実践的アルゴリズム理論
Theory of Advanced Algorithms

分割統治法と漸化式

担当:上原隆平

Theory of Advanced Algorithms
実践的アルゴリズム理論

Divide and Conquer and Recurrence Equation

Ryuhei Uehara

分割統治法

問題を幾つかの部分問題に分解して，それぞれの部分問題を再帰的に解いた後，部分問題の解を利用して元の問題を解く.

分割：問題をいくつかの部分問題に分割する（2分割など）.
統治：部分問題を再帰的に解く．ただし，部分問題のサイズが小さいときは直接的に解く.
統合：部分問題の解を統合して全体の解を構成する.

プログラムは次のような形式：

```
function DQ(x){
  if 問題 x が十分に小さい then 特別の方法を適用して答を返す;
  問題 x を幾つかの部分問題 x_1, x_2, ... , x_k に分割;
  for i=1 to k
    y_i = DQ(x_i); //部分問題 x_i の解 y_i を再帰的に求める;
  部分問題の解 y_1, y_2, ... , y_k を組み合わせて元の問題 x の
  解 y を得て，y を返す;
}
```

Divide and Conquer

Decompose the problem into several subproblems, solve them recursively, and then find a solution to the original problem by combining the solutions to the subproblems.

Divide: Decompose the problem into several subproblems.

Conquer: Solve the subproblems recursively. If they are small enough, we solve them directly.

Merge: Combine those solutions to have a solution to the problem.

Program is of the following form:

```
function DQ(x){
  if problem x is small enough then apply an adhoc procedure;
  decompose x into several subproblems x_1, x_2, ... , x_k ;
  for i=1 to k
    y_i = DQ(x_i); //solve x_i recursively;
  combine the solutions y_1, y_2, ... , y_k to obtain a solution y to the
  original problem x and return y;
}
```

問題P7：配列に蓄えられたデータの最大値を求めよ.

最も単純なアルゴリズムは，順に配列要素を調べて，以前に求まっている最大値より大きい要素を見つければ最大値を更新するというもの．--->アルゴリズムP7-A0.

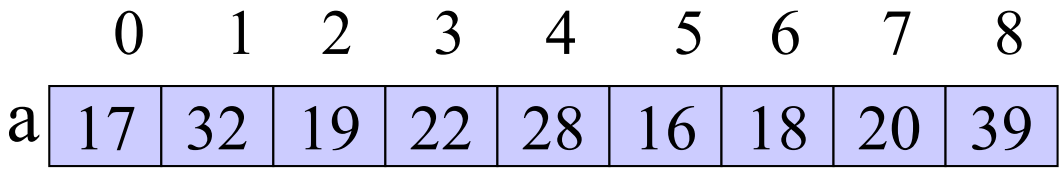
	0	1	2	3	4	5	6	7	8
a	17	32	19	22	28	16	18	20	39

アルゴリズムP7-A0:

```
max=a[0];
for(i=1; i<n; i++)
   if(a[i] > max) max = a[i];
cout << max;
```

Problem P7 : Find a largest value in an array.

In the most simple algorithm we check all the elements and if we find a larger one than the largest one so far then we update the largest value.--->algorithm P7-A0.




```
Algorithm P7-A0:
max=a[0];
for(i=1; i<n; i++)
   if(a[i] > max) max = a[i];
cout << max;
```

分割統治法に基づく最大値発見

分割：配列を左半分と右半分に分割.
統治：配列のサイズが1になれば，その配列要素を出力.
統合：左右の部分の最大値のうち大きい方を出力.

アルゴリズムP7-A1:

```
int FindMax(int left, int right){
   if(right==left) return a[left];
   int mid = (left+right)/2;
   int x1 = FindMax(left, mid);
   int x2 = FindMax(mid+1, right);
   if(x1>x2) return x1; else return x2;
}
main(){
   ......
   cout << FindMax(0, n-1);
}
```

サイズnの配列で最大値を求めるのに要する時間をT(n)とすると，

$T(1) = 1.$

$T(n) \leq 2T(n/2)+3.$

この漸化式を解くと

$T(n) = O(n).$

練習問題：上の漸化式を解け.

Finding Maximum based on Divide and Conquer

Divide: Decompose array array into two halves

Conquer: If the array size is 1, output the element.

Merge: Output the larger one of the two maximums.

Algorithm P7-A1:

```
int FindMax(int left, int right){
   if(right==left) return a[left];
   int mid = (left+right)/2;
   int x1 = FindMax(left, mid);
   int x2 = FindMax(mid+1, right);
   if(x1>x2) return x1; else return x2;
}
main(){
   ......
   cout << FindMax(0, n-1);
}
```

Let T(n) be time to find a maximum among n data, then we have

$T(1) = 1$.

$T(n) \leq 2T(n/2)+3$.

Solving it, we have

$T(n) = O(n)$.

Exercise: Solve the above recurrence equation.

分割統治法に基づく最大値発見

配列を１個と残り全部に分けるとどうか?

アルゴリズムP7-A2:

```
int FindMax(int left, int right){
   if(right==left) return a[left];
   int x1 = FindMax(left, left);
   int x2 = FindMax(left+1, right);
   if(x1>x2) return x1; else return x2;
}
main(){
   ......
   cout << FindMax(0, n-1);
}
```

サイズnの配列で最大値を求めるのに要する時間をT(n)とすると,

$T(1) = 1.$

$T(n) \leqq T(1)+T(n-1)+2.$

この漸化式を解くと

$T(n) = O(n).$

練習問題:上の漸化式を解け.

つまり，最大値発見のような簡単な問題であれば，どちらも空でないように分割すれば，どのようにしても効率は殆んど同じ.

Finding maximum based on divide and conquer

What happens if we decompose an array into one and remaining?

Algorithm P7-A2:

```
int FindMax(int left, int right){
   if(right==left) return a[left];
   int x1 = FindMax(left, left);
   int x2 = FindMax(left+1, right);
   if(x1>x2) return x1; else return x2;
}
main(){
   ......
   cout << FindMax(0, n-1);
}
```

Let T(n) be time to find a maximum among n data, we have

$T(1) = 1$.

$T(n) \leqq T(1)+T(n-1)+2$.

Solving it, we have

$T(n) = O(n)$.

Exercise : Solve the above recurrence equation.

That is, for a simple problem of finding a maximum, the efficiency is almost the same if we decompose an array two non-empty parts.

