アルゴリズムとデータ構造
第9回: ソーティング(1)

担当: 上原隆平(uehara)

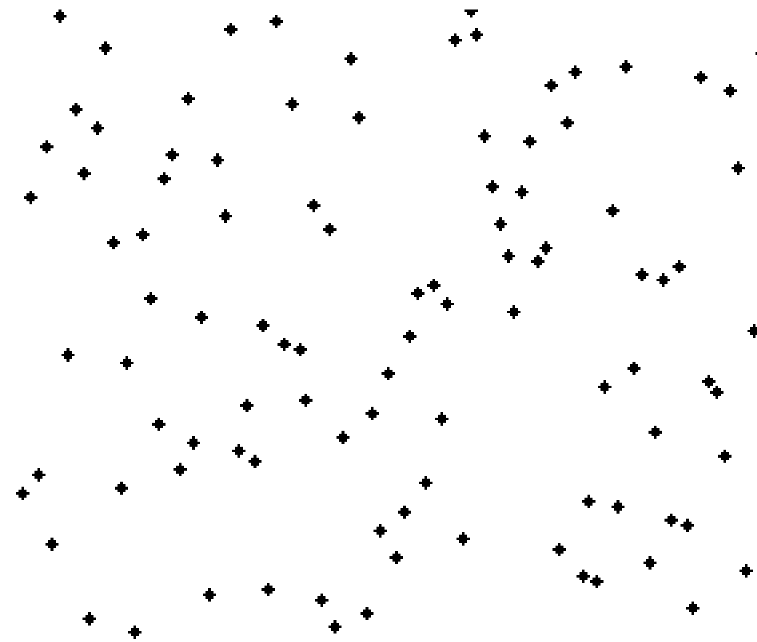
2014/05/13

ソーティング(Sorting)

- 与えられたデータを順序よく並べる
  - 数値データ: 昇順、降順

65	12	46	97	56	33	75	53	21	入力データ
12	21	33	46	53	56	65	75	93	昇順ソート
93	75	65	56	53	46	33	21	12	降順ソート

  - 文字列データ: 辞書式順序
    e.g., aaa, aab, aba, abb, baa, bab, bbc, bcb
- ソーティングアルゴリズム
  - バブルソート, インサーションソート, シェルソート, ヒープソート, クイックソート, マージソート, トポロジカルソート

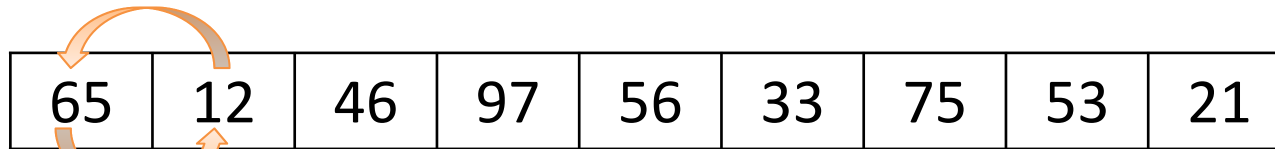


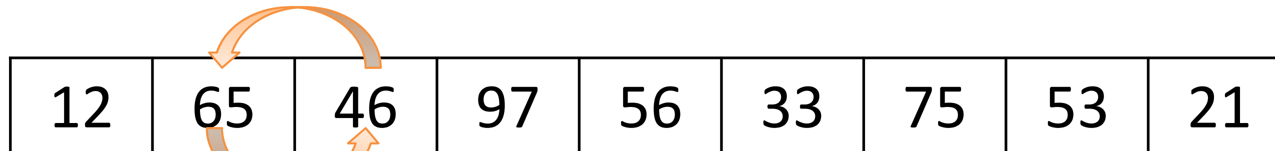
BUBBLE SORT

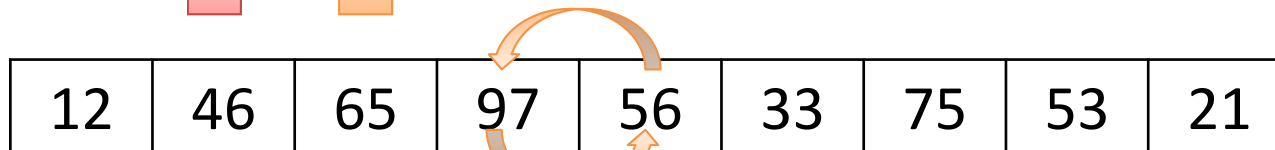
バブルソート

バブルソート(基本交換法)

