Information Science
10: Algorithm and Complexity
Sorting Algorithm

Naonori Kakimura

垣村尚徳

kakimura@global.c.u-tokyo.ac.jp

PEAK
Programs in English at Komaba

Schedule

- Today: Sorting algorithm

- Dec 19:
  - String (and random number)
- Class cancelation on Dec 26
- Jan 7:
  - Dynamic programming
- Jan 13(Make-up), P2
  - Answers and Q&A

Today's Contents

- Review of Fibonacci Numbers
  - Solutions of Exercise 2
  - Eigenvalue problem

- Sorting algorithm
  - Introduction
  - Two basic algorithms
    - simple sort
    - merge sort
- Exercises

(Review) Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, $\cdots$
  - Definition: Letting $\boldsymbol{f_k}$ be the kth number,

$$f_k = \begin{cases} f_{k-1} + f_{k-2} & (k \geq 2) \\ 1 & (k = 0, 1) \end{cases}$$

- Problem: find the kth number of the sequence
  - How many mice are there on the kth day?

Efficient Computing by Iteration


```
def fibl(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1      #fib(i-2)
    p1 = f       #fib(i-1)
    f = p1 + p2 #fib(i)
  end
  f             #fib(k)
end
```

For each i, update p2, p1

Orderings in the for-loop is important

Efficient Computing by Iteration


```
def fibl(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1      #fib(i-2)
    p1 = f       #fib(i-1)
    f = p1 + p2 #fib(i)
  end
  f              #fib(k)
end
```

k	0	1	2	3	4	5	6	7	8	9	10	11	12
fib(k): f	1	1	2	3	5	8	13	21	34	55	89	144	233
Last : p1	–	1	1	2	3	5	8	13	21	34	55	89	144
Before last p2	–	–	1	1	2	3	5	8	13	21	34	55	89

f_2, f_1 f_3, f_2

Exercise 2: Ordering of Assignment

- Suppose that we swap the two lines of fibl as follows. Explain what sequence will be obtained by completing the table

```
def fibl_error(k)
  f=1
  p1=1
  for i in 2..k
    p1 = f        #fib(i-1)
    p2 = p1       #fib(i-2)
    f = p1 + p2 #fib(i)
  end
  f             #fib(k)
end
```

k	0	1	2	3	4	5	6	7	8	9	10	11	12
$\mathrm{fib}(k)$: f													
Last p1													
Before last : p2													

Exercise 2: Ordering of Assignment


```
def fibl_error(k)
  f=1
  p1=1
  for i in 2..k
    p1 = f        #fib(i-1)
    p2 = p1       #fib(i-2)
    f = p1 + p2 #fib(i)
  end
  f               #fib(k)
end
```

Red part changes p1 before assigning p2.
→ p2=p1=(previous) f

	i	0	1	2	3	4	5	6	7	8	9	10	11
	$\mathrm{fib}(k)$: f	-	1	2	4	8	16	32	64	128	⋯		
Last	p1	-	1	1	2	4	8	16	32	64	⋯		
Before last	p2	-	-	1	2	4	8	16	32	64	⋯		

Exercise 4: Review of Eigenvalues

➢ Explain also how you obtain solutions

6. Let f_k be the kth Fibonacci number. Then it holds that

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix}.$$

(a) Explain why the above equation holds.

(b) Let A be the matrix in the form of

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Determine the eigenvalues λ_1 and λ_2 of A.

(c) By the definition of eigenvalues, there exists a nonsingular(invertible) matrix U such that

$$A = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1}$$

Determine the matrix U and U^{-1}.

(d) By (a), we have

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = A^k \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = A^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Using the equation in (c), express the kth Fibonacci number f_k with λ_1 and λ_2.

Fibonacci Numbers by Matrix Computation

- It holds that

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_{k-1} \\ f_{k-2} \end{pmatrix}$$

k matrix multiplications

$$= \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- If you obtain the eigenvalue decomposition $A = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1}$

$$A^k = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} \cdots U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} = U \begin{pmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{pmatrix} U^{-1}$$

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = U \begin{pmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{pmatrix} U^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

3 matrix multiplications
kth power of lambda

Hand Calculation by Definition

- Eigenvalues and corresponding eigenvectors

$$\lambda_1 = \frac{1+\sqrt{5}}{2} \qquad x_1 = \begin{pmatrix} \lambda_1 \\ 1 \end{pmatrix}$$

$$\lambda_2 = \frac{1-\sqrt{5}}{2} \qquad x_2 = \begin{pmatrix} \lambda_2 \\ 1 \end{pmatrix}$$

- The matrix U is given by $U = \begin{pmatrix} \lambda_1 & \lambda_2 \\ 1 & 1 \end{pmatrix}$ $\quad U^{-1} = \frac{1}{\lambda_1 - \lambda_2} \begin{pmatrix} 1 & -\lambda_2 \\ -1 & \lambda_1 \end{pmatrix}$

$$\left(AU = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U \right)$$

Plug in them into

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = U \begin{pmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{pmatrix} U^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Continuation of Calculation

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \frac{1}{\lambda_1 - \lambda_2}\begin{pmatrix} \lambda_1 & \lambda_2 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} \lambda_1{}^k & 0 \\ 0 & \lambda_2{}^k \end{pmatrix}\begin{pmatrix} 1 & -\lambda_2 \\ -1 & \lambda_1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1-\lambda_2 \\ -1+\lambda_1 \end{pmatrix}$$

$$\begin{pmatrix} \lambda_1{}^k(1-\lambda_2) \\ \lambda_2{}^k(-1+\lambda_1) \end{pmatrix}$$

$$\begin{pmatrix} \text{We don't need to obtain ?} \\ \lambda_1{}^k(1-\lambda_2) + \lambda_2{}^k(-1+\lambda_1) \end{pmatrix}$$

$$f_k = \frac{1}{\lambda_1 - \lambda_2}\left(\lambda_1{}^k(1-\lambda_2) + \lambda_2{}^k(-1+\lambda_1)\right)$$

$$\boxed{= \frac{1}{\sqrt{5}}\left(\lambda_1{}^{k+1} - \lambda_2{}^{k+1}\right)}$$

integer, but
has irrational numbers

Simplified using

$$\lambda_1 - \lambda_2 = \sqrt{5}$$

$$\lambda_1 + \lambda_2 = 1$$

Summary

- Application of Eigenvalue problem
  - review of math, although it is out of the class's scope

- Eigenvalue decomposition is useful
  - obtain the explicit representation of Fibonacci num
  - Also useful in engineering

Fibonacci Numbers by Matrix Computation

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

- It holds that

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_{k-1} \\ f_{k-2} \end{pmatrix}$$

k matrix multiplications

$$= \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- If you obtain the eigenvalue decomposition $A = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1}$

$$A^k = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} \cdots U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1} = U \begin{pmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{pmatrix} U^{-1}$$