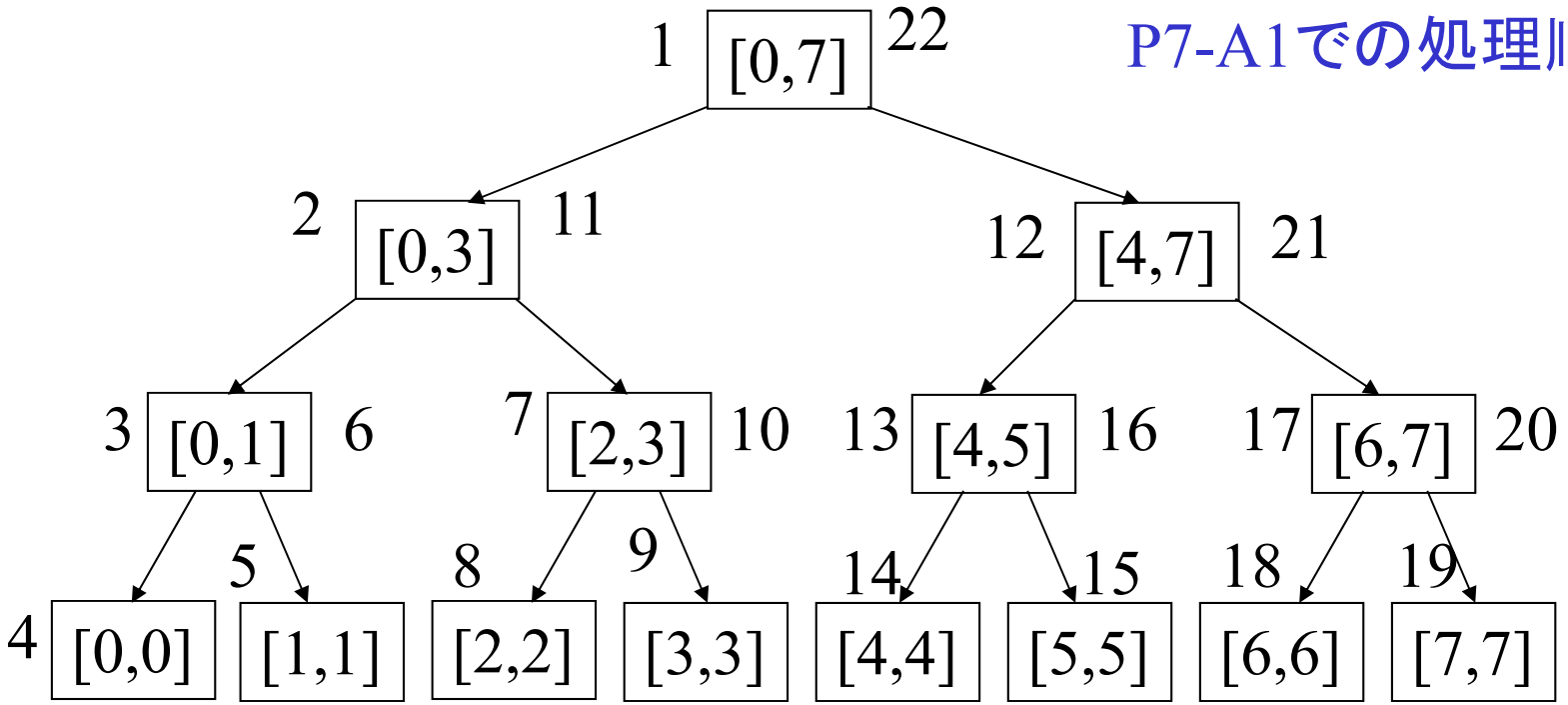
最大値を求めるアルゴリズム

P7-A1:配列を毎回ほぼ等しいサイズに分割.

P7-A2:配列を１個と残り全部に分割.

どの方法も計算時間はO(n)で同じ.