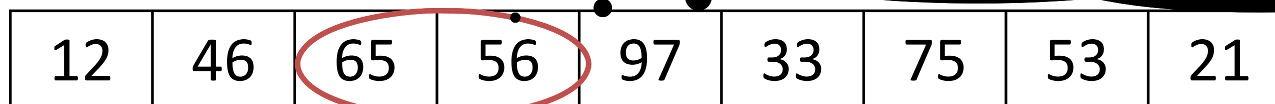
- 左から右にデータを見て、逆順になっているペアがあれば交換する



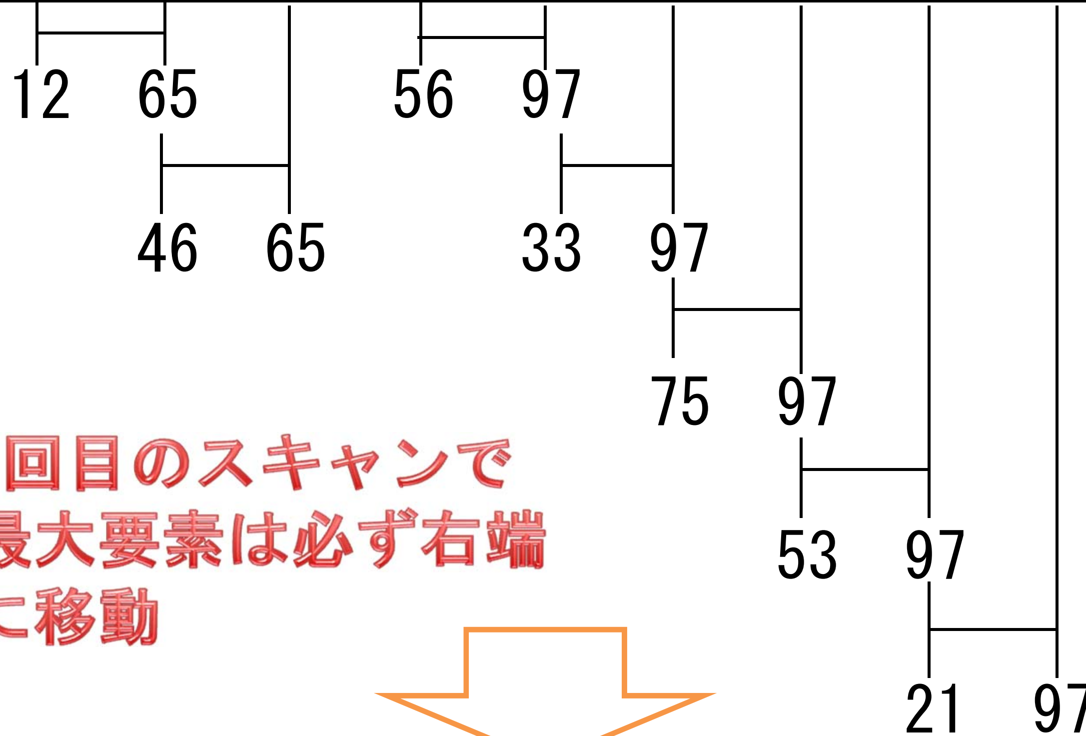


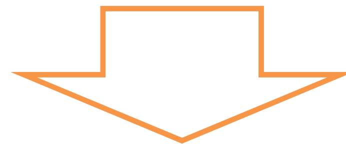


完全にソートされてはいない



バブルソート: 1回目のスキャン



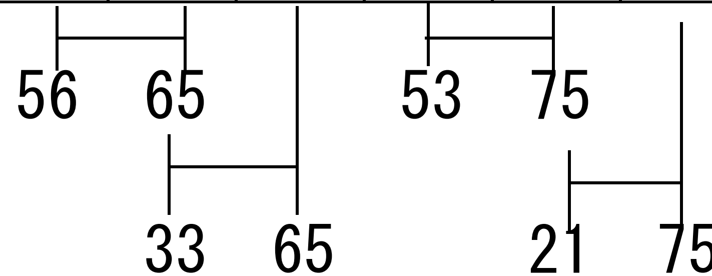


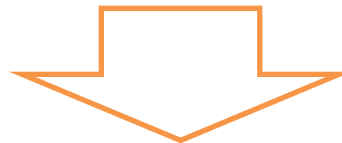
12	46	65	56	33	75	53	21	97

最大値

バブルソート: 2回目のスキャン

12	46	65	56	33	75	53	21	97





ソート済み

12	46	56	33	65	53	21	75	97

ソートしきれていない ➔ 次のスキャンへ

バブルソート: スキャン回数

- k回のスキャンでk個のデータがソートされる
  → 全要素のソートに要するスキャンはn-1回

65	12	46	97	56	33	75	53	21	：	k=0 入力
12	46	65	56	33	75	53	21	97	：	k=1
12	46	56	33	65	53	21	75	97	：	k=2
12	46	33	56	53	21	65	75	97	：	k=3
12	33	46	53	21	56	65	75	97	：	k=4
12	33	46	21	53	56	65	75	97	：	k=5
12	33	21	46	53	56	65	75	97	：	k=6
12	21	33	46	53	56	65	75	97	：	k=7
12	21	33	46	53	56	65	75	97	：	k=8 ソート完了

ソートを完了した部分

バブルソート: 計算量

- プログラム

```
for(k=1; k<n; k=k+1)
   for(i=0; i<n-k; i=i+1)
      if(data[i] > data[i+1])
         swap(&data[i], &data[i+1]);
```

- 比較回数: $\sum_{k=1}^{n-1}(n-k) = n(n-1)/2 \in \Theta(n^2)$
  - ソート済みデータでも n^2 に比例する比較回数
  - 逆順のデータだと，データ交換回数も n^2 に比例

バブルソート: 直接選択法
データ交換の回数を少なくする

- Q. データ交換(swap) を高々n回に改善したい

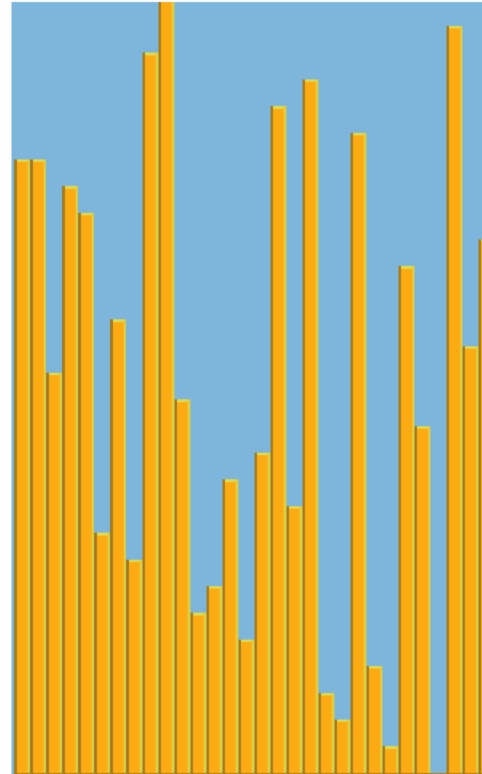
```
for(k=1; k<n; k=k+1)
   for(i=0; i<n-k; i=i+1)
      if(data[i] > data[i+1])
         swap(&data[i], &data[i+1]);
```

