Lecture 4: Divide and Conquer

Algorithm

Jeong-Hun Kim

Remind

- Performance evaluation
  - Running time
  - Confusion matrix
    - Accuracy
    - Precision
    - Recall
    - F1-score
  - MAE, MSE

- Comparison of algorithms
  - Theoretical comparison
  - Experimental comparison
    - Fair comparison

Table of Contents

Part 1

WHAT IS DIVIDE AND CONQUER

What is divide and conquer?

- Divide and conquer
  - Strategy of solving a big problem through small partial problems
  - Solving the problem recursively
  - There are three steps:
    - Divide
      - Breaking down a large problem into smaller sub-problems
    - Conquer
      - Solve the divided problems recursively
    - Combine
      - Integrate the solutions to the sub-problems
      - It can generate a solution for the original problem

What is divide and conquer?

- Subproblem
  - It can be solved with the same algorithm as the original problem
    - Only the size is smaller
  - Base case
    - Subproblem that cannot be further divided

- Advantages of the divide and conquer
  - Simplification of the problem
  - Efficient time complexity
  - Parallel programming (scalability)

What is divide and conquer?

- Drawbacks of the divide and conquer
  - Overheads like context switch due to recursive function calls
  - Poor space complexity
  - High implementation complexity

What is divide and conquer?

- When should we use divide and conquer?
  - Problems that are sufficiently large and complex
  - Distributed processing is required
  - High independence between subproblems

Part 2

REPRESENTATIVE ALGORITHMS

Representative algorithms

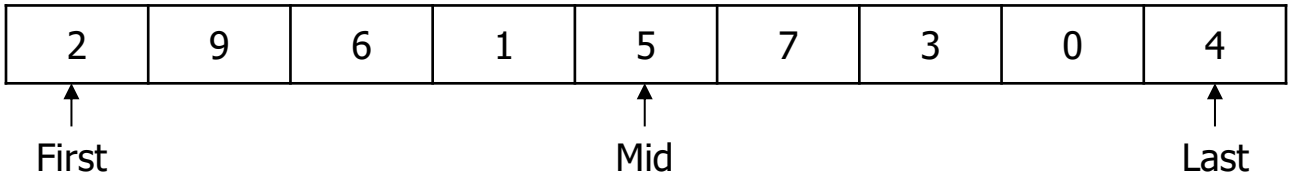
- Merge sort algorithm
  - It divides a target list into sublists and sorts them
  - Three steps:
    - Divide
      - The list is recursively divided into two sublists based on its midpoint
      - Base case is a list of length 1
    - Conquer
      - Each sublist is sorted
    - Combine
      - Two sorted sublists are merged
      - Comparing the elements of two lists and inserting smaller element first

Representative algorithms

- Merge sort algorithm
  - Check lists
    - Are the solutions to subproblems derived using the same algorithm?
      - Yes (sorting and comparing algorithm)
    - Are each of the subproblems independent?
      - Yes