何か違いはあるか？



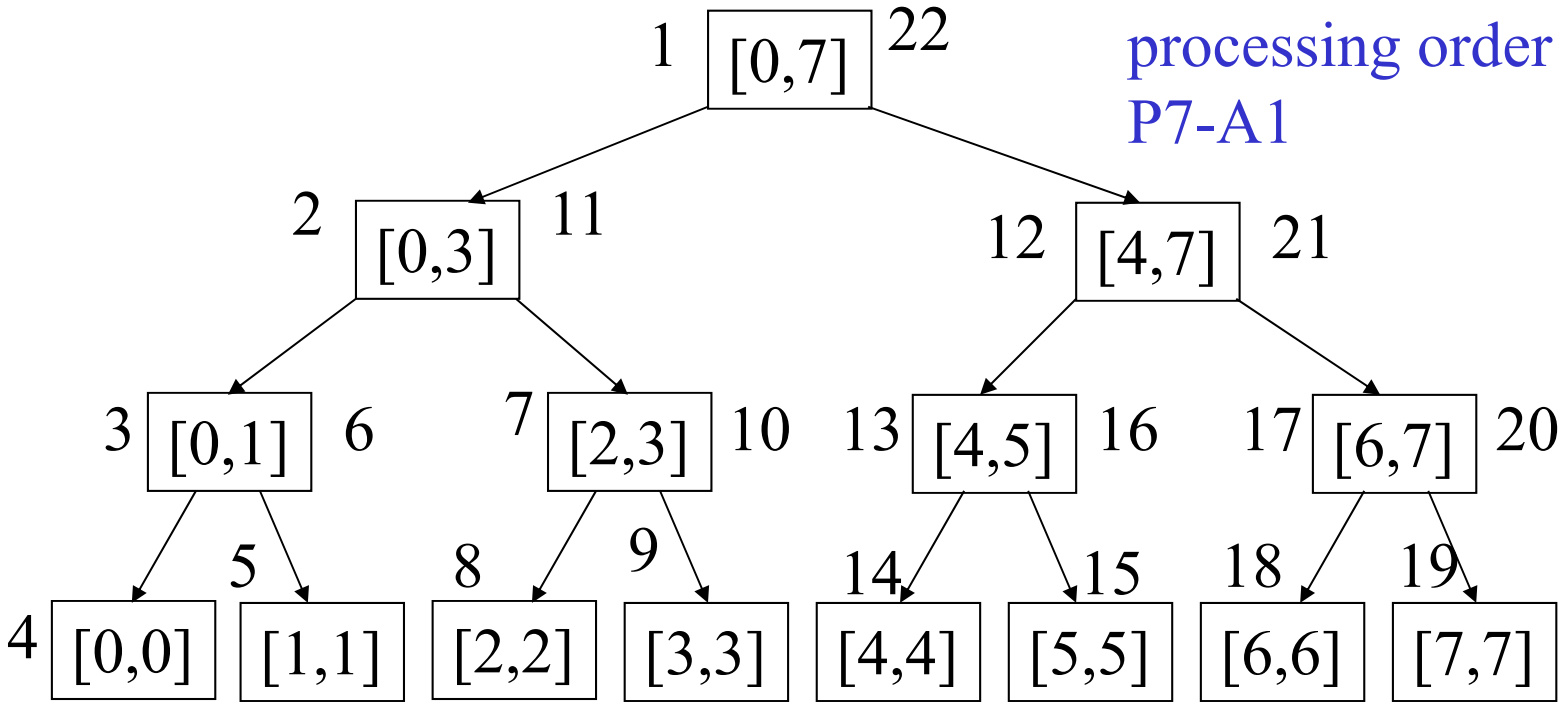
Algorithm for finding a maximum

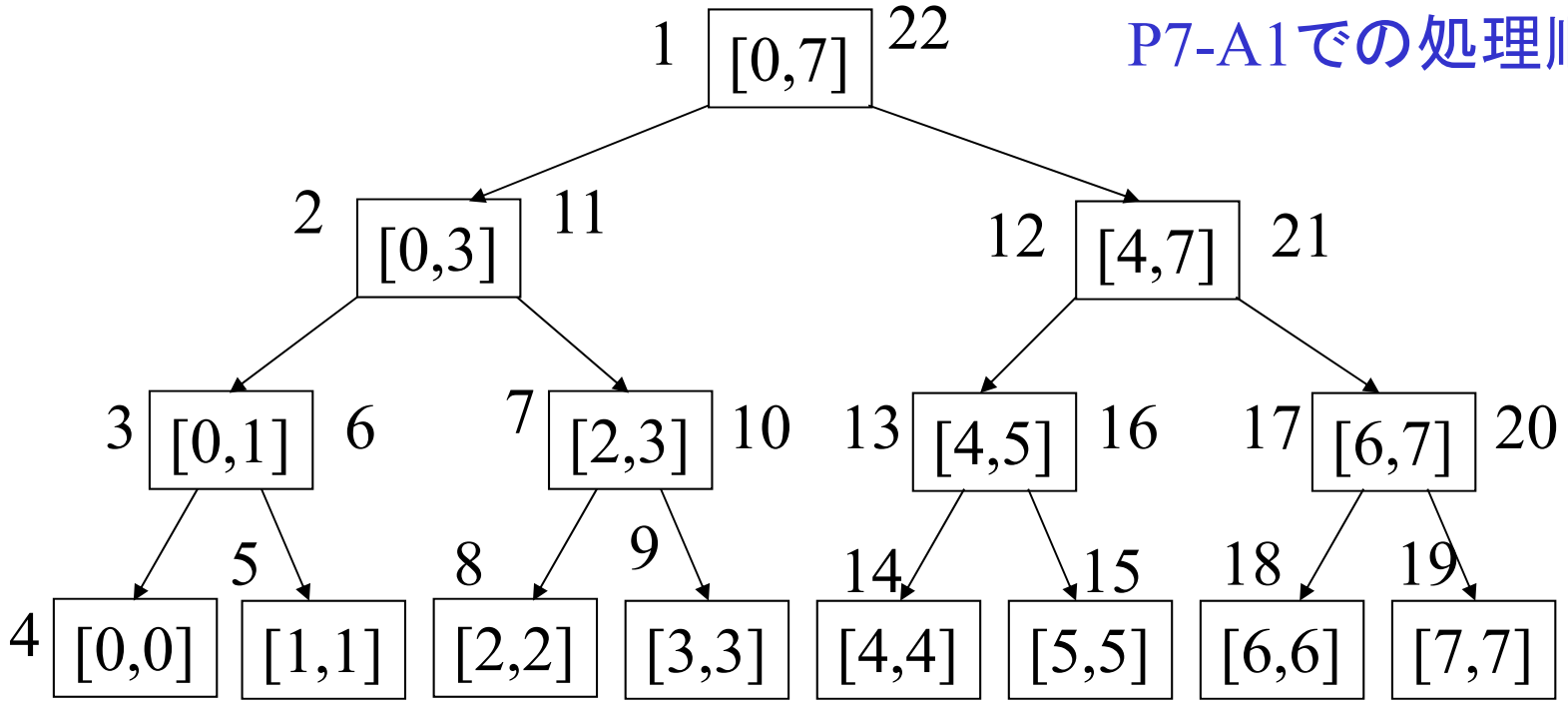
P7-A1:decompose an array into two parts of the same length.

P7-A2:decompose an array into one and the remaining.

Both algorithms run in O(n) time.

Any difference between them?