- A. 最大値を探して右端のデータと交換する

```
for(k=n-1; k>0; k=k-1){
  m=0;
  for(i=1; i<=k; i=i+1)
    if(data[i] > data[m]) m=i;
  swap(&data[k], &data[m]);
}
```

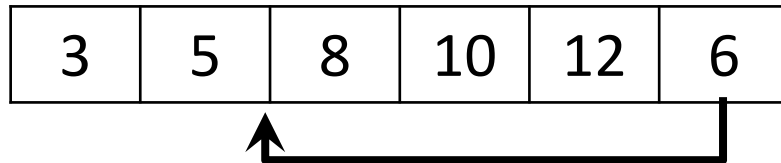
INSERTION SORT

插入法

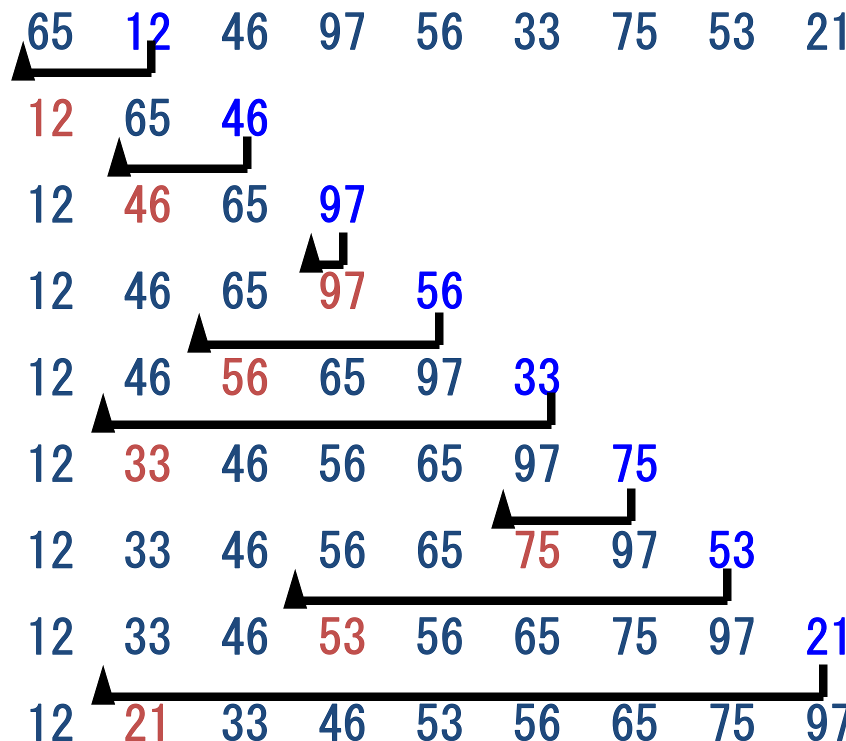


挿入法 (insertion sort)

- 毎回1個の要素をソート列に加えてゆく



挿入すべき要素より大きい要素を右にずらす必要がある




```
for(i=1; i<n; i=i+1){
 x = data[i]; j=i;
 while(data[j-1]>x &&
j>0){
   data[j] = data[j-1];
   j=j-1;
 }
 data[j] = x;
}
```

Q. 挿入場所を求めるのに2分探索法を使わないのは何故?

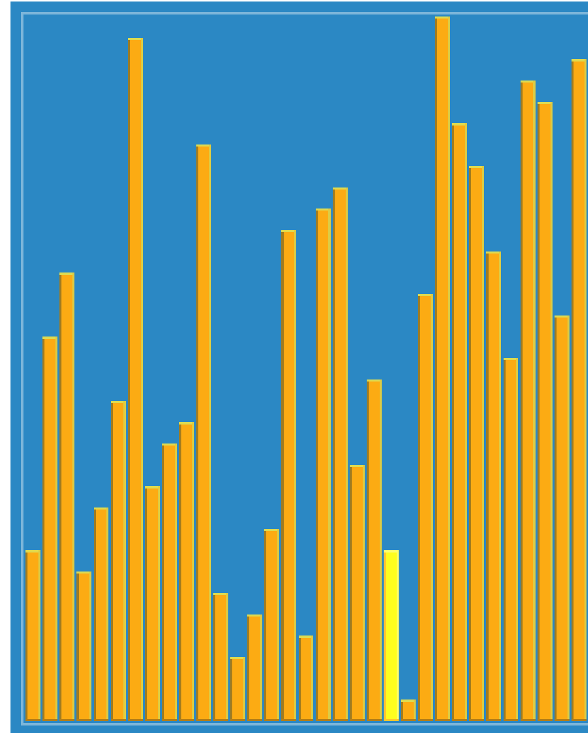
挿入法: 計算時間

- 最良の場合: $\Theta(n)$
  - ソート済みのデータが入力
- 最悪の場合: $\Theta(n^2)$
  - 逆順にソートされたデータが入力
  - 毎回全ての要素と比較して移動
- 平均的な場合: $\Theta(n^2)$
  - 新たに挿入する要素がソート済みのm個の要素の中でk番目に大きい要素のときk回の比較
  - k番目に大きい要素である確率は 1/m

Donald L. Shell
1924−

SHELL SORT

改良挿入法



D.L. Shell, “A high-speed sorting procedure”. Communications of the ACM 2 (7): 30–32 (1959)

