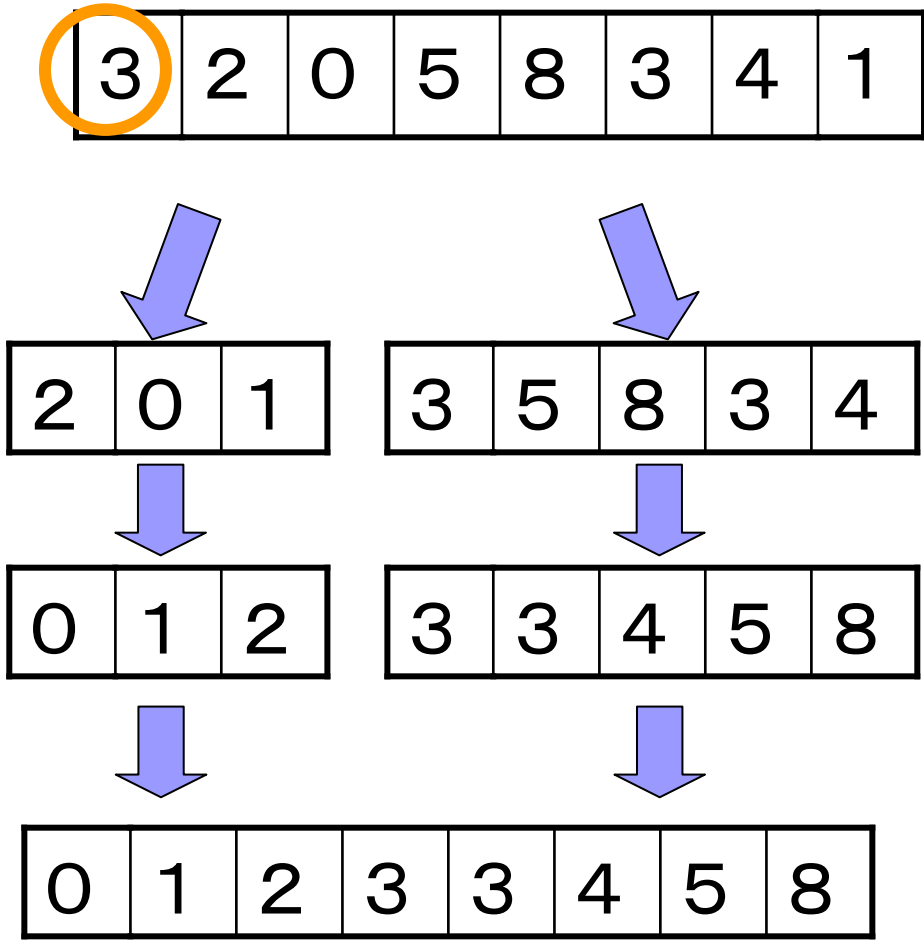
軸要素 = 3

①A[1], …, A[n]から
ひとつの値（軸要素, pivot）
　を選ぶ

② 軸要素未満の要素と
それ以外に分割

③2分割された配列を
それぞれ再帰的にソート

④ソートされた2つの配列
　をつなげる



クイックソートの擬似コード
Pseudo-Code for Quick Sort

- 手続き（関数）quicksort(i, j, A)
  - 配列Aの要素A[i], A[i+1],…,A[j]をクイックソートによりソートする

```
quicksort(i, j, A)
{
  p = A[i],…,A[j] の中から選んだ軸要素;
  k = 要素A[i],…,A[j]のうち，p未満の要素の数;
  A[i],…,A[i+k-1]にp未満の要素を入れる;
  A[i+k],…,A[j]にp以上の要素を入れる;
  quicksort(i, i+k-1, A);
  quicksort(i+k, j, A);
}
```