P7-A1での処理順

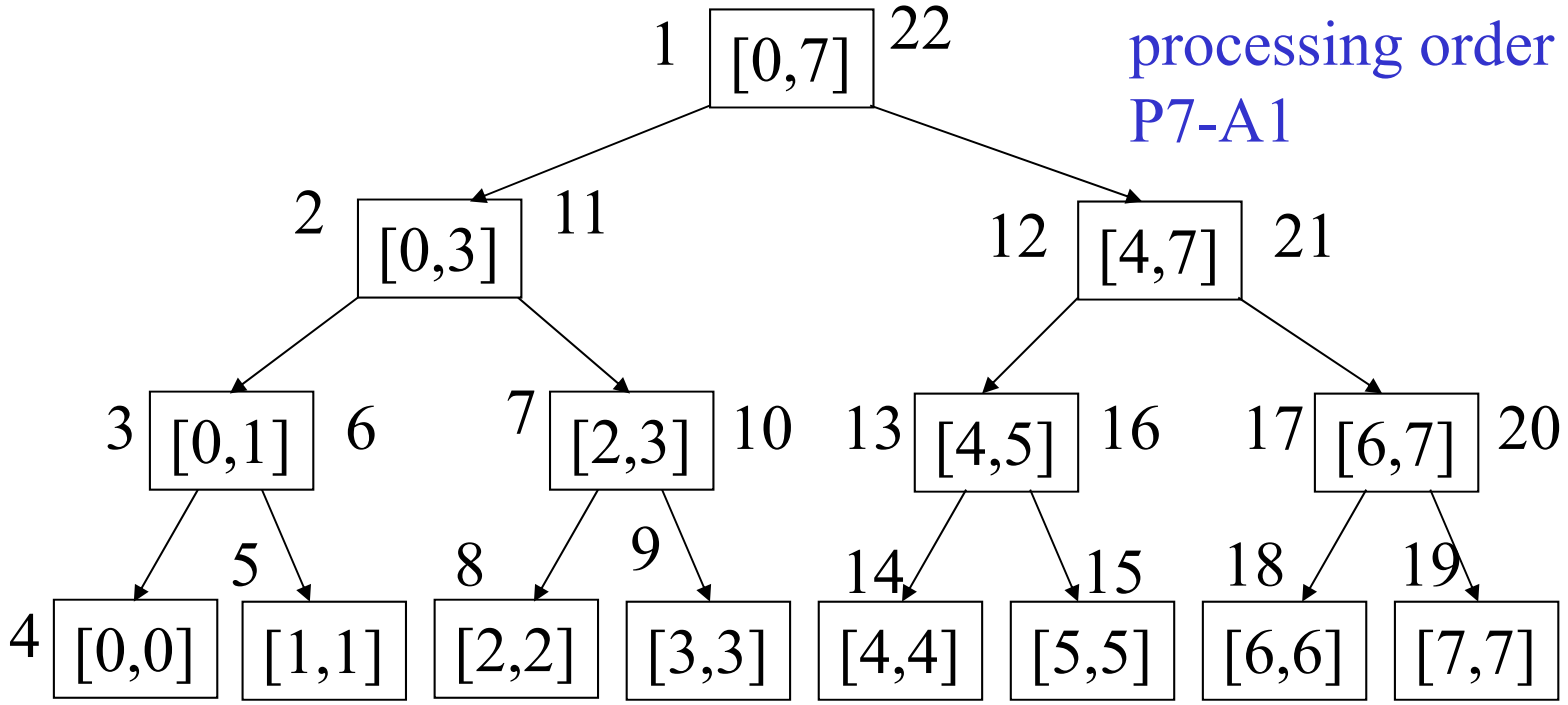
区間[p,q]を処理するとき，その戻り道を記憶しておく必要がある．

例：[3,3]の場合には，[2,3],[0,3],[0,7]

=>木の深さに対応

配列のサイズをnとするとき，木の深さは $\log_2 n$

よって，必要な記憶量は$O(\log_2 n)$.



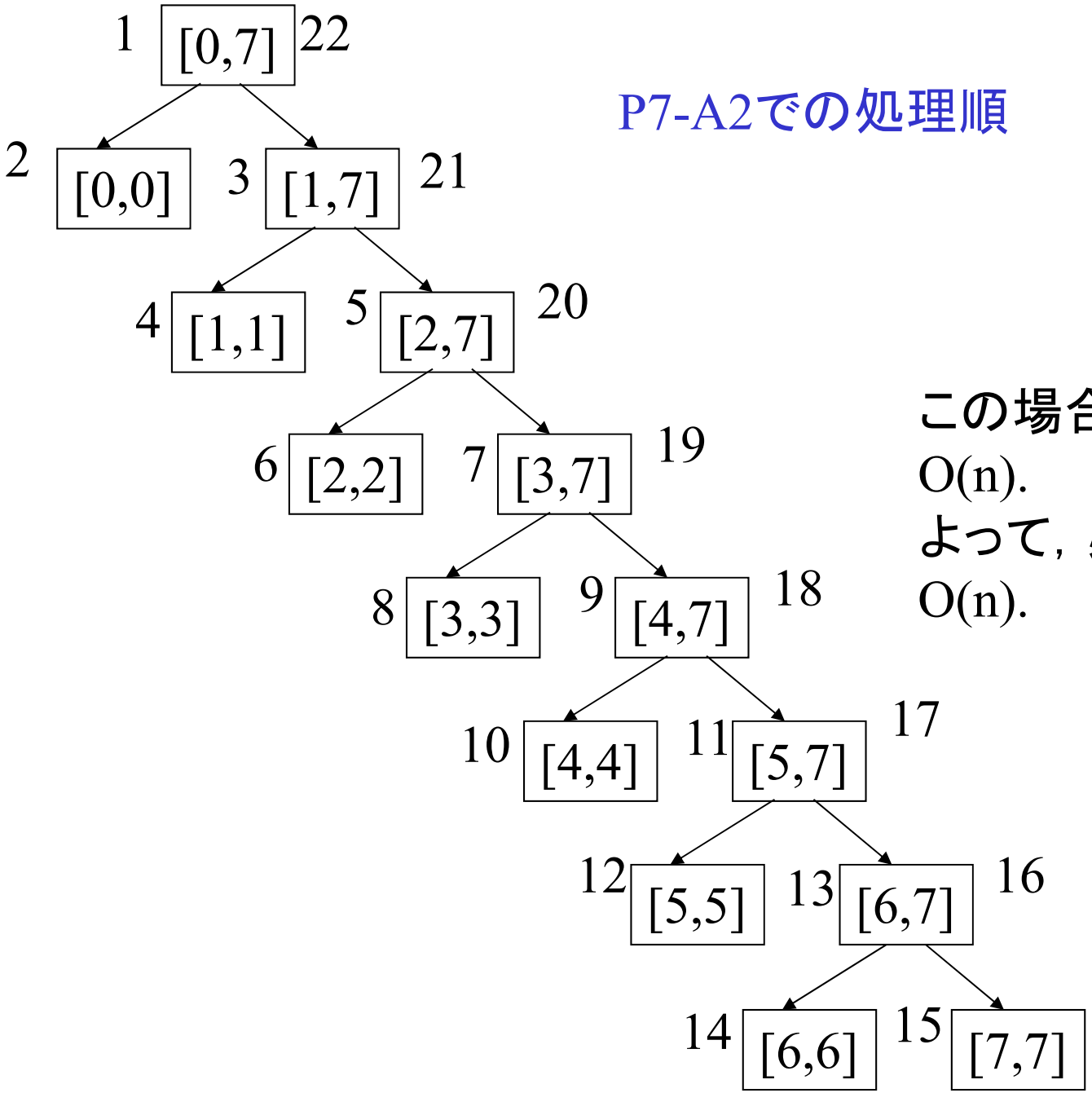
To process an interval [p,q] we need to keep its back path.

Example : for [3,3] we remember [2,3],[0,3], and [0,7]