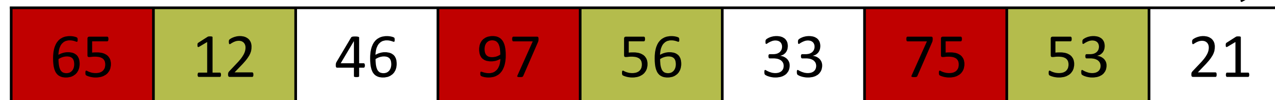
改良挿入法 (shell sort)

- 挿入法の反省
  - 利点: ほぼソートされた列のソートは速い
  - 欠点: あまりソートされていない列に対して遅い

➔ 前処理で“ほぼソートされた”列を作る

- 挿入法に対する改良: h-整列
  h要素分離れた要素の集合を整列させる
  - e.g., 3-整列の場合

同じ色の要素間でソート



改良挿入法: アルゴリズムの概観

1. 適当なhを定める
2. h-整列な列を作る
3. hの値を小さくして
   - h!=1のとき: 2を実行
   - Otherwise: 普通の挿入法を実行

E.g., h の値をn/2, n/4, n/8, ... , 1と変化させる

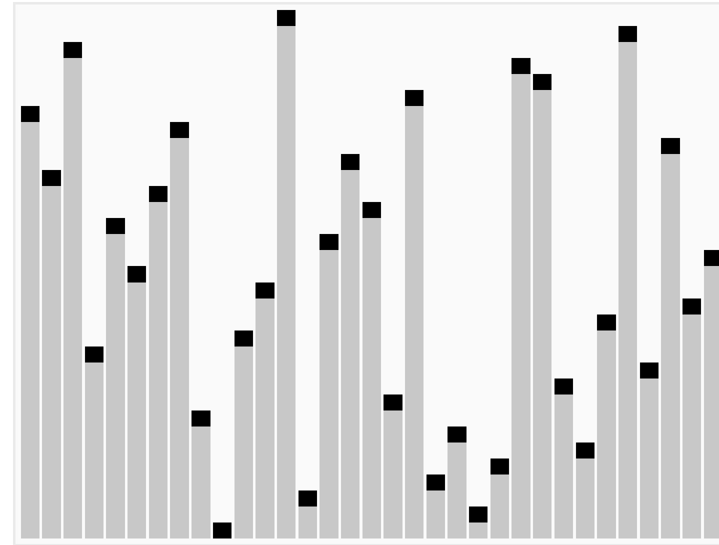
65	12	46	97	56	33	75	53	21	h=4
21	12	46	53	56	33	75	97	65	h=2
21	12	46	33	56	53	65	97	75	h=1
12	21	33	46	53	56	65	75	97	完了

改良挿入法: プログラムと計算量

- プログラム

```
for(gap=n/2; gap>0;gap=gap/2)
  for(i=gap; i<n; i=i+1)
    for(j=i-gap; j>=0 && a[j]>a[j+gap]; j=j-gap)
      swap(&a[j], &a[j+gap]);
```

- 計算量: $O(n^2)$ よりは良い
  - 正確に計算量を見積もるのは難しい
  - ギャップの取り方によって, $\Theta(n \log^2 n)$ にできる



HEAP SORT

ヒープソート

ヒープソート (heap sort)

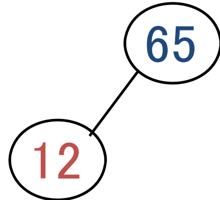
- データ構造 ヒープ
  - データ追加: Θ(log *n*)
  - 最大要素の取り出し: Θ(log *n*)
- ソートの方法
  - Step 1: *n* 個のデータを順にヒープに入れる
  - Step 2: ヒープから最大要素を取り出して，配列の右端から順に格納
- 計算時間: Step 1, 2 ともに Θ(*n* log *n*) 時間