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = U \begin{pmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{pmatrix} U^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

3 matrix multiplications
kth power of lambda

On Complexity

- We can find $\boldsymbol{f_k}$ by computing $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k$

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix} = \cdots = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Its 2nd entry is $\boldsymbol{f_k}$

- The k-th power of a matrix of order 3 can be computed in $\mathrm{O}(\log k)$ time
  - More efficient than the previous algorithms
  - See Exercise 7 in Section 6.2

Today's Contents

- Review of Fibonacci Numbers
  - Solutions of Exercise 2
  - Eigenvalue problem

- Sorting algorithm
  - Introduction
  - Two basic algorithms
    - simple sort
    - merge sort
- Exercises

Sorting: Primitive Method

- Order n elements in a specified order

[Alice,
Charlie,
Elizabeth,
David,
Bob,]

[Alice,
Bob,
Charlie,
David,
Elizabeth]

[7,3,5,2,4,1,6]

[1,2,3,4,5,6,7]

Problem: Sorting an Integer Sequence

- Order n integers in nondecreasing order
  - Used in various fields
    - sort student scores or IDs
    - dictionaries (regard a=1,b=2,..)
    - ranking in Google in terms of relevance

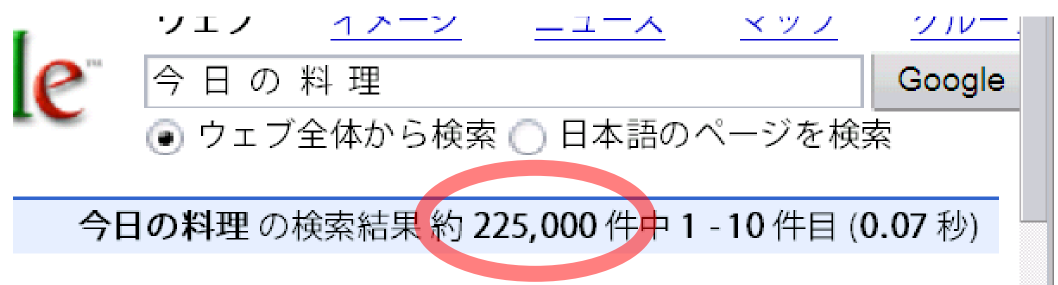
[7,3,5,2,4,1,6] → [1,2,3,4,5,6,7]

Problem: Sorting an Integer Sequence

- Order n integers in nondecreasing order
  - Used in various fields
    - sort student scores or IDs
    - dictionaries (regard a=1,b=2,..)
    - ranking in Google in terms of relevance

- #data n is often huge
    - used in preprocess of big data
  - # students in UTokyo > 3000 every year
  - # web pages > 1 trillion

Efficient sorting is required



Today's Contents

- Two basic algorithms for sorting
  - Simple sort
  - Merge sort
- Compare computational times
  - In practice
  - In theory

Today's Contents

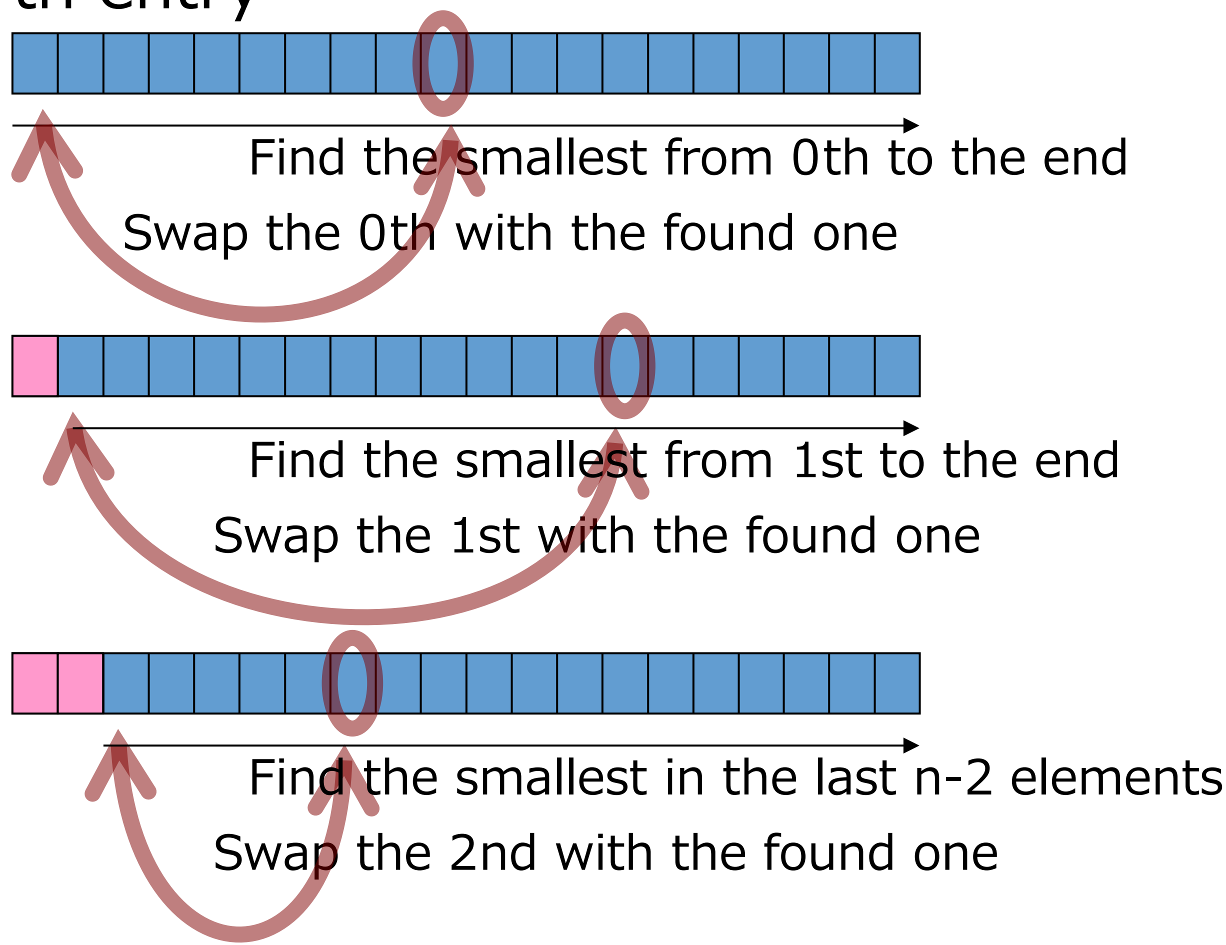
- Two basic algorithms for sorting
  - Simple sort
    - Repeatedly find the minimum in remaining items
  - Merge sort
    - Sort a part of a sequence, and merge it