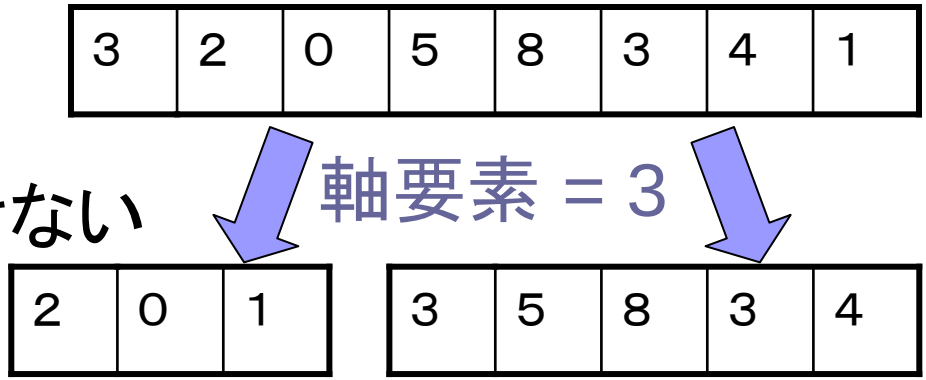
再帰呼び出し(recursive call)

軸要素の選び方
How to Choose Pivot

良い選び方：

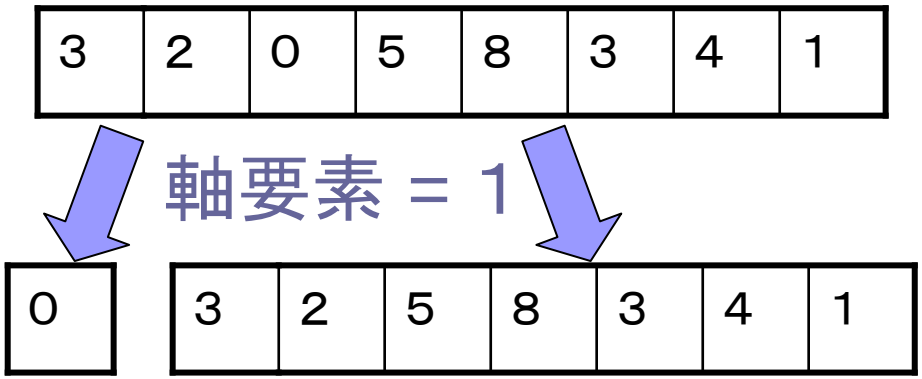
配列をほぼ二等分する

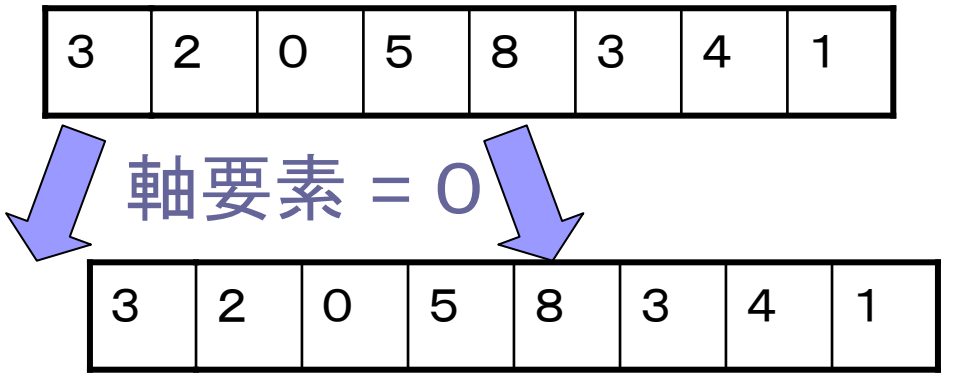
軸要素の選択に時間をかけない



悪い選び方：

2分された配列の大きさがアンバランス



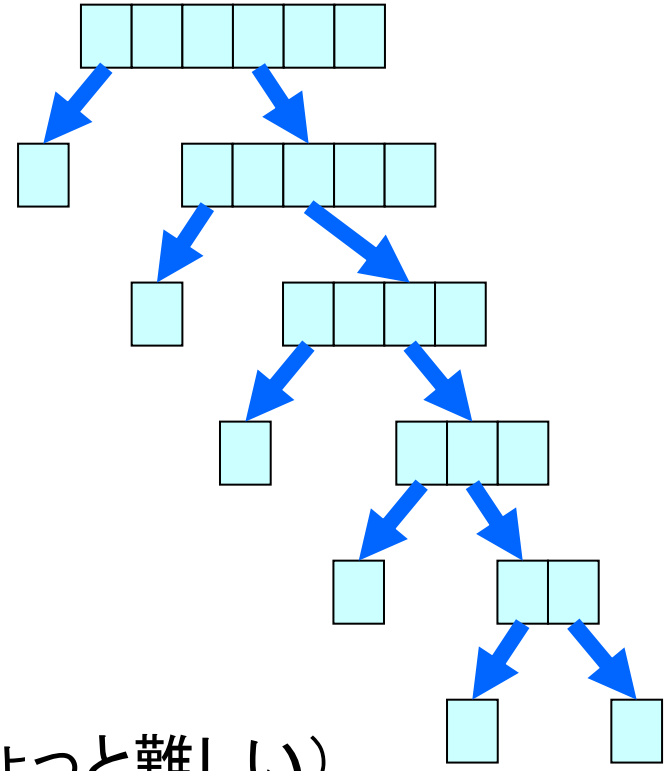


クイックソートの計算時間
Time Complexity of Quick Sort

- 各レベルでの分割に必要な計算時間の合計＝ cn
- 分割の深さ: 最悪の場合 n

∴最悪計算時間＝cn^2 = $O(n^2)$

（帰納法でも証明可能）



実用的には，数列がほぼ半分に
分割されることが多い

➔ 分割の深さは O(log n) に近い

➔ 実用上の計算時間はO(n log n) に近い

平均時間計算量は　O(n log n)　（解析はちょっと難しい）

T(n)＝クイックソートでn個の要素をソートするときの計算時間

証明すること：$T(n)=O(n^2)$

帰納法により証明「$T(n)\leq cn^2$」

cn＝分割に要する時間

n=1のときは明らか

n=kに成立すると仮定

n=k+1で成り立つことを示す

n個の要素がk個とn−k個に分割されたと仮定 $(1\leq k\leq n-1)$