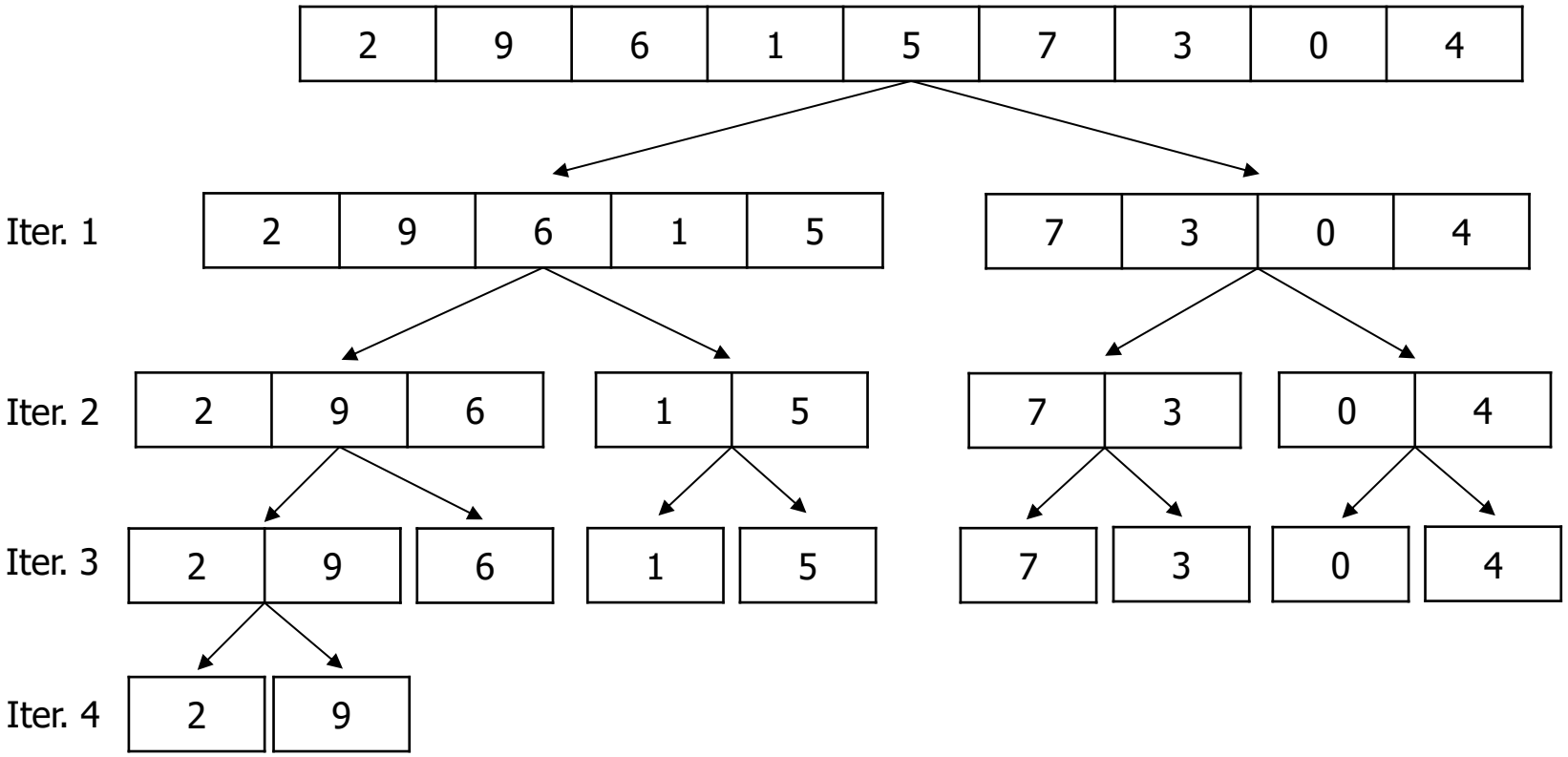
Representative algorithms

- Merge sort algorithm
  - Example of merge sort algorithm
    - Task: sort the list [2, 9, 6, 1, 5, 7, 3, 0, 4] in ascending order