Cf) many other algorithms (see Past Exam 2015 Q2)

- Having many basic techniques in algorithm design
- Bin sort
- Quick sort
- Radix sort

Simple Sort Algorithm (n=length of a)

- For each i=0,1,2,⋯, n-1
  - Find the i-th smallest number
    - = the smallest num from i-th to the end
  - Move it to the i-th entry



Simple Sort


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

Simple Sort


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

Find the index of the minimum in the last (n-i) elements

Exercise: Simple Sort


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

Make the function min_index(a,i)

Simple Sort


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

Swap a[k] and a[i].
Necessary to use another variable v

Cf) Wrong Algorithm

- Auxiliary variable v is necessary

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

a[i] becomes a[k]

a[k] becomes a[i]

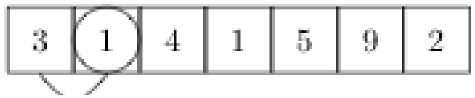
equal to a[k]

→ both become a[k]

Behaviors

Array a

i=0



Find the minimum

i=1

Find the min in a[1]..a[6]

i=2

Find the min in a[2]..a[6]

…

…

1 1 2 3 4 5 9

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

Behaviors

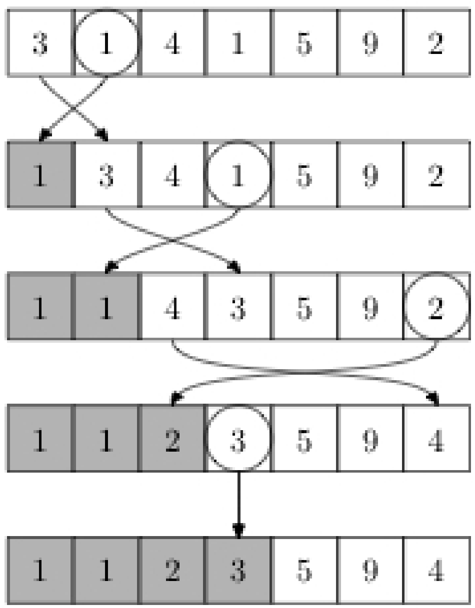
Array a

i=0

i=1

i=2

…



Find the minimum

Find the min in a[1]..a[6]

Find the min in a[2]..a[6]

…

1 1 2 3 4 5 9

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

Today's Contents

- Two basic algorithms for sorting
  - Simple sort
    - Description
    - Computational complexity
  - Merge sort
    - Sort a part of a sequence, and merge it
    - Description
    - Computational complexity

Review: How to Estimate the Complexity

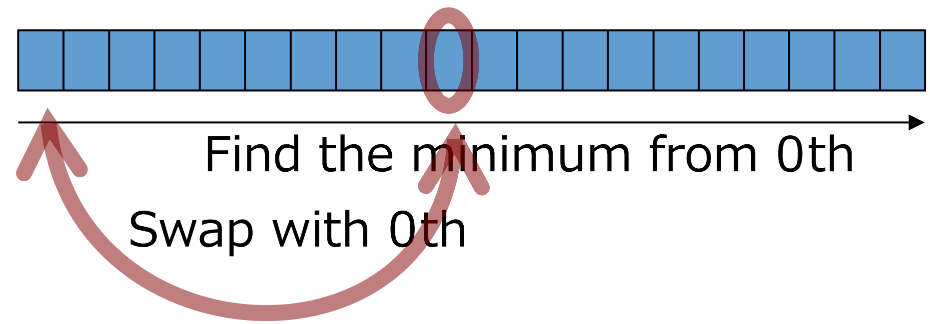
- Time ≈ # arithmetic operations & comparisons

1. We suppose that each operation, each calling of functions, and each branching are regarded as "one computation"
2. Express the number of computations using the size of an input
3. Delete constants, and leave the biggest term (= Order of the complexity)

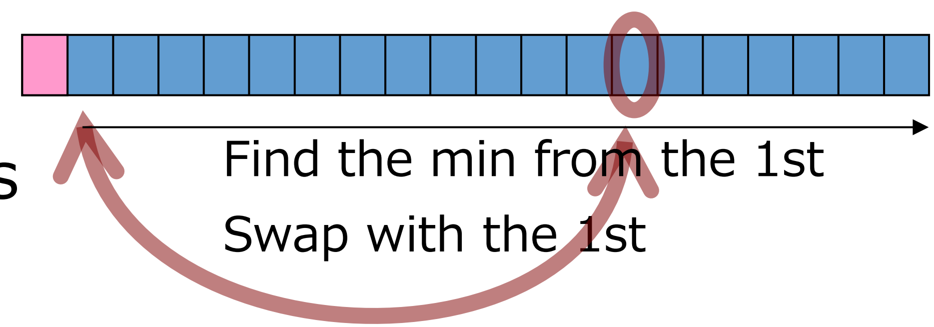
※ If 3 is accepted, it doesn't matter if 1, 2 are not rigorous

Complexity of Simple Sort

- # arithmetic operations
  - n: # numbers



- 0th step: n-1 comparisons
- 1st step: n-2 comparisons



- 2nd step: n-3 comparisons
- …
- i-th step: n-i-1 comparisons
- …
- (n-1)-th step: 0 comparisons

Proportional to n^2

$$\sum_{i=0}^{n-1}(n-i-1) = (n-1)+(n-2)+\cdots+1+0 = \frac{(n-1)n}{2} = O(n^2)$$

Summary of Computational Time of Sorting

Problem:

Sort an integer sequence of length at most $\boldsymbol{n}$

- Simple sort — $\mathbf{O}(\boldsymbol{n}^2)$
- Merge sort — ???

10 times 100 times

n	10	100	1,000	10,000	100,000	1.0E+06	1.0E+07	1.0E+08	1.0E+09
n^2	100	10,000	1.0E+06	1.0E+08	1.0E+10	1.0E+12	1.0E+14	1.0E+16	1.0E+18

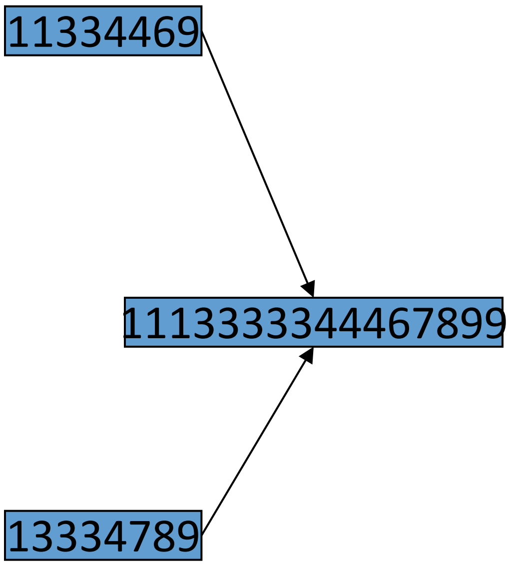
100 times 10000 times

Today’s Contents

- Two basic algorithms for sorting
  - Simple sort
    - Description
    - Computational complexity
  - Merge sort
    - Sort a part of a sequence, and merge it
    - Description
    - Computational complexity