すると $T(n) = cn + T(k) + T(n-k)$

cn：分割　$T(k)$：k個をクイックソート　$T(n-k)$：$n-k$個をクイックソート

帰納法の仮定より

$\leqq cn + ck^2 + c(n-k)^2$

$= c(n + k^2 + n^2 - 2nk + k^2)$

$\leqq cn^2$　　$n + 2k^2 - 2nk^2 \leqq 0$

よって $n = k+1$ のときも成立

証明おわり //

（$n \geqq 2$, $1 \leqq k \leqq n-1$ の条件の下で成り立つ. 次ページ参照）

$n+2k^2-2nk \leq 0$ の証明 (仮定 $n \geq 2$, $1 \leq k \leq n-1$)

$f(k)=2k^2-2nk+n$ とおくと,

$f(k)$ が最大となるのは $k=1$ または $k=n-1$ のときである

$f(1)=2-2n+n=2-n \leq 0$

($n \geq 2$ より)

$f(n-1)=2(n-1)^2-2n(n-1)+n$

$=-2(n-1)+n=-n+2 \leq 0$

よって $1 \leq k \leq n-1$ のとき $f(k) \leq 0$ が示された. //

データ構造
Data Structures

- アルゴリズムの中で，与えられた問題に関連するデータ集合を管理するための道具
- 良いデータ構造とは？
  - データ管理に必要な計算時間が短い
  - シンプル
  - 必要な領域計算量（記憶容量，領域量）が小さい