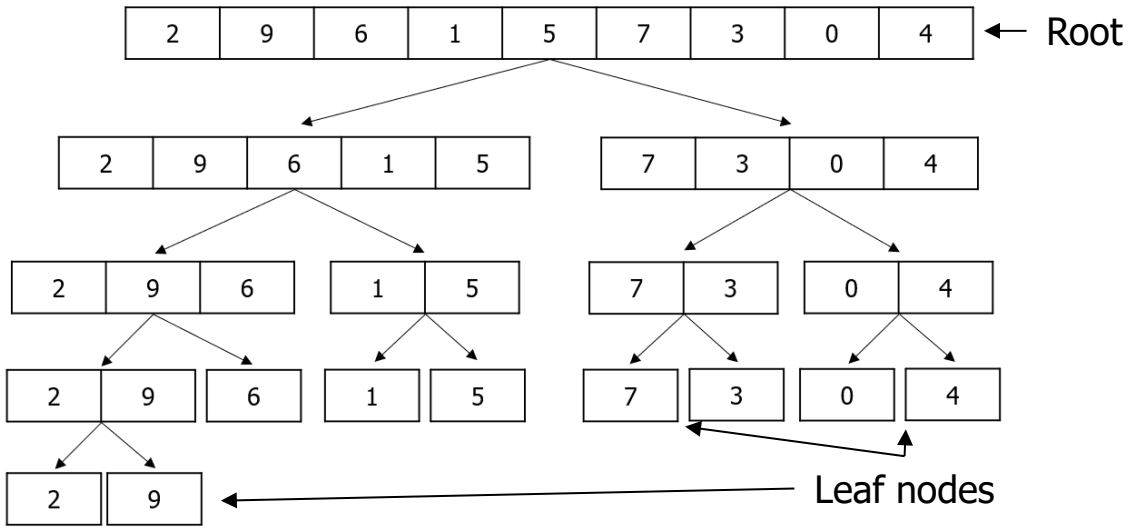
Representative algorithms

- Merge sort algorithm
  - Example of merge sort algorithm
    - Recursively divide the list (divide step)



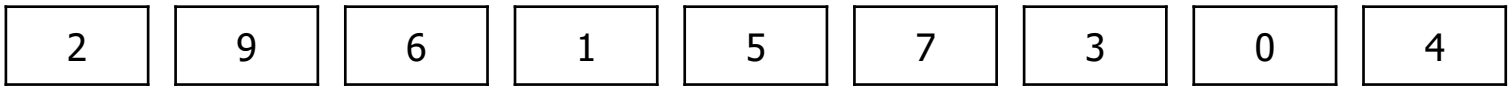
Representative algorithms

- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (divide step)
      - Recursion tree
        - » The structure where subproblems are derived through recursion
        - » Its root corresponds to the input of the original problem
        - » Its leaf nodes represent the base cases



Representative algorithms

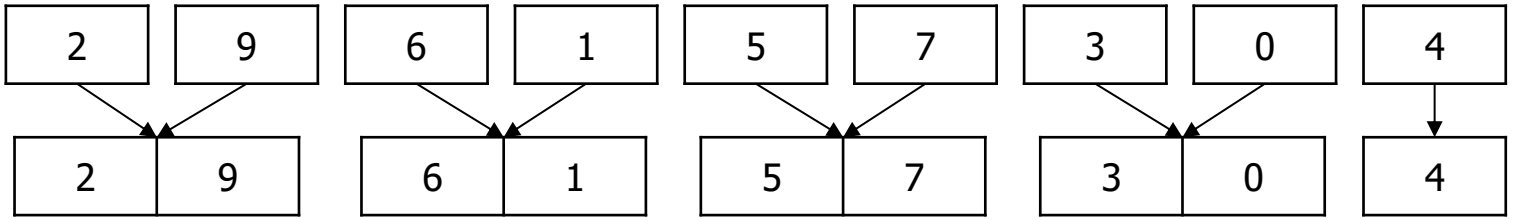
- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)



Sorted?

Representative algorithms

- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)