Key Idea of Merge Sort

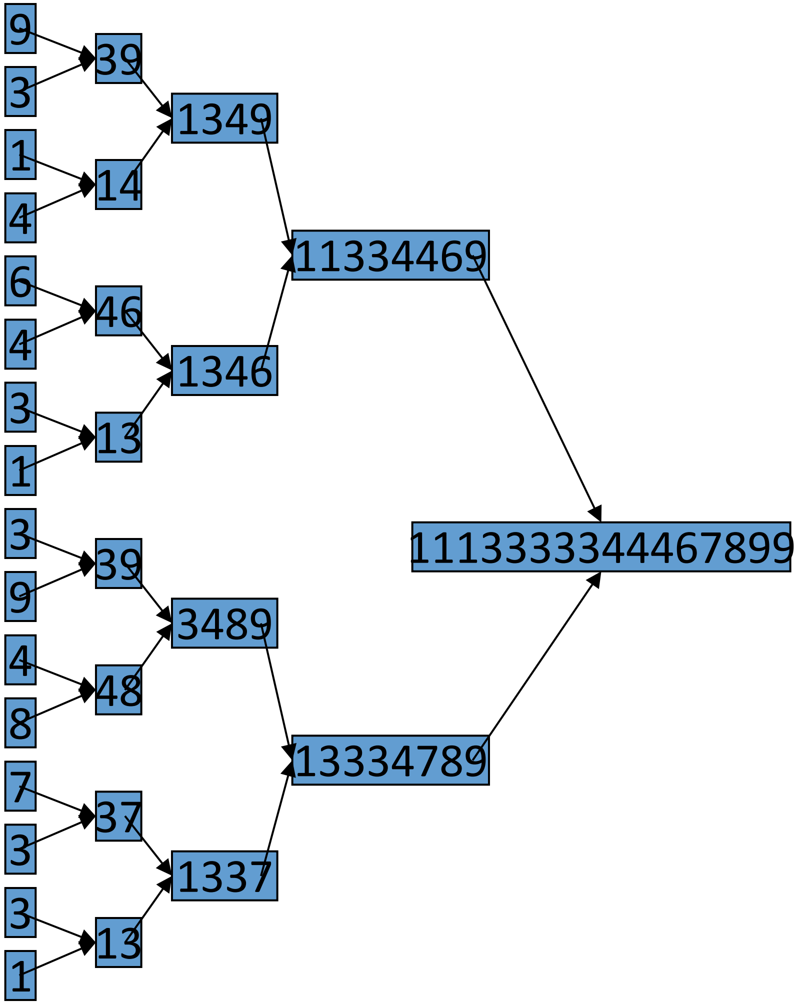
- If we have two sorted sequences, then it is easy to sort
  - by combining the two sequences
    - This procedure is called merge



Apply it repeatedly

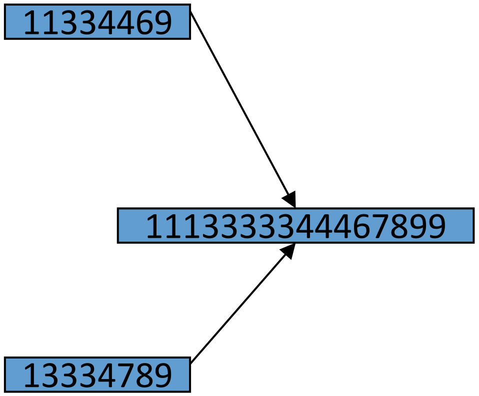
Key Idea of Merge Sort

- Begin with sequences each of which has one number
  - Obtain sorted sequences of size two
- From current sorted sequences
  - Obtain sorted sequences
    - size doubles
- Repeat until only one sequence

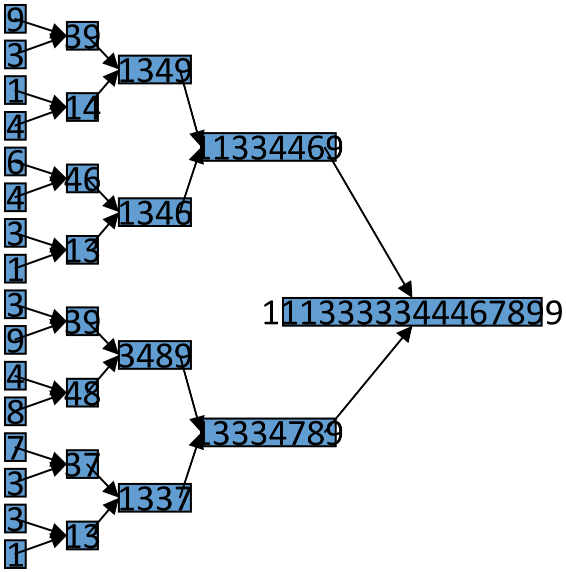


Two Tasks to Make the Merge Sort Algorithm

- Prepare the function merge(a,b)
  - Merge two (sorted) sequences



- Make the function mergesort(a)
  - Sort an array a using merge(a,b)



Prepare merge (from next slide)

Two sorted sequences

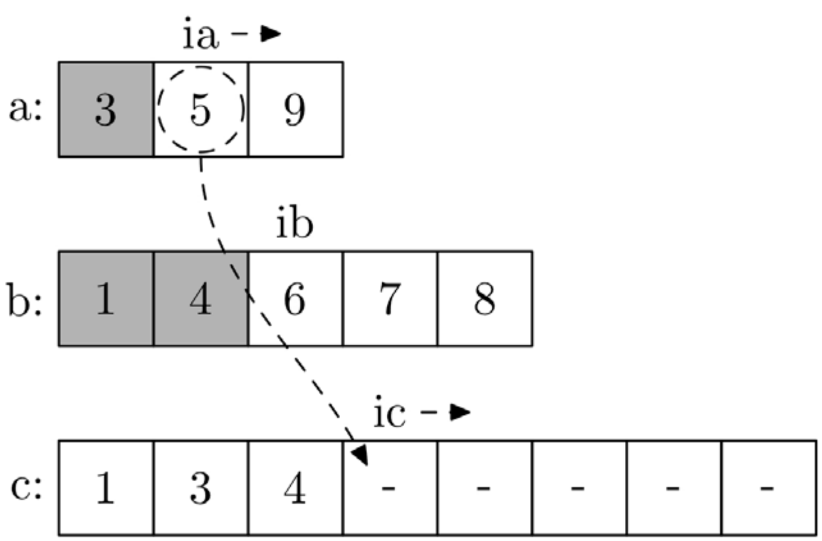
Prepare a new sequence

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

  Process for merging

  c
end
```

Return c having elements in a&b



Basic observation:

1st = min of a[0] & b[0]

3 > 1

2nd = min of a[0] & b[1]

3 < 4

3rd = min of a[1] & b[1]

Merging two sequences