=>depth of a tree

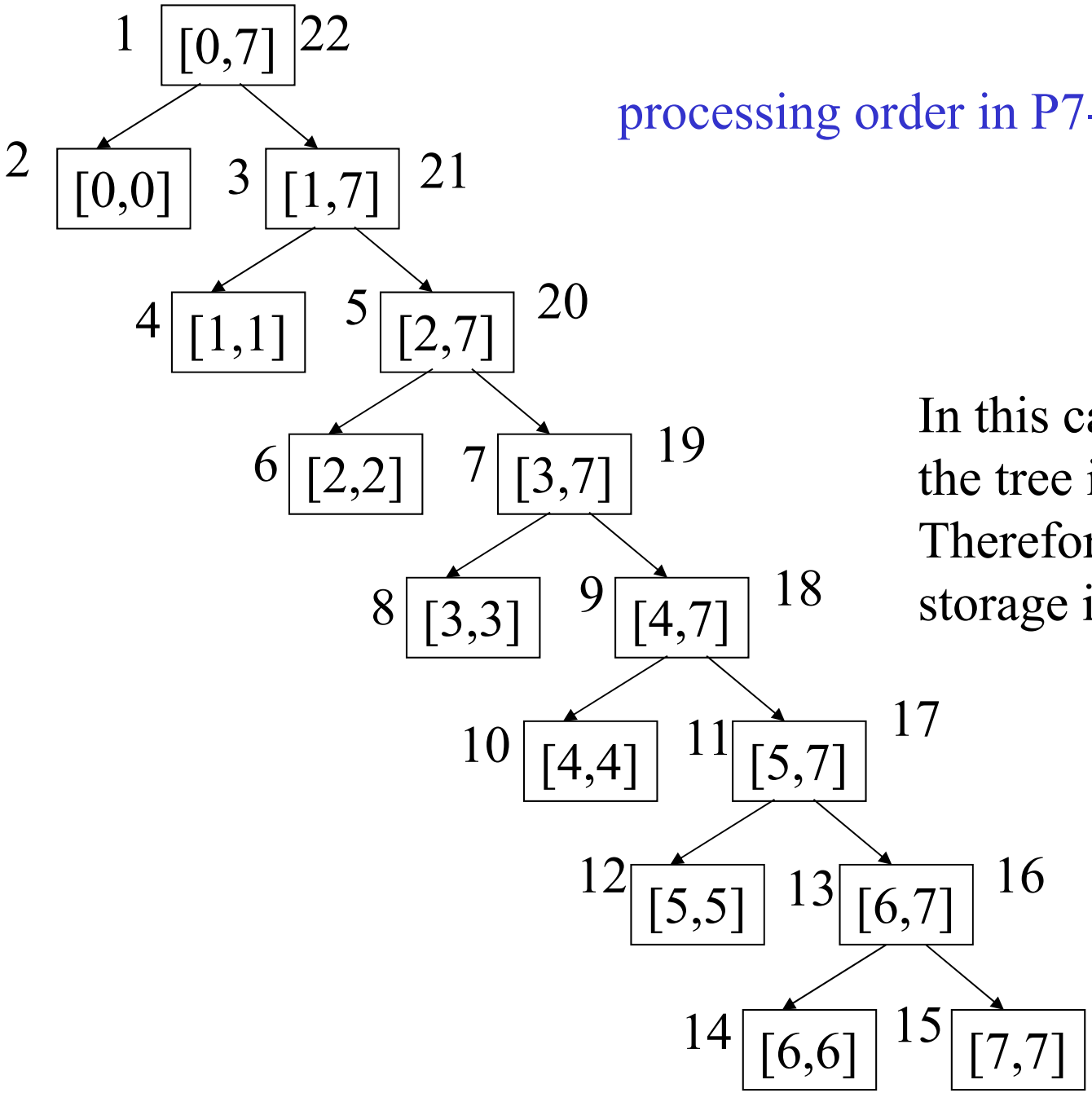
When the size of an array is n, the depth of a tree is $\log_2 n$.

Thus, the required storage is $O(\log_2 n)$.



P7-A2での処理順

この場合，木の深さは
O(n).
よって，必要な記憶量も
O(n).

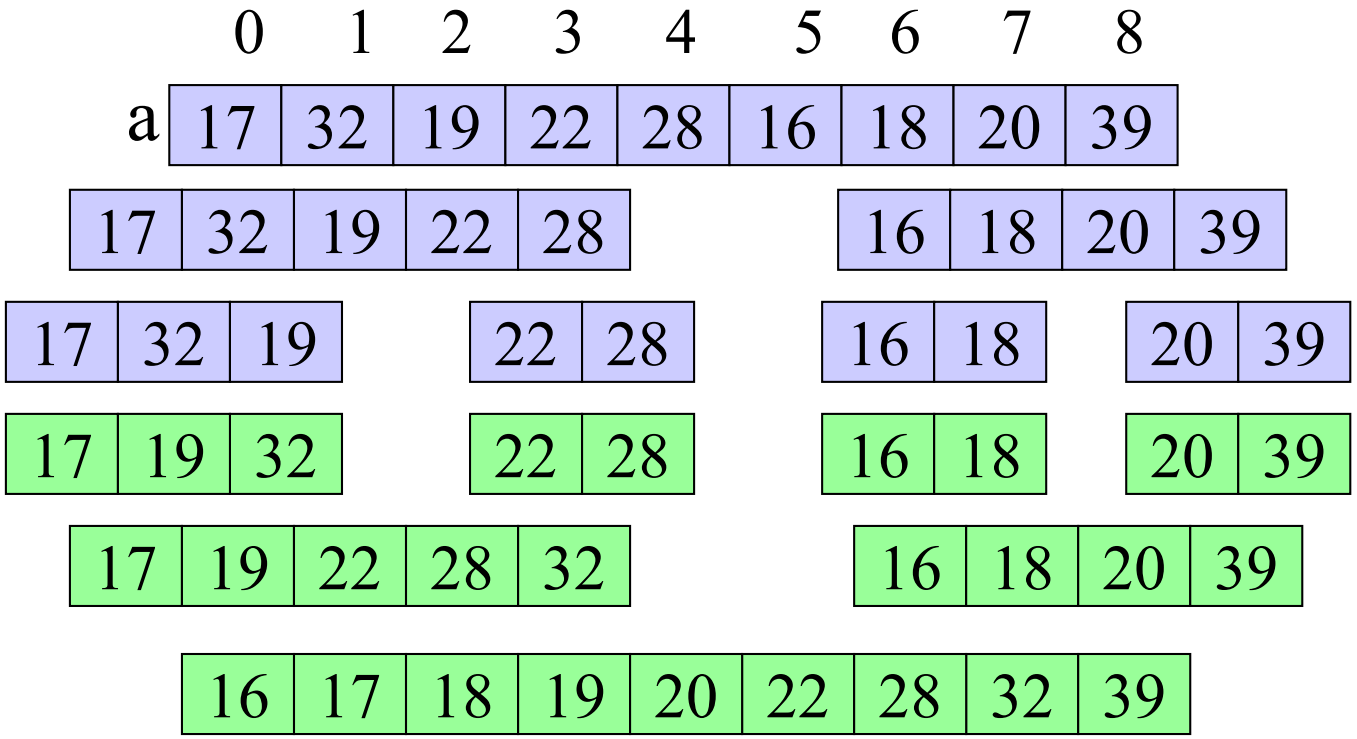


processing order in P7-A2

In this case the depth of the tree is O(n). Therefore, the required storage is O(n).