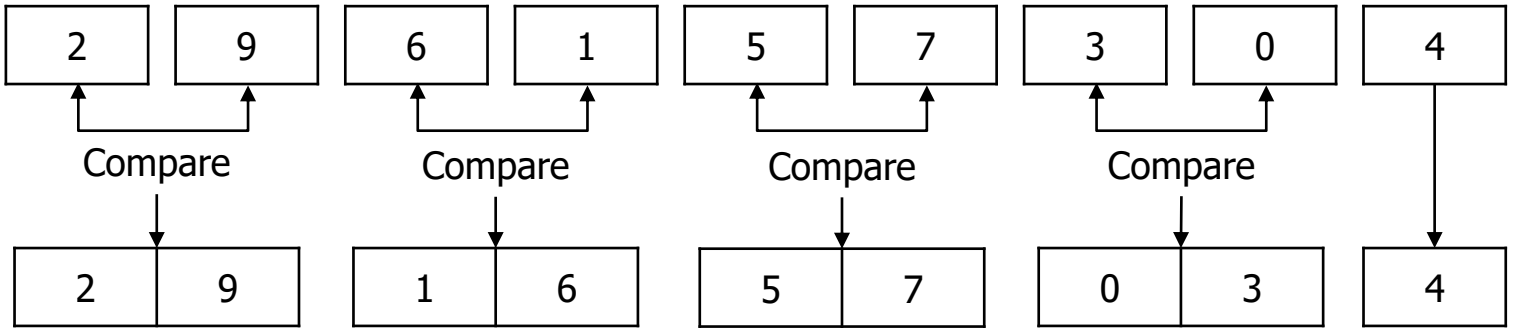


Sorted? X

If not, how to sort?

Representative algorithms

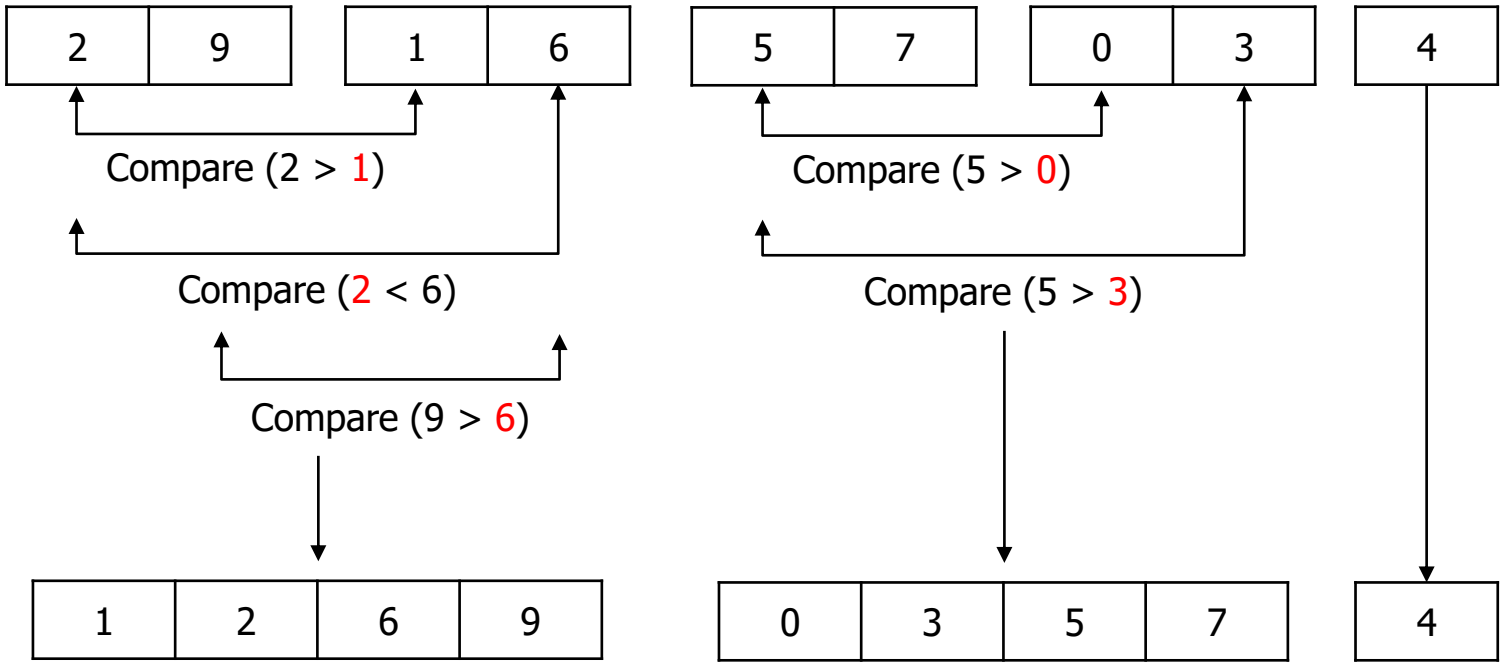
- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)