```
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] then
      c[ic] = a[ia]
      ia = ia + 1
    else
      c[ic] = b[ib]
      ib = ib + 1
    end
    ic = ic + 1
  end
  Move all the remaining elements in either a or b to c
  c
end
```

"pointers" to elements in a,b,c

Move the smaller one in a, b to c

Behavior of Merge(a,b)

Prepare a new sequence

```
c = Array.new(a.length() + b.length())
ia=0
ib=0
ic=0
```

"pointers" to elements in a,b,c

a [3, 5, 6]
↑ia

b [1, 4, 7, 8, 10]
↑ib

c [, , , , , , ,]
↑ic

ia: a variable pointing the "current" index in a
ib: in b
ic: in c

Put the smaller one of a[ia] and b[ib] to c[ic]


```
while ia < a.length() && ib < b.length()
  if a[ia] < b[ib] then
    c[ic] = a[ia]
    ia = ia + 1
  else
    c[ic] = b[ib]
    ib = ib + 1
  end
  ic = ic + 1
end
```

put the smaller one of a[ia] and b[ib] to c[ic]

a [3, 5, 6]
↑ia

b [1, 4, 7, 8, 10]
↑ib

c [1, , , , , , ,]
↑ic

Update Pointing Indices


```
while ia < a.length() && ib < b.length()
  if a[ia] < b[ib] then
    c[ic] = a[ia]
    ia = ia + 1
  else
    c[ic] = b[ib]
    ib = ib + 1
  end
  ic = ic + 1
end
```

put the smaller one of a[ia] and b[ib] to c[ic]

Update the pointers

a [3, 5, 6]
↑ia

b [1, 4, 7, 8, 10]
↑ib

c [1, , , , , , ,]
↑ic

Repeat Moving the Smallest Value


```
while ia < a.length() && ib < b.length()
  if a[ia] < b[ib] then
    c[ic] = a[ia]
    ia = ia + 1
  else
    c[ic] = b[ib]
    ib = ib + 1
  end
  ic = ic + 1
end
```

a [3, 5, 6]
↑ia

b [1, 4, 7, 8, 10]
↑ib

put the smaller one of a[ia] and b[ib] to c[ic]

Update the pointers

c [1,3, , , , , ,]
↑ic

Repeat Until One of a and b Becomes Empty


```
while ia < a.length() && ib < b.length()
  if a[ia] < b[ib] then
    c[ic] = a[ia]
    ia = ia + 1
  else
    c[ic] = b[ib]
    ib = ib + 1
  end
  ic = ic + 1
end
```

put the smaller one of a[ia] and b[ib] to c[ic]

Update the pointers

a [3, 5, 6]

↑ia

c [1,3,4,5,6, , ,]

↑ic

b [1, 4, 7, 8, 10]

↑ib

Repeat until ia or ib exceeds the length of a or b

Finally, Move the Rest to c

- We have only one sorted sequence
  - Just copy all the remaining elements to c

Move all the remaining elements in either a or b to c

Exercises

a [3, 5, 6]

↑ia

c [1,3,4,5,6,7,8,10]

↑ic

b [1, 4, 7, 8, 10]

↑ib

Merging two sequences

Prepare a new sequence