集合を管理する
Maintenance of Sets

- 整数の集合が与えられている

  4, 5, 8, 2, 9, 1, 3

- ときどき，新しい整数が追加される

  7を追加 ➔ 4, 5, 8, 2, 9, 1, 3, 7

- ときどき，ある整数が削除される

  9 を削除 ➔ 4, 5, 8, 2, 1, 3, 7

- アルゴリズム（プログラム）の中でどのように表現するか？

配列の利用（その1）
Use of Arrays

配列A[1], A[2], …, A[N]を使って表現（N: 十分大きな数）

集合の中に整数 k が　ある➔ A[k] = 1,　ない➔ A[k] = 0

4, 5, 8, 2, 9, 1, 3 ⇒

1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	0	0	1	1	0

集合の中に整数 k が複数存在する場合も対応可能

A[k] = 集合の中に存在する k の個数

4, 1, 8, 2, 8, 1, 3, 1 ⇒

1	2	3	4	5	6	7	8	9	10
3	1	1	1	0	0	0	2	0	0

整数 k を追加 ➔ A[k] を1増やす --- O(1) 時間で可能

整数 k を削除 ➔ A[k] を1減らす --- O(1) 時間で可能

欠点: 整数 N より大きい整数が追加されるとダメ

実際の集合のサイズより大きい配列が必要

無駄な領域計算量

配列の利用（その2）
Use of Arrays

集合に含まれる整数を配列に代入（非負の整数を仮定）

4, 5, 8, 2, 9, 1, 3 →

1	2	3	4	5	6	7	8	9	10
4	5	8	2	9	1	3	-1	-1	-1

空のところには-1を入れる

7 を追加 →

1	2	3	4	5	6	7	8	9	10
4	5	8	2	9	1	3	7	-1	-1

O(1)時間で可能

5 を削除 →

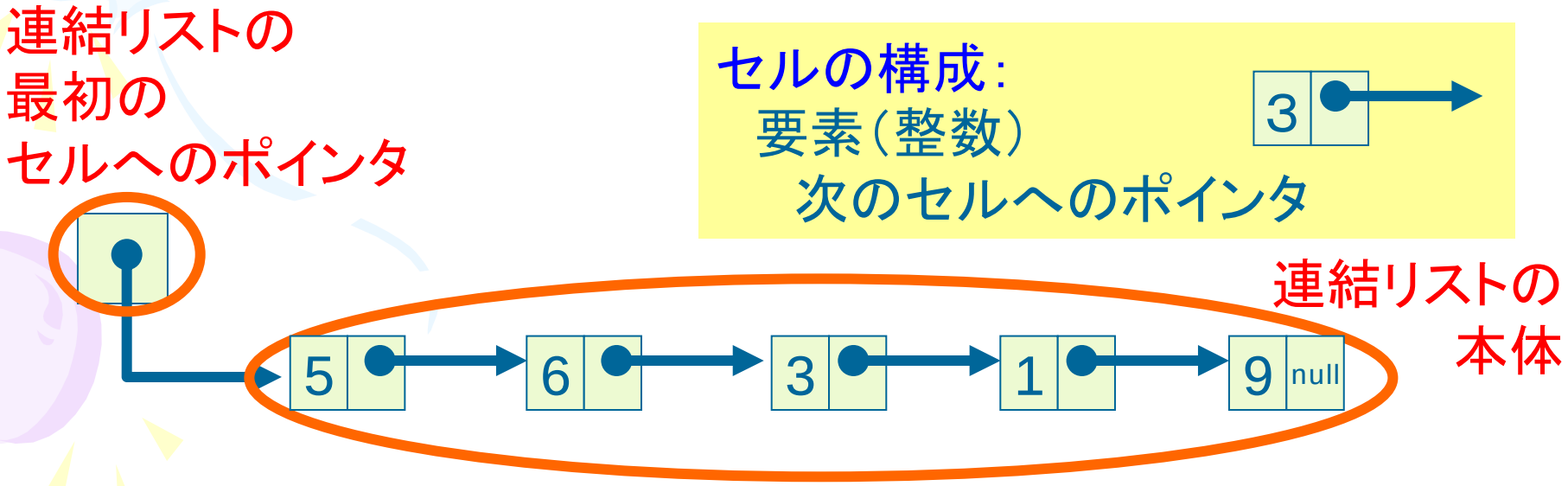
1	2	3	4	5	6	7	8	9	10
4	8	2	9	1	3	7	-1	-1	-1

“5”より後ろの整数を移動させる
➔ O(n)時間（n: 集合のサイズ）

連結リストの利用
Use of Linked Lists

連結リスト:「セル」と呼ばれる基本要素をポインタにより連結したもの

- 必要な領域計算量は集合のサイズに等しい
- 要素の追加はO(1)時間で可能
- 先頭の要素の削除はO(1)時間で可能
- 先頭以外の要素の削除はO(1)時間では不可能