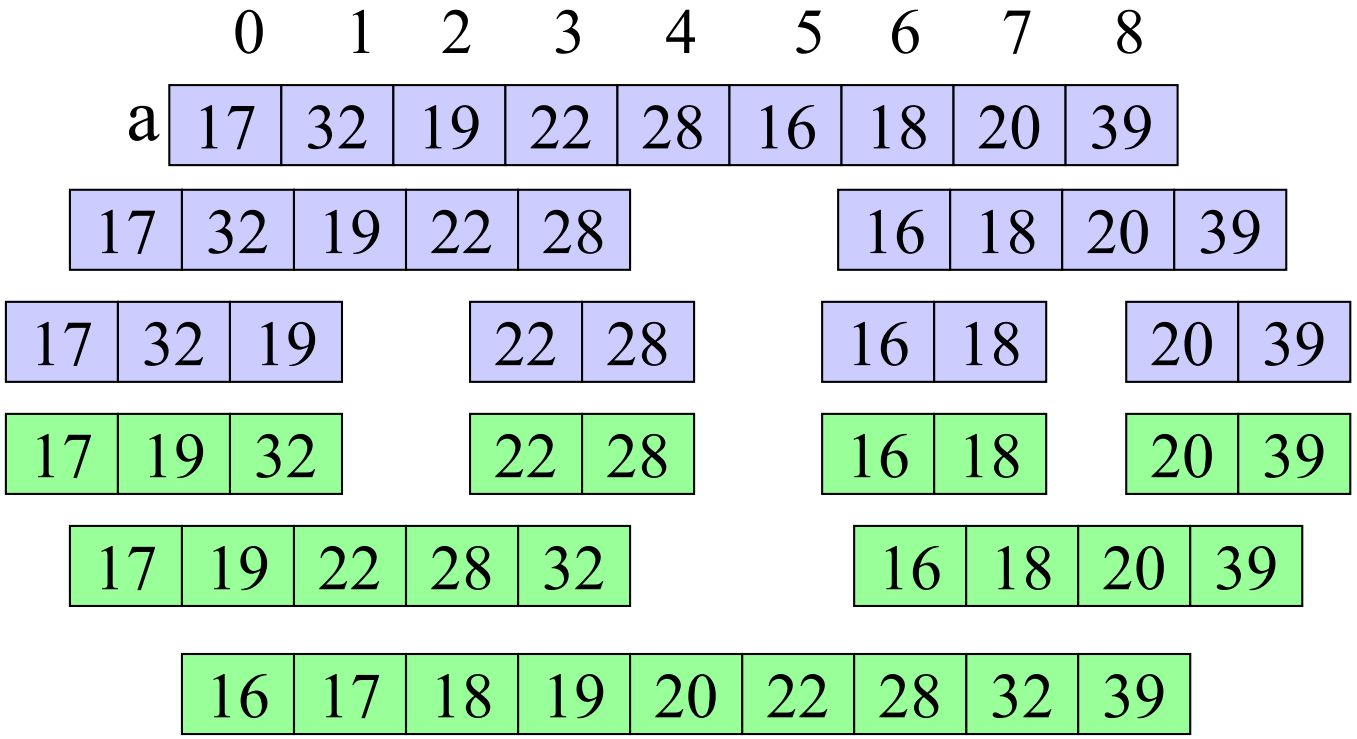
問題P8:（マージソート）
配列に蓄えられた n 個のデータをソートせよ.

分割統治法の考え方に基づいてデータをソート可能.
分割:配列を左半分と右半分に分割.
統治:それぞれの部分を再帰的にソート.
統合:得られた2つのソート列をマージして一つのソート列を得る.



Problem P8:（Mergesort）
Sort n data stored in an array.

We can sort data based on divide and conquer.
Divide : decompose the array into two halves.
Conquer : Sort each half recursively.
Merge : Merge two sorted lists into a sorted list.



計算時間の解析

n 個のデータをマージソートでソートするのに要する時間（比較回数）をT(n)と書くことにする.
半分のサイズの問題を2回解いて，最後に2つのソート列をマージするが，マージは線形時間でできるから cn と置くと，

$$T(n) \leqq 2T(n/2) + cn$$

を得る. これを解けばよい.

$$T(n) \leqq 2T(n/2) + cn$$
$$\leqq 2(2T(n/2^2)+c(n/2))+cn= 2^2T(n/2^2)+2cn$$
$$\leqq 2^2\,(2T(n/2^3)+c(n/2^2))+2cn= 2^3T(n/2^3)+3cn$$
$$\leqq \ldots \leqq 2^kT(n/2^k)+kcn$$

ここで，$2^k=n$, T(1)=定数d，とすると，k=log nだから，

$$T(n) \leqq dn + cn \log n = O(n \log n)$$

を得る.

Analysis of Computation Time

Let T(n) be time (# of comparisons) for sorting n data by Mergesort. We solve the half-sized problems twice and then sort two sorted lists. Since merge operation is done in linear time, let it be cn. Then, we have

$$T(n) \leqq 2T(n/2) + cn,$$

Solving it,

$$T(n) \leqq 2T(n/2) + cn$$
$$\leqq 2(2T(n/2^2)+c(n/2))+cn= 2^2T(n/2^2)+2cn$$
$$\leqq 2^2\,(2T(n/2^3)+c(n/2^2))+2cn= 2^3T(n/2^3)+3cn$$
$$\leqq \ldots \leqq 2^kT(n/2^k)+kcn.$$

If we assume $2^k=n$, $T(1)=$a constant d, then we have $k=\log n$ and

$$T(n) \leqq dn + cn \log n = O(n \log n).$$

問題P9:（中央値選択）
配列に蓄えられた n 個のデータの中央値を求めよ.

	0	1	2	3	4	5	6	7	8
a	17	32	19	22	28	16	18	20	39

昇順に並べると，16,17,18,19,20,22,28,32,39
なので，中央値は20.
nが偶数なら，中央値は2つある.

一般には，n 個のデータの中の k 番目に大きいものを求める問題.

アルゴリズムP9-A0:
n 個のデータをO(n log n)時間でソート.
k 番目のデータを出力.

このアルゴリズムで正しく k 番目の要素が求まる.
しかし，O(n log n)の時間が必要だろうか？

Problem P9: (Median finding)
Find a median of n data stored in an array.

	0	1	2	3	4	5	6	7	8
a	17	32	19	22	28	16	18	20	39

increasing order: 16,17,18,19,20,22,28,32,39
thus, the median is 20.
Note that is n is even then there are two medians.