```
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] then
      c[ic] = a[ia]
      ia = ia + 1
    else
      c[ic] = b[ib]
      ib = ib + 1
    end
    ic = ic + 1
  end
  Move all the remaining elements in either a or b to c
  c
end
```

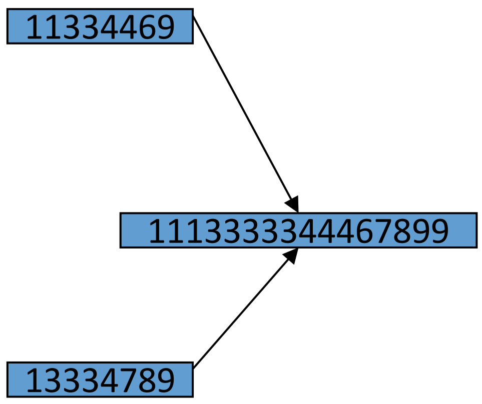
"pointers" to elements in a,b,c

Move the smaller one in a, b to c

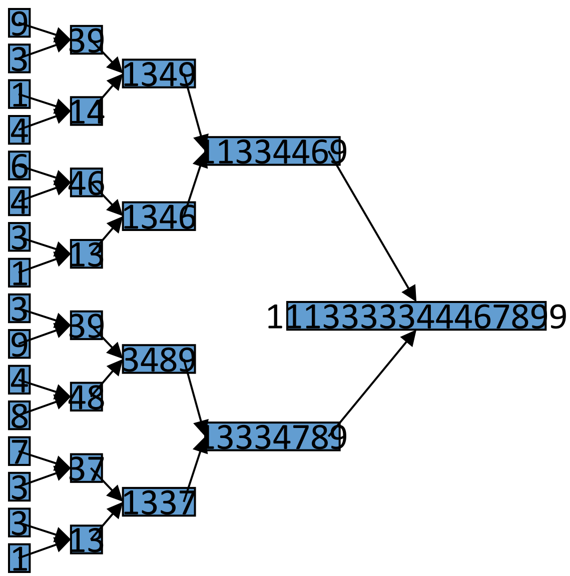
Exercises

Two Tasks to Make the Merge Sort Algorithm

- Prepare the function merge(a,b)
  - Merge two (sorted) sequences



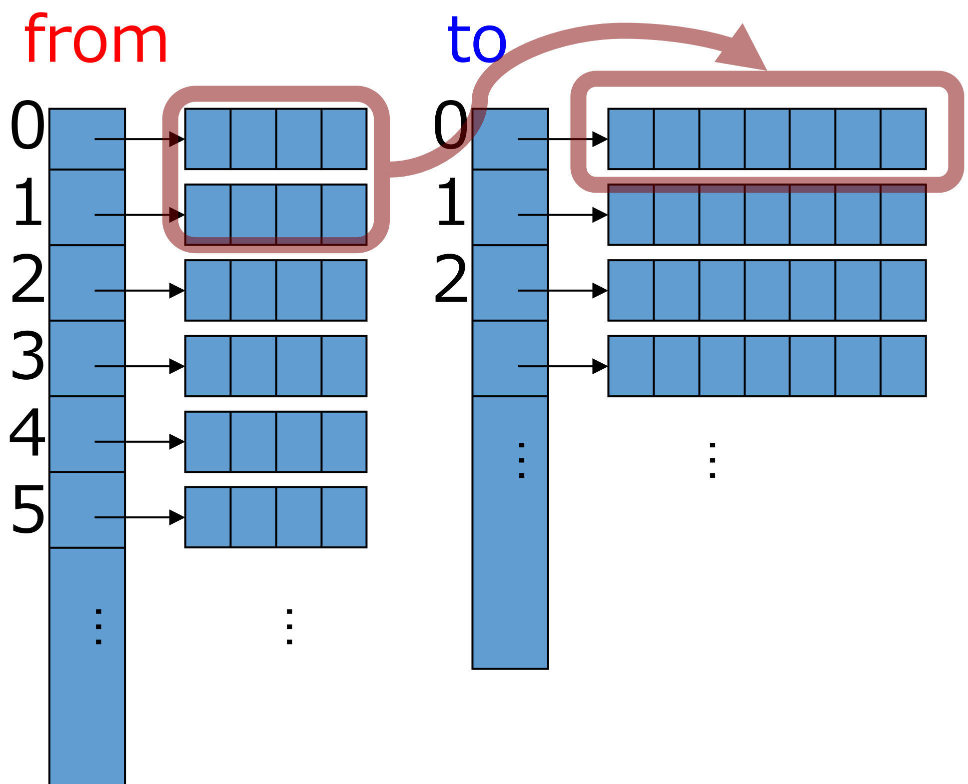
- Make the function mergesort(a)
  - Sort an array a using merge(a,b)



Merge Sort Algorithm

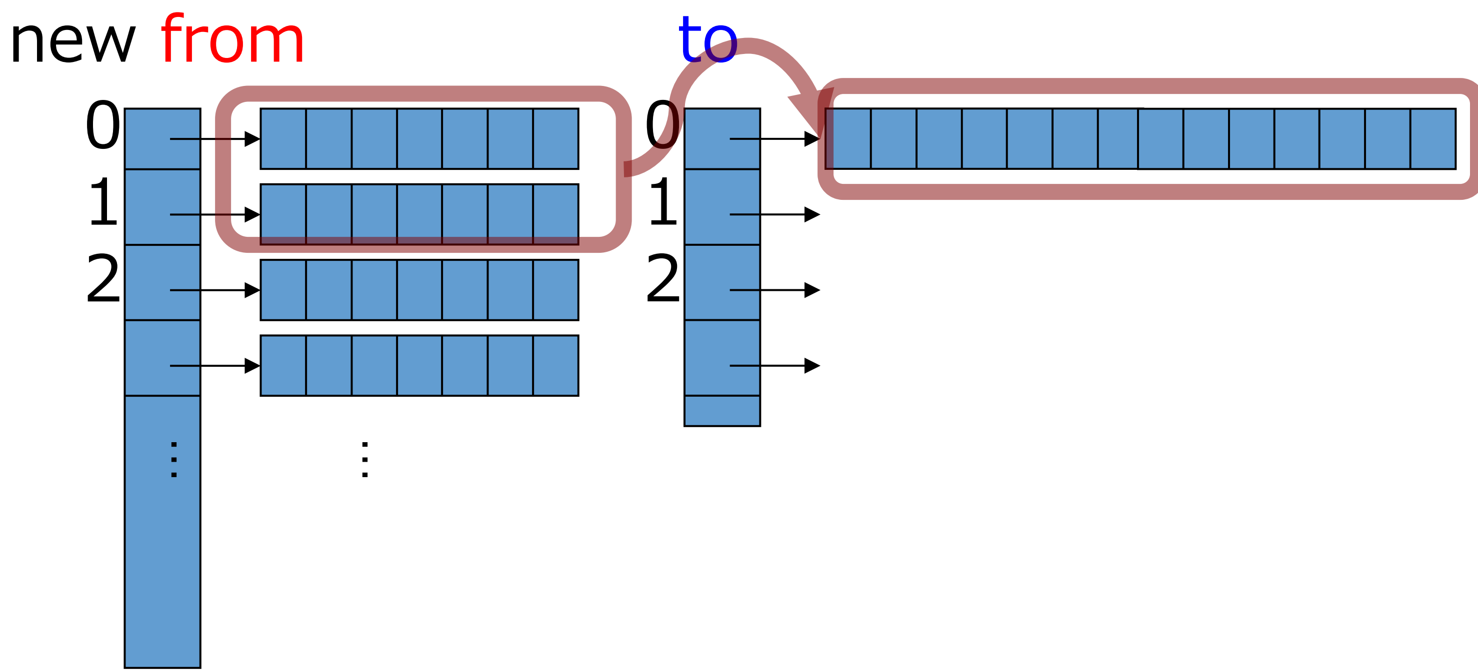
2-dim array

- Define a set of sorted sequences "from"