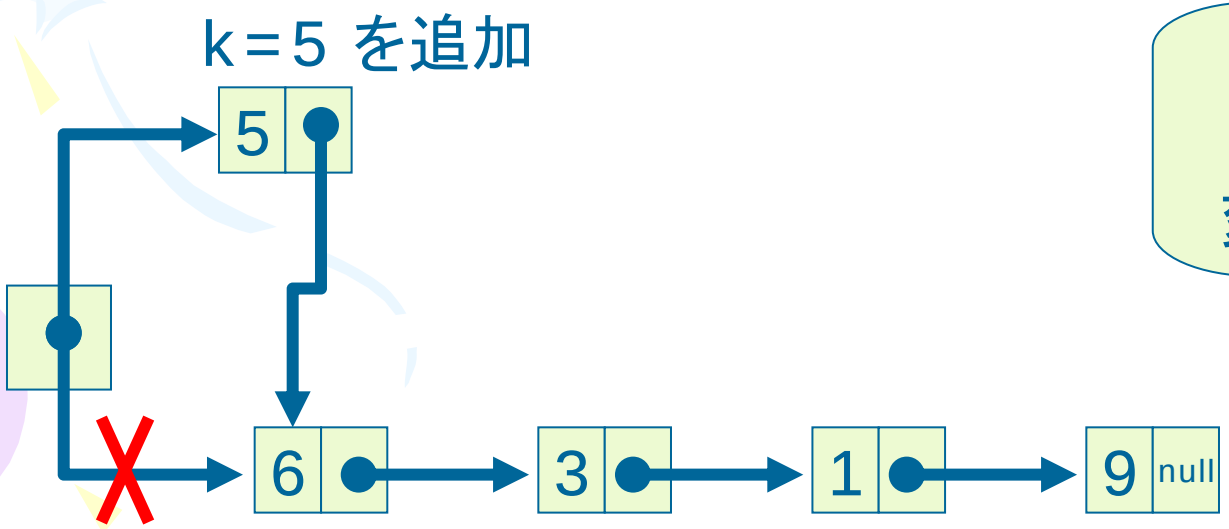


連結リスト:要素の追加
Linked Lists: Addition of Elements

リストの先頭への整数kの追加---O(1) 時間で可能

入力:リスト，追加する整数 k　　出力:新たなセルを追加したリスト

1. 新しいセル C を準備，セルに整数 k と書く
2. Cの次のセルへのポインタを，現在の最初のセルとする
3. 連結リストの最初のセルへのポインタを，Cに変更



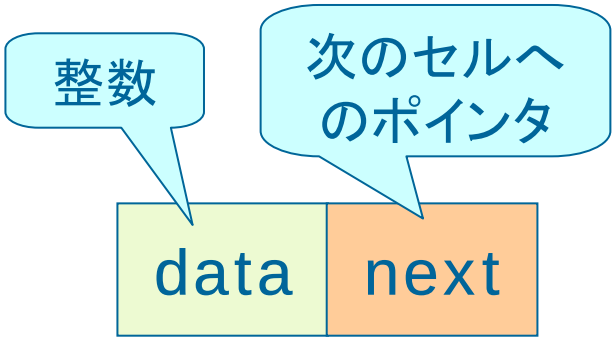
ポインタ変更の
順番を間違えると
変なリストができる

連結リストの実装
Implementation of Linked Lists

C言語での連結リストの実装例

セルを表す構造体の定義

```
struct cell {
  int data;
  struct cell *next;
};
```



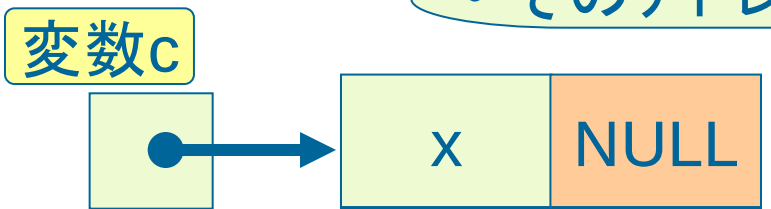
新しいセルを作る関数

```
struct cell *newcell(int x)
{
  struct cell *c;

  c = (struct cell *) malloc(sizeof(struct cell));
  c->data = x;
  c->next = NULL;
  return(c);
}
```