ヒープソート: 実行例@Step 1

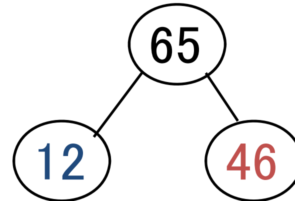
Data = 65 12 46 97 56 33 75 53 21

(1)add 65

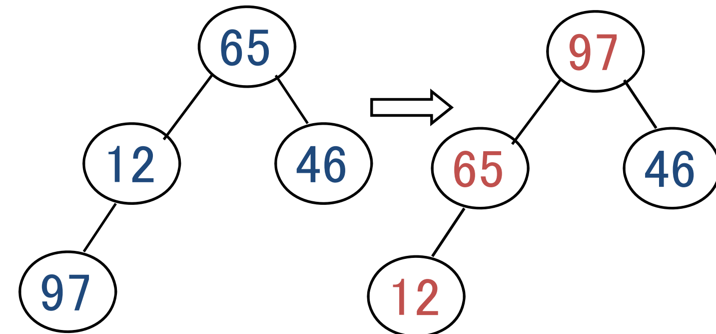
(2)add 12



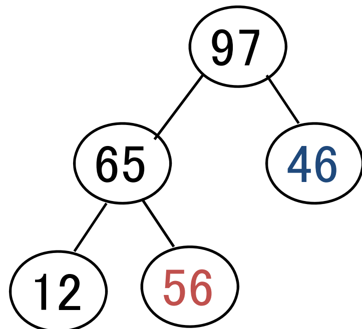
(3)add 46



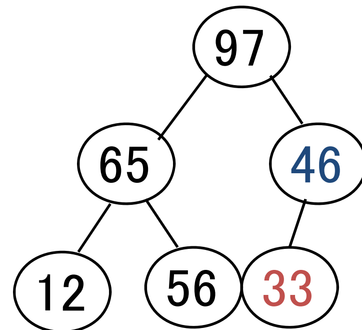
(4)add 97



(5)add 56



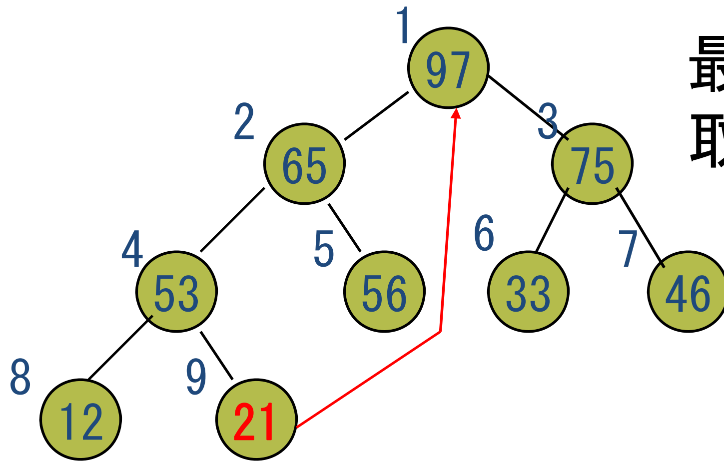
(6)add 33



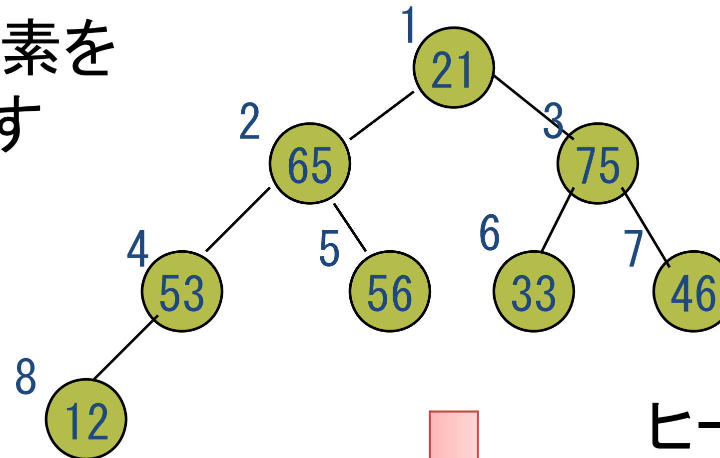
以下，同様にヒープにデータを順に加えていく

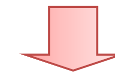
1	2	3	4	5	6	7	8	9
97	65	75	53	56	33	46	12	21