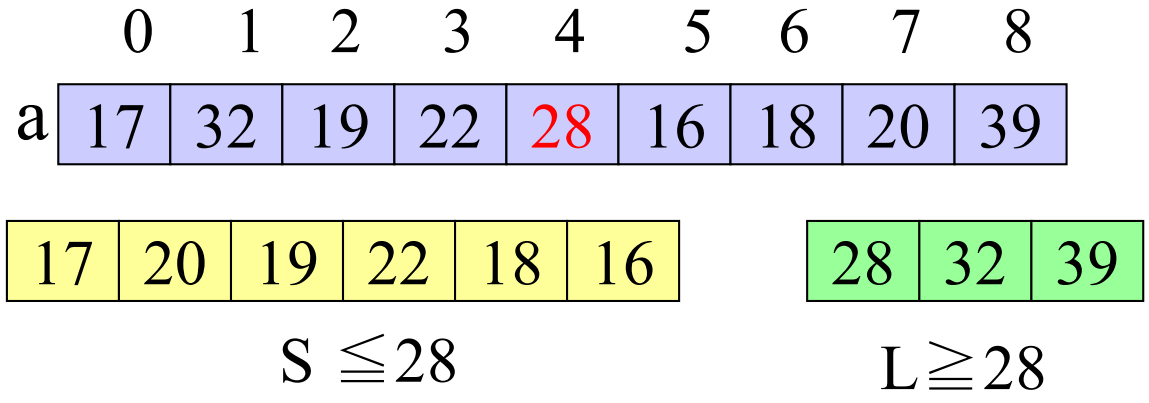
General problem is to find the k-th largest element among n data.

Algorithm P9-A0:
Sort n data in O(n log n) time.
Output the k-th element.

This algorithm always finds the k-th largest element.
But, does it really need O(n log n) time?

分割統治法に基づく方法

アルゴリズムP9-A1:
n 個のデータを配列a[]に蓄える.
一つの配列要素 x を適当に選び，配列を順に調べて，
　　x 以下の要素の集合Sと，x以上の要素の集合Lに分ける.
if k ≦ |L| then
　　集合Lの中でk番目に大きい要素yを再帰的に求める.
else
　　集合Sの中でk-|L|番目に大きい要素yを再帰的に求める.
yを出力する.



Algorithm based on Divide and Conquer

Algorithm P9-A1:

Store n data in an array a[].

Choose an arbitrary element x and decompose n data into

a set S smaller or equal to x and a set L larger or equal to x.

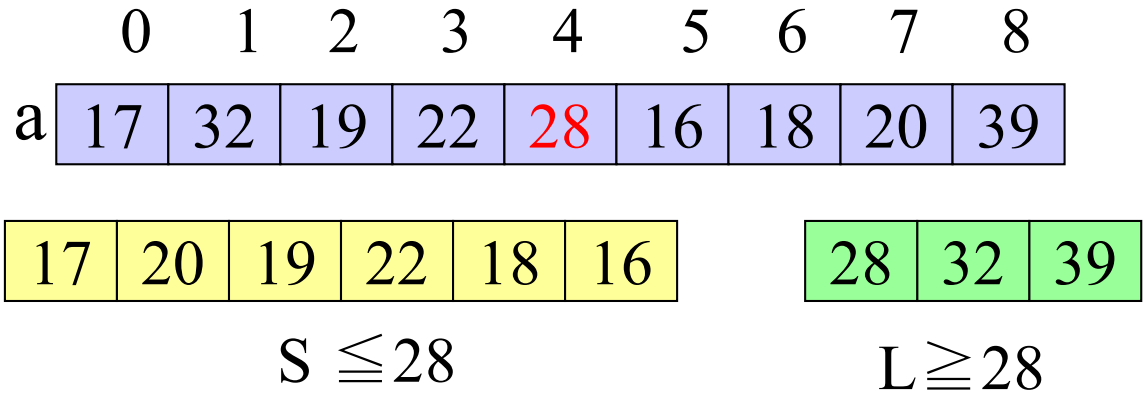
if k $\leqq$ |L| then

recursively find the k-th largest element in the set L.

else

recursively find the (k-|L|)-th largest element in the set S.

Output y.



プログラム例

```
int Find_k_largest(int low, int high, int k)
{
   int s=low, t=high, x=a[(s+t)/2];
   while(s < t){
      while(a[s]>x) s++;
      while(a[t]<x) t--;
      if(s<t) swap(&a[s++], &a[t--]);
   }
   if(k <= t+1) find_k_largest(low, t, k);
   else if(k >= s) find_k_largest(s,high, k-s);
   else return x;
}
main()
{
   ......
   cout << find_k_largest(0,n-1,k);
   ......
}
```

An example of a program

```
int Find_k_largest(int low, int high, int k)
{
   int s=low, t=high, x=a[(s+t)/2];
   while(s < t){
      while(a[s]>x) s++;
      while(a[t]<x) t--;
      if(s<t) swap(&a[s++], &a[t--]);
   }
   if(k <= t+1) find_k_largest(low, t, k);
   else if(k >= s) find_k_largest(s,high, k-s);
   else return x;
}
main()
{
   ......
   cout << find_k_largest(0,n-1,k);
   ......
}
```

計算時間の解析

最悪の場合，長さnの区間を長さ1とn-1の2つの区間に分割することになる．長さnの区間を処理するのに必要な時間をT(n)とすると，

$T(n) \leqq T(1)+T(n-1)+cn$

となる．これを解くと，

$T(n) = O(n^2)$.

平均比較回数をC(n,k)とすると，

$C(n,k)=n+1+(1/n)(\Sigma_{t=0\sim k-2}\, C(n-t-1,k-t-1)+\Sigma_{t=k+1\sim n-1}\, C(t+1,k))$

この漸化式を解くと，

$C(n,k) = O(n)$

となり，平均的には線形時間で終わることが分かる．

練習問題：アルゴリズムから上記の漸化式を導け．

練習問題：上記の漸化式を解け．

Analysis of Computation Time

Worst case: an interval of length n is decomposed into intervals of lengths 1 and n-1. If we denote by T(n) time for processing an interval of length n, we have

$T(n) \leqq T(1)+T(n-1)+cn.$

Solving it, we have

$T(n) = O(n^2)$.

If we denote the average number of comparisons by C(n,k),

$C(n,k)=n+1+(1/n)(\Sigma_{t=0\sim k-2}\, C(n-t-1,k-t-1)+\Sigma_{t=k+1\sim n-1}\, C(t+1,k))$.

Solving this recurrence equation, we have

$C(n,k) = O(n)$,

which implies that the average running time of the algorithm is O(n).

Exercise : Obtain the recurrence equation from the algorithm.

Exercise: Solve the above recurrence equation.