引数 x はセルの要素となる
新しく作ったセルへのポインタを出力

- セルの分の記憶領域を確保
- そのアドレスを c に代入



要素の追加の実装

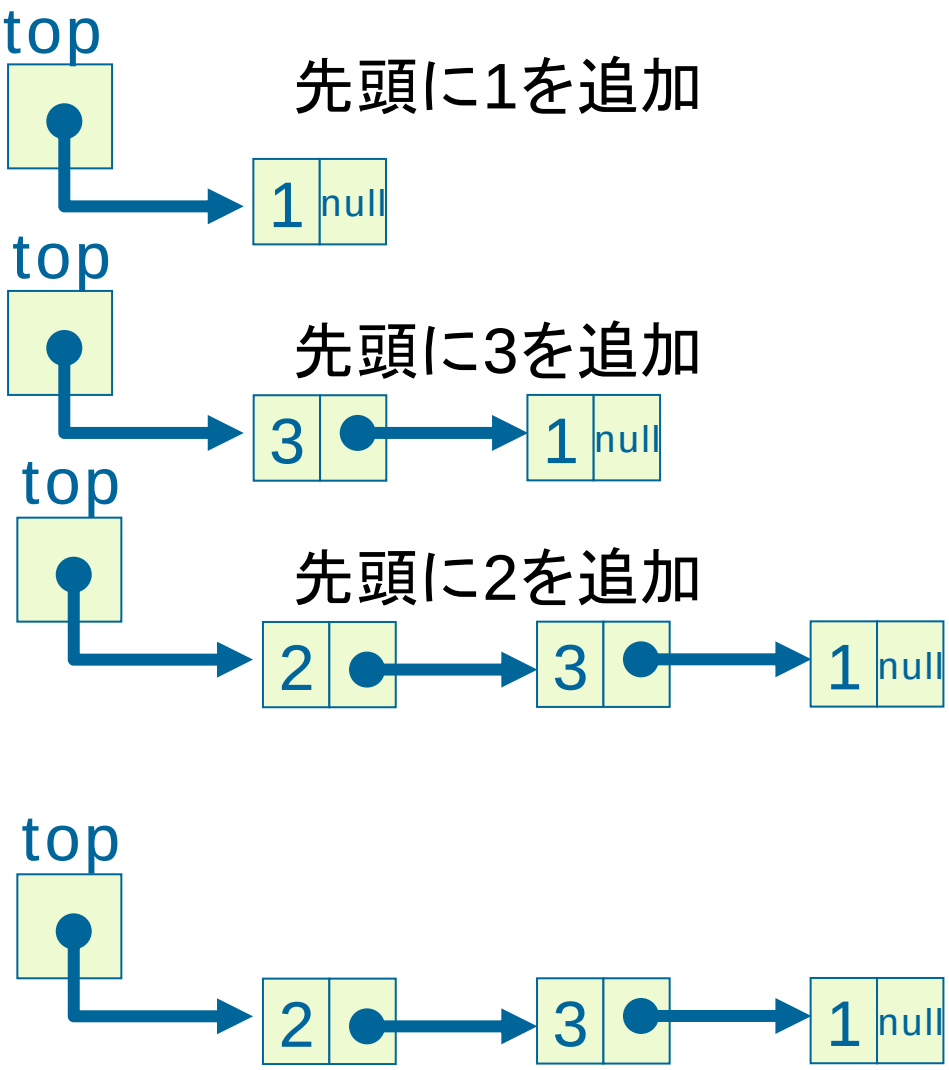
```
main()
{
  struct cell *top, *cell;

  cell = newcell(1);
  cell->next = NULL; top = cell;

  cell = newcell(3);
  cell->next = top; top = cell;

  cell = newcell(2);
  cell->next = top; top = cell;

  cell = top;
   do {
     printf("%d ", cell->x);
     cell = cell->next;
   } while (cell != NULL);
   printf("¥n");
}
```



先頭から順番に，各セルの要素を表示