  - At first, each sorted sequence has one entry
- Make a new set "to" with size half of "from"
  - By merging two consecutive sequences in "from"
  - Replace "from" with "to"



Merge Sort Algorithm

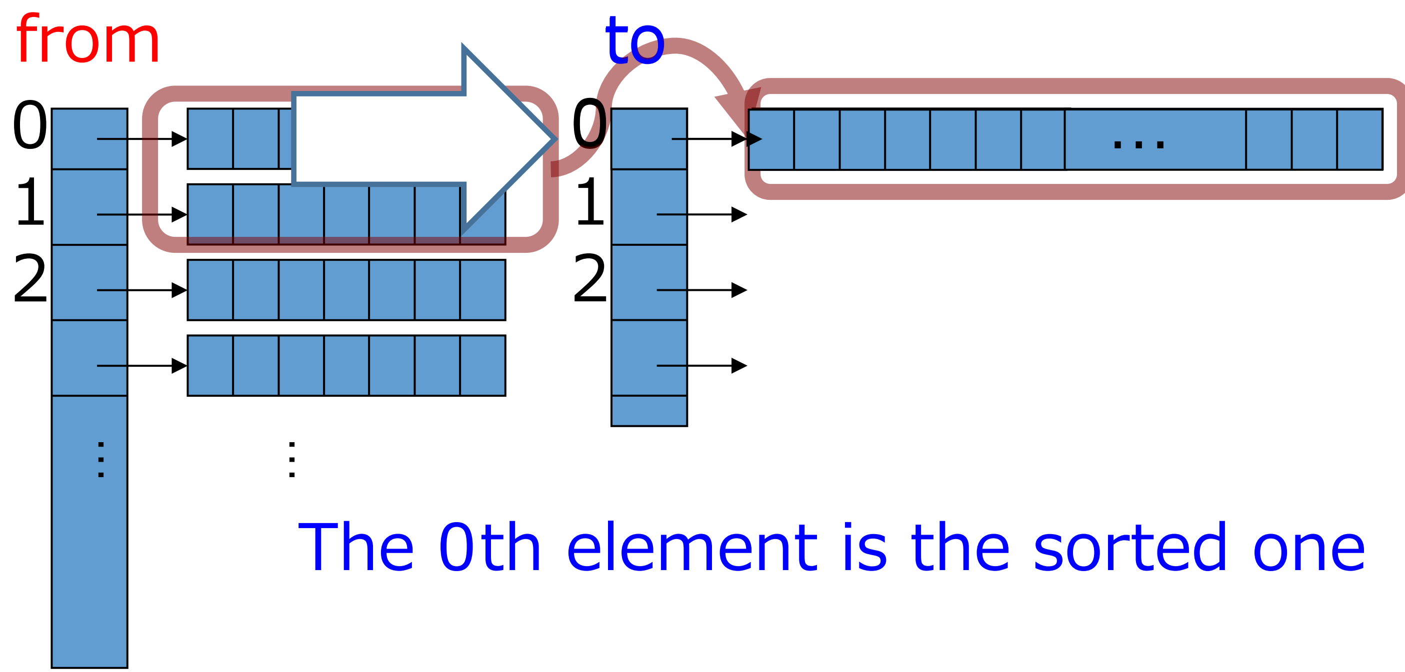
- Define a set of sorted sequences "from"
  - At first, a set of sequences with size one
- Make a new set "to" with size half of "from"
  - By merging two consecutive sequences in "from"
  - Replace "to" with "from"



Repeat until we have only one (sorted) sequence

Merge Sort Algorithm

- Define a set of sorted sequences “from”
  - At first, a set of sequences with size one
- Make a new set “to” with size half of “from”
  - By merging two consecutive sequences in “from”
  - Replace “to” with “from”



Merge Sort: Repeat Merging