Sorted? O

Representative algorithms

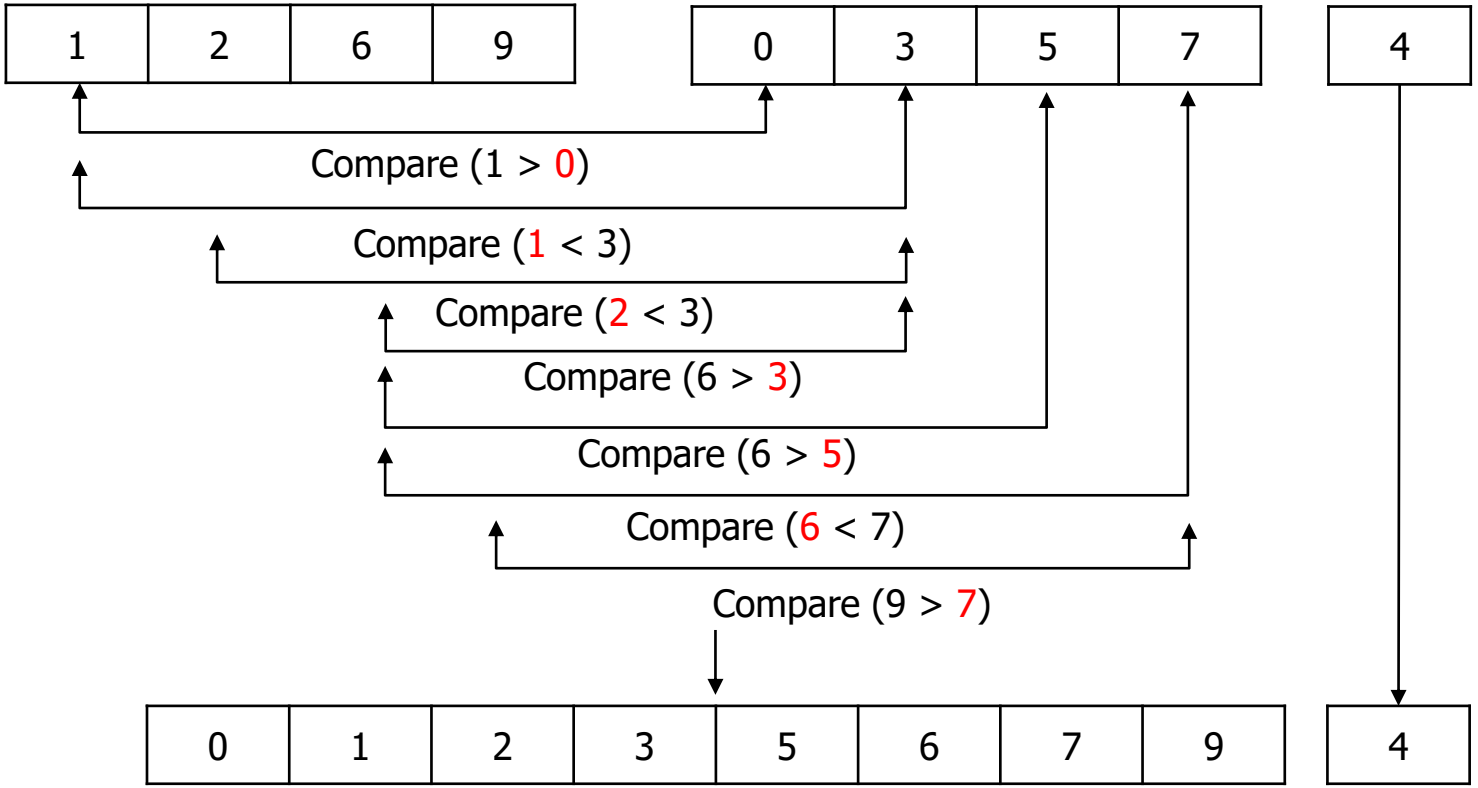
- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)



Sorted? O

Representative algorithms