ヒープソート:
データの取り出し@Step 2

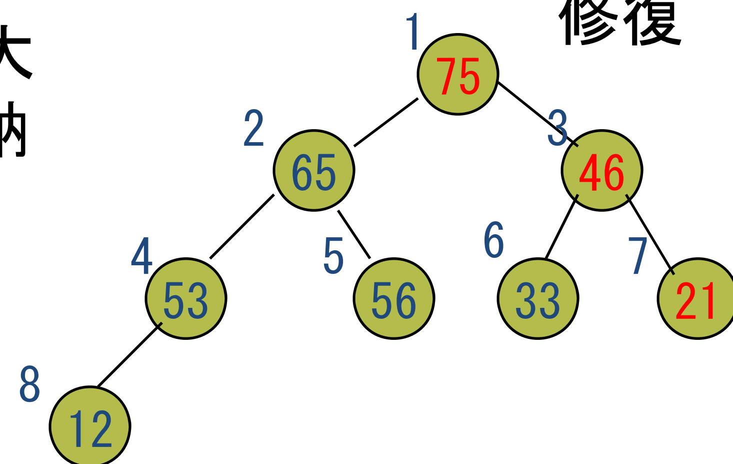


最大要素を
取り出す

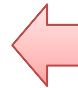




ヒープを
修復



右端に最大
要素を格納



75	65	46	53	56	33	21	12	97

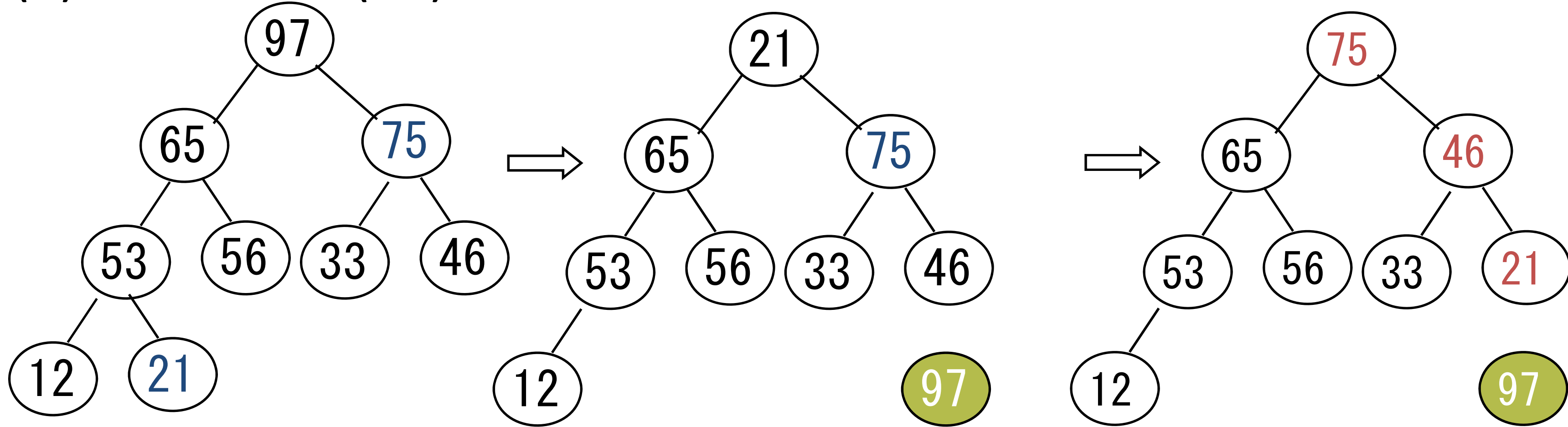
75	65	46	53	56	33	21	12	

ヒープソート: 実行例@Step 2

配列=

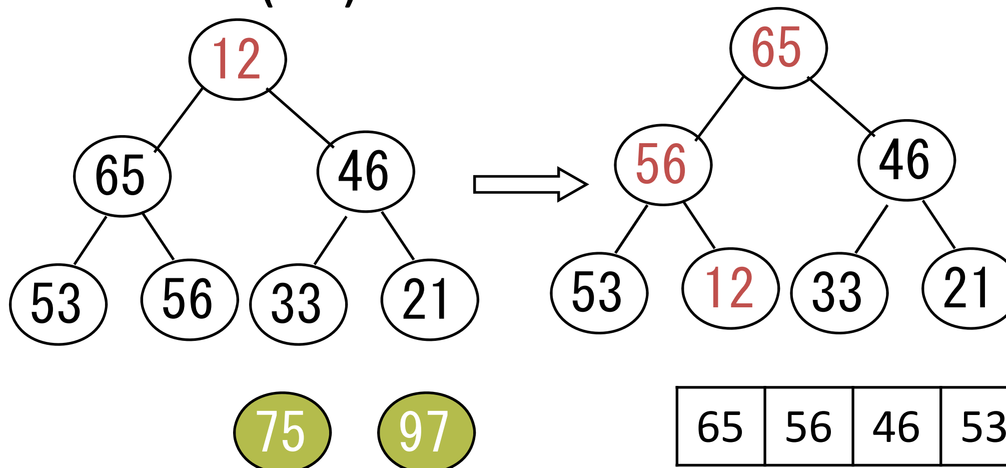
97	65	75	53	56	33	46	12	21

(1)delete max (97)



75	65	46	53	56	33	21	12	97

(2) delete max (75)

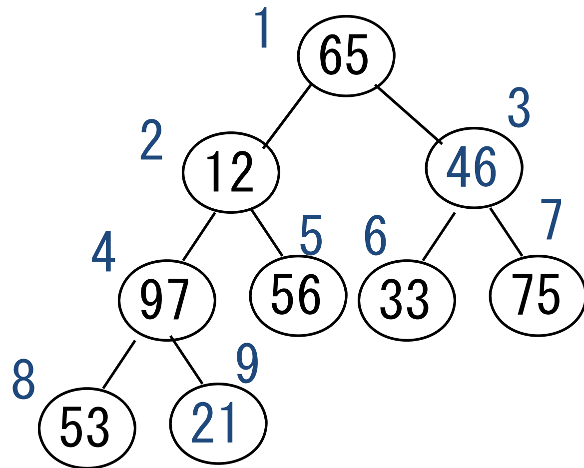


65	56	46	53	12	33	21	75	97

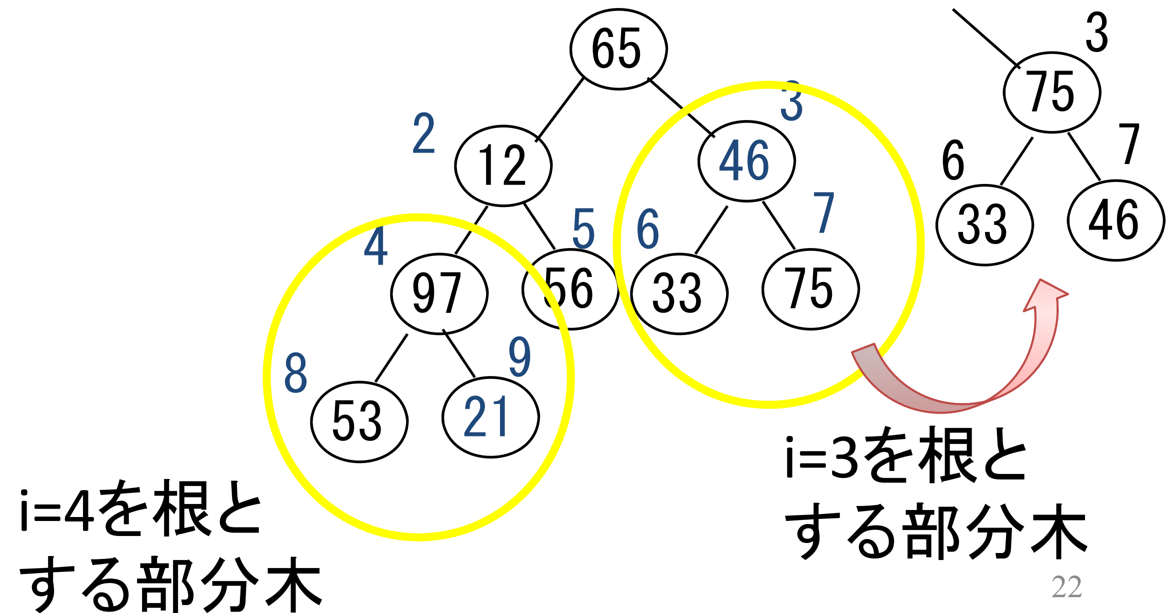
ヒープソートの改善

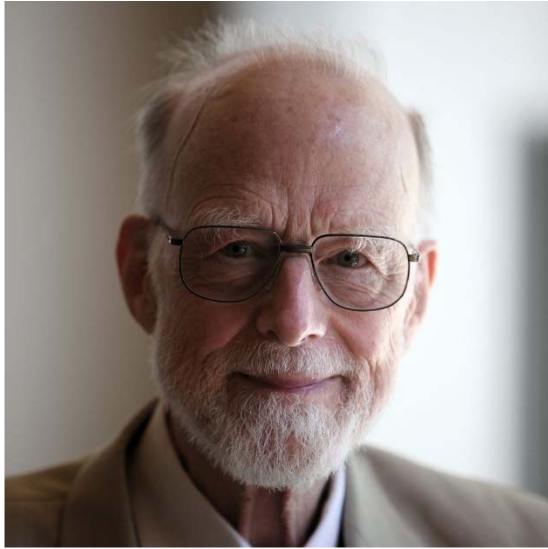
- Step 1 は，Θ(n) にできる
  - 与えられた順に配列にデータを格納する
  - 下から順に，親子節点のデータを交換