```
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
```

Merge Sort: Repeat Merging


```
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
```

make an array “from”
each entry is a size-1 array

Merge Sort: Repeat Merging


```
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
```

Make an array “to” of half-size

Merge Sort: Repeat Merging


```
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
```

Merge two elements in “from” and put it into “to”

If “from” has odd arrays copy the last array

Merge Sort: Repeat Merging


```
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
```

Replace “from” with “to”

Replace n with the size of “to”

Merge Sort: Repeat Merging


```
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
```

Make an array “to” of half-size

Merge two elements in “from” into “to”

If from has odd arrays copy the last array

Replace “from” with “to”

Replace n with the size of “to”

Repeat

Merge Sort: Repeat Merging


```
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
```

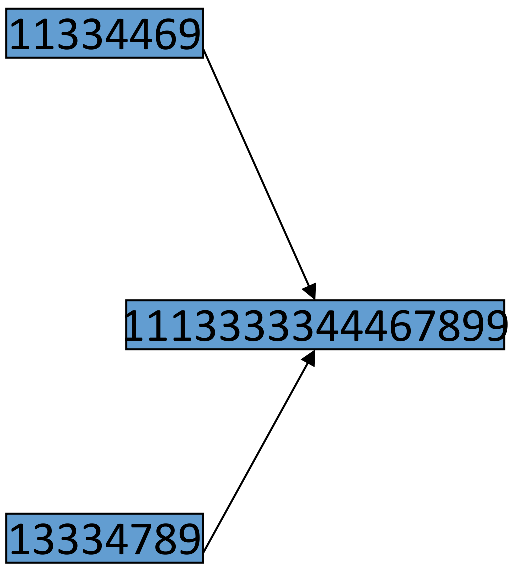
Finally, "from" has only one array, which is sorted one

Today's Contents

- Two basic algorithms for sorting
  - Simple sort
    - Description
    - Computational complexity
  - Merge sort
    - Sort a part of a sequence, and merge it
    - Description
    - Computational complexity

Complexity of Merge Sort

- Computation time of function merge:
  - Each iteration copies an element in a or b to c
  - # operation
    ≈ # iterations = O((length of a)+(length of b))

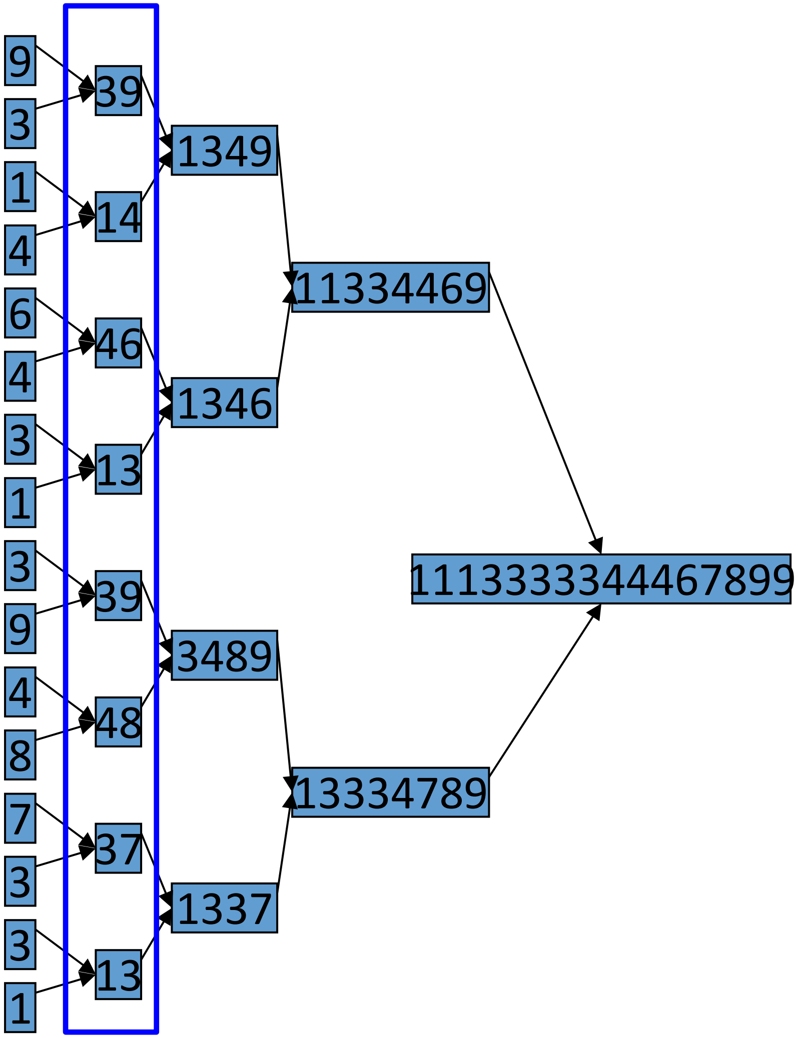


Complexity of Merge Sort

- Computation time:
  - 1st step: total = $\mathrm{O}(n)$ to obtain all sequences
  - 2nd step: total = $\mathrm{O}(n)$
  - …
  - Final(t-th) step: $\mathrm{O}(n)$
    - t :min integer satisfying $\dfrac{n}{2^t} \leq 1$

$$\rightarrow t \cong \log n$$

Total: $\mathbf{O}(\boldsymbol{n} \log \boldsymbol{n})$



Summary of Computational Time of Sorting

Problem:

Sort an integer sequence of length at most n

- Simple sort — $\mathbf{O}(n^2)$
- Merge sort — $\mathbf{O}(n \log n)$

10 times 1000 times

n	10	100	1,000	10,000	100,000	1.0E+06	1.0E+07	1.0E+08	1.0E+09
n^2	100	10,000	1.0E+06	1.0E+08	1.0E+10	1.0E+12	1.0E+14	1.0E+16	1.0E+18
$n \log n$	33	664	9,966	132,877	1.7E+06	2.0E+07	2.3E+08	2.7E+09	3.0E+10

～10 times 2000 times

Better than 1000^2

Funny Visualization of Sorting

- See also
  - http://cg.scs.carleton.ca/~morin/misc/sortalg/
- Video
  - Simple sort(=selection sort)
    - http://www.youtube.com/watch?v=Ns4TPTC8whw
  - Merge sort
    - http://www.youtube.com/watch?v=XaqR3G_NVoo

Exercises

Exercise 1

Suppose a=[1,4,2,9,8,3,2,6,4]

(1) When we apply the simple sort to a, write what a is at the beginning of the i-th iteration for each i.

(2) When we apply the merge sort to a, write what “from” is at the beginning of the i-th iteration for each i.

Examples of (1) and (2) in Exercise 1

- Write how the sequence will change for each iteration

(1)

(2)

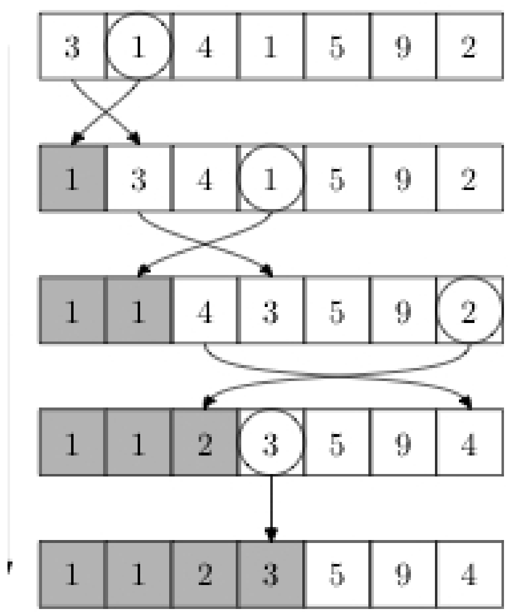
Each column corresponds to one iteration

i=0

i=1

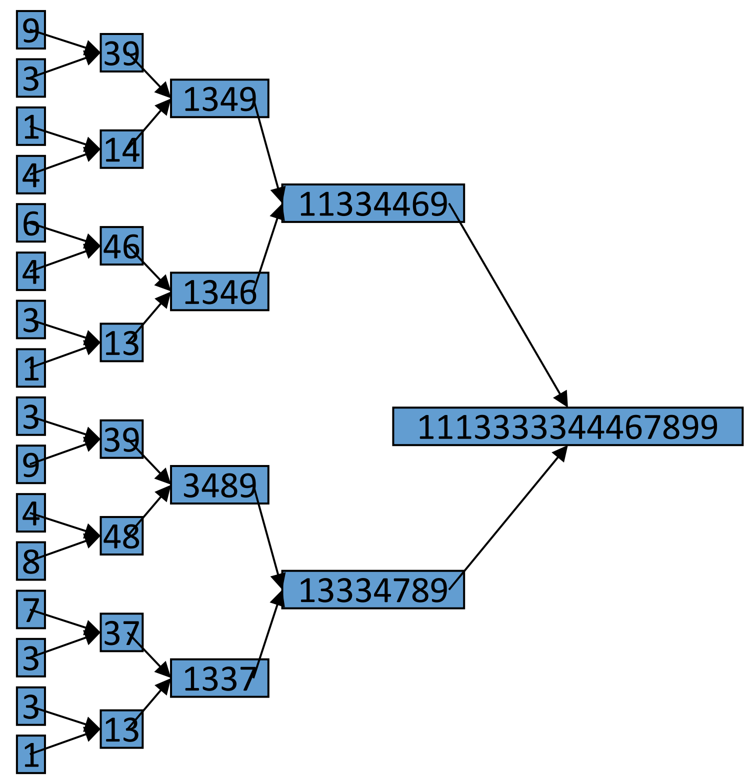
i=2

…



…

1 1 2 3 4 5 9



Exercises

2. Complete the simple sort algorithm

- By defining the function min_index(a,i)

3. Complete the merge sort algorithm

- By filling in the missing parts in merge(a,b)

Exercises

4. Compare the actual computational time

- Using compare_sort.rb(or compare_sort2.rb), make a graph showing the computational times
  - see Exercise Handout on how to use compare_sort
    - Problem 5 in 6.2
  - compare_sort2 is an alternative if you cannot use bench
    - compare_sort2 outputs only a table.
    - Make a graph using another software such as Excel

- Discuss
  - How proportional to the size of input
  - Compare to theoretical computational complexity

See Exercise Hand-out 6-2-4


```
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
```

compare_sort.rb

※ Randoms.rb is available at ITC-LMS

If You cannot use “bench”


```
load("./randoms.rb")    # randoms(id,size,max)
load("./simplesort.rb") # simplesort(a)
load("./mergesort.rb")  # mergesort(a)
def compare_sort2(max, step)
  print "size¥t simplesort ¥t mergesort¥n"
  for i in 1..(max/step)
    x=i*step
    print x, "¥t"
    a=randoms(i,x,1)
    t = Time.now # time before execution
    simplesort(a)
    print Time.now-t, "¥t"
    t = Time.now # time before execution
    mergesort(a)
    print Time.now-t, "¥n"
  end
end
```

compare_sort2.rb

(If you have time)Exercise 5: Other Algorithms

5. Solve Past Exam 2015 Problem 2

- Bubble sort and Bucket sort

Deadline

- **By Dec.28（Wed） 23:59**
  - Two weeks to do

- Submit
  - By ITC-LMS
  - It is OK to submit a scanned hand-written one

- Note
  - This exercise may have higher points than usual
    - because of the amount

  (Every week exercises may have different points)