余談(Side story):

*k*番目の要素を選ぶ問題は、Selection Problem と呼ばれていて、以下の文献の線形アルゴリズムのすばらしさから、とても有名。(This “find the *k*-th element” problem is called Selection Problem, which is famous by the great linear time algorithm shown in the following paper):

Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. "Time bounds for selection". *Journal of Computer and System Sciences* **7** (4): 448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

どの著者も現在は「大御所」や「神様」です。(Each author is now “legend” or “god” ☺)

最悪の場合にも線形時間のアルゴリズム

アルゴリズムP9-A2:

(1)n個のデータを15個ずつのグループに分割し，各グループごとに15個のデータの中央値を求める.

(2)このようにして得られたn/15個の中央値に，この方法を再帰的に適用し，中央値の中央値Mを求める.

(3)n個のデータをMに関して分割:

S = Mより小さいデータの集合,

L = Mより大きいデータの集合,

E = Mに等しいデータの集合.

(4) $k \leqq |L|$のとき，Lの中でk番目に大きな要素を再帰的に求める.

(5) $k>|L|+|E|$のとき，Sの中で$k-|L|-|E|$番目に大きな要素を再帰的に求める.

(6) 上記以外の場合，Mが求める答である.

S	E	L

Linear-time Algorithm in the worst case

Algorithm P9-A2:

(1)decompose n data into groups each containing at most 15 data and find the median in each group.

(2)Find the median M of these n/15 medians obtained recursively.

(3)Decompose the n data with respect to M:

S = a set of data < M,

L = a set of data > M,

E = a set of data = M.

(4) If $k \leqq |L|$, find the k-th largest element in L recursively.

(5) If $k>|L|+|E|$, find the $(k-|L|-|E|)$-th largest element in S recursively.

(6) Otherwise, return M as a solution.

例題

: 24個のデータをサイズ5のグループに分割し，中央値を求める

S = {12,56,43,22,31,25,57,75,45,33,39,85,37,44,19,28,18,23,92,73,77,28,64,35}

S = {12,56,43,22,31,25,57,75,45,33,39,85,37,44,19,28,18,23,92,73,77,28,64,35}

中央値 ={31,45,39,28,35}　中央値の中央値M = 35

35より大きい = {56,43,57,75,45,39,85,37,44,92,73,77,64}　13 要素 > 24/2

全体の中央値はこの集合にある.

この集合で12番目に大きい要素を再帰的に求める

L = {56,43,57,75,45,39,85,37,44,92,73,77,64}

中央値 = {56,44,73},中央値の中央値M = 56

56より大きい = {57,75,85,92,73,77,64}　7 要素 > 13/2

12番目に大きい要素はこの集合にない

残りの集合の中で (12-7)番目に大きい要素を求める

S = {56,43,45,39,37,44}　5番目に大きい要素= 39

全体の中央値は 39

実際

> 39:　56,43,57,75,45,85,44,92,73,77,64,

39

< 39:　12,22,31,25,33,37,19,28,18,23,28,35

Example: Decompose 24 data into groups of size 5 to find the median.

S = {12,56,43,22,31,25,57,75,45,33,39,85,37,44,19,28,18,23,92,73,77,28,64,35}

S = {12,56,43,22,31,25,57,75,45,33,39,85,37,44,19,28,18,23,92,73,77,28,64,35}

group medians ={31,45,39,28,35} the median of the mediansM = 35

data > 35 = {56,43,57,75,45,39,85,37,44,92,73,77,64} 13 elements > 24/2

The overall median must be in this set

Find the 12th largest element in this set recursively

S = {56,43,57,75,45,39,85,37,44,92,73,77,64}

group medians = {56,44,73}, the median of the mediansM = 56

data > 56 = {57,75,85,92,73,77,64} 7 elements > 13/2

The 12-th largest element cannot be in this set

Find the (12-7)-th largest element in the remaining set.

S = {56,43,45,39,37,44} 5-th largest lement= 39

The overall median is 39

In fact,

> 39: 56,43,57,75,45,85,44,92,73,77,64,

39

< 39: 12,22,31,25,33,37,19,28,18,23,28,35

計算時間の解析

15個のデータのソート： 42回の比較で十分
　　全体では， 42×(n/15) 回の比較

Mは中央値の中央値であるから，
　M以上の中央値をもつグループは(n/15)/2グループ
　それらのグループでは半数（8個）以上が中央値以上．
　よって，M以上の要素数は少なくとも (8/30)n = (4/15)n
つまり，M以下の要素数もM以上の要素数も高々(11/15)n個

Mに関する分割にn回の比較が必要

以上より，
　$T(n) \leqq 42(n/15) + T(n/15) + n + T((11/15)n)$
よって，
　$T(n) \leqq 19n$.

練習問題：上の漸化式を解け．

Analysis of Computation Time

Sort of 15 data: 42 comparisons suffice
in total, $42 \times (n/15)$ comparisons

M is the median of the group medians, and so
there are $(n/15)/2$ groups whose median are $\geqq$M,
where 8 or more elements (more than half) are $\geqq$ M.
Thus, there are at least $(8/30)n = (4/15)n$ data $\geqq$M.
That is, there are at most $(11/15)n$ elements $\leqq$M and $\geqq$M.

For the decomposition w.r.t. M, n comparisons are required.

From the above arguments, we have
$T(n) \leqq 42(n/15) + T(n/15) + n + T((11/15)n)$
Hence,
$T(n) \leqq 19n$.

Exercise: Solve the above recurrence equation.

とても有用なツール：マスター定理

定理 4.1 (分類定理 (master theorem)) $a \geq 1$ と $b > 1$ を定数とし，$f(n)$ を関数とする．また，$T(n)$ は次の漸化式によって非負の整数に関して定義されているものとする．

$$T(n) = aT(n/b) + f(n) .$$

ここで，n/b は $\lfloor n/b \rfloor$ または $\lceil n/b \rceil$ を意味するものと解釈する．このとき，$T(n)$ は漸近的に次のような限界をもつ．

1. ある定数 $\epsilon > 0$ に対して $f(n) = O(n^{\log_b a - \epsilon})$ ならば，$T(n) = \Theta(n^{\log_b a})$ である．
2. $f(n) = \Theta(n^{\log_b a})$ ならば，$T(n) = \Theta(n^{\log_b a} \lg n)$ である．
3. ある定数 $\epsilon > 0$ に対して $f(n) = \Omega(n^{\log_b a + \epsilon})$ であり，しかもある定数 $c < 1$ と十分大きな n に対して $af(n/b) \leq cf(n)$ ならば，$T(n) = \Theta(f(n))$ である．

『アルゴリズムイントロダクション』
コルメン・ライザーソン・リベスト著，
浅野哲夫・岩野和生・梅尾博司・山下雅史・和田幸一訳，近代科学社

Very useful tool; master theorem

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

"Introduction to Algorithms"
T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein
MIT Press, 2009.