(1)データを格納



(2)親子関係の節点のデータを交換

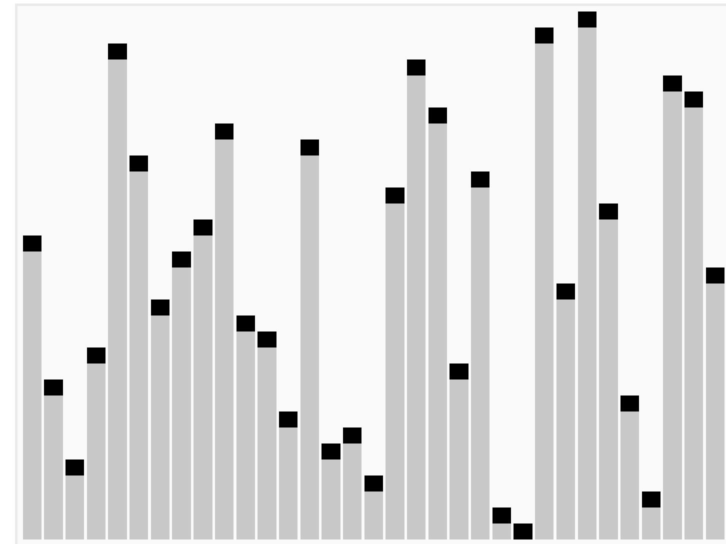




Tony Hoare
1934–

QUICK SORT

クイックソート



C.A.R. Hoare, "Algorithm 64: Quicksort". Communications of the ACM 4 (7): 321 (1961)

クイックソート (quick sort)

- 特徴: 平均的に最も速い
- ソートの仕方:
  - Step 1: 配列中の任意の要素xを選ぶ
  - Step 2: x 未満の要素を配列の左から
    x 以上の要素を配列の右から詰める

  - Step 3: x未満の列およびx以上の列を再帰的にソート
    - 列が十分に短くなったら素朴なソート

クイックソート: 実行例
Step 1. 任意の要素xを選ぶ

- 以下の配列のソートを考える

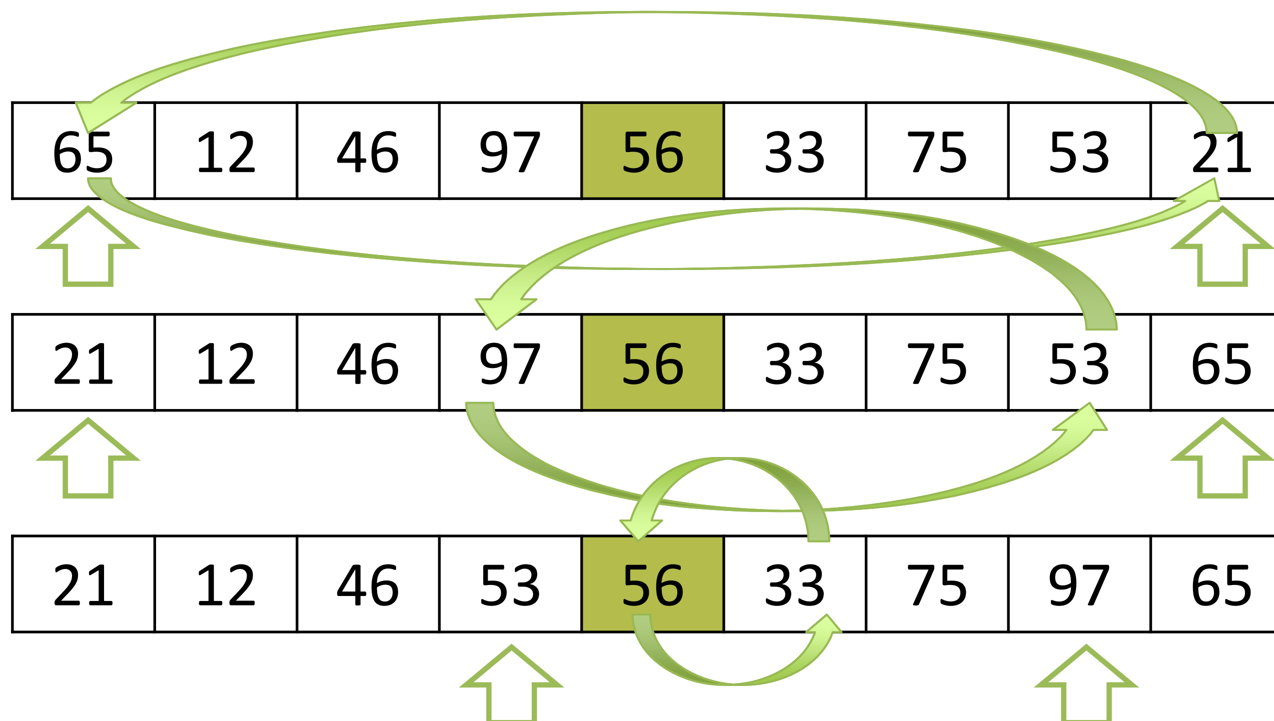
65	12	46	97	56	33	75	53	21

- x=56を選んだとする

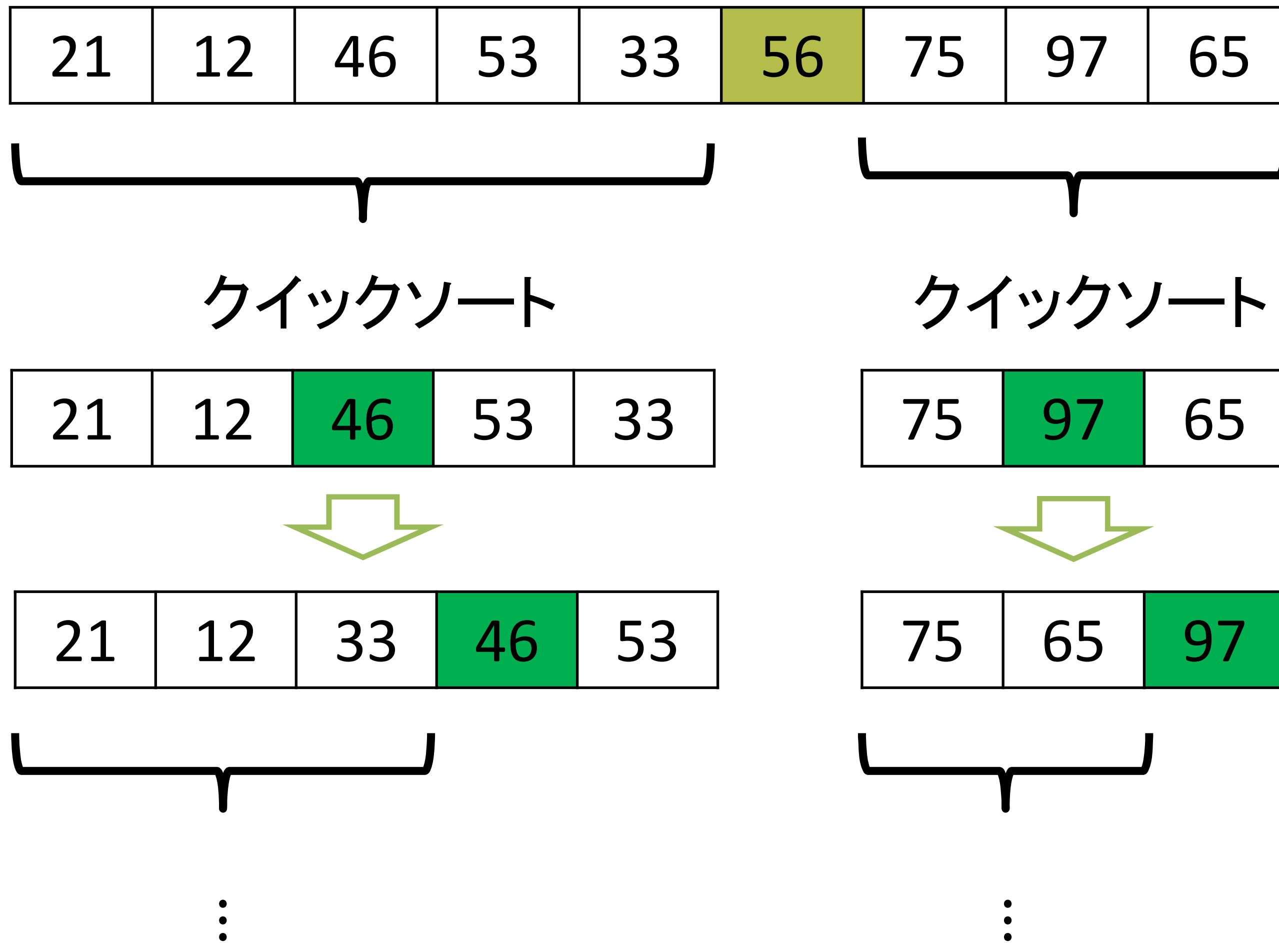
65	12	46	97	56	33	75	53	21

クイックソート: 実行例
Step 2. x未満とx以上の列にわける

- x未満 | x以上
- [l, r] = [0,n-1] から始めて，l, r を動かし，
  a[l] >= x && a[r] < x ⇒ a[l] と a[r] を交換



クイックソート: 実行例
Step 3. x未満とx以上の列をソート



クイックソート: プログラム

```
qsort(int a[], int left, int right){
  int i, j, x;
  if(right <= left) return;
  i = left; j = right; x = a[(i+j)/2];
  while(i<=j){
    while(a[i]<x) i=i+1;
      while(a[j]>x) j=j-1;
       if(i<=j){
      swap(&a[i], &a[j]); i=i+1; j=j-1;
    }
  }
  qsort(a, left, j); qsort(a, i, right);
}
```

クイックソート: 計算時間(1/2)
最悪の場合

- 毎回 x として，列の最大値/最小値を選ぶ
長さ n の列 → 長さ 1 の列 : 長さ n-1 の列
- 長い方の列が長さ2になるまでの繰り返し
- よって比較回数は $\sum_{k=2}^{n} k \in \Theta(n^2)$

クイックソート: 計算時間(2/2) 平均的な場合

- n 個の要素の中から任意の要素 x を選ぶとき,
  x が列の中で k 番目の要素である確率: 1/n
- x が k 番目の要素:
  長さ n の列 → 長さ k の列 : 長さ n-k の列
- 全体の比較回数

$$C(n) = \sum_{k=1}^{n} \frac{1}{n}(n + C(k) + C(n-k))$$

$$\approx n + \frac{2}{n}\sum_{k=1}^{n} C(k)$$

$$\Longrightarrow \; C(n) = 1.36n\log n + O(n)$$

ミニ演習

- 前述の qsort に対して，計算時間が最悪となる入力を作れ（入力の長さは10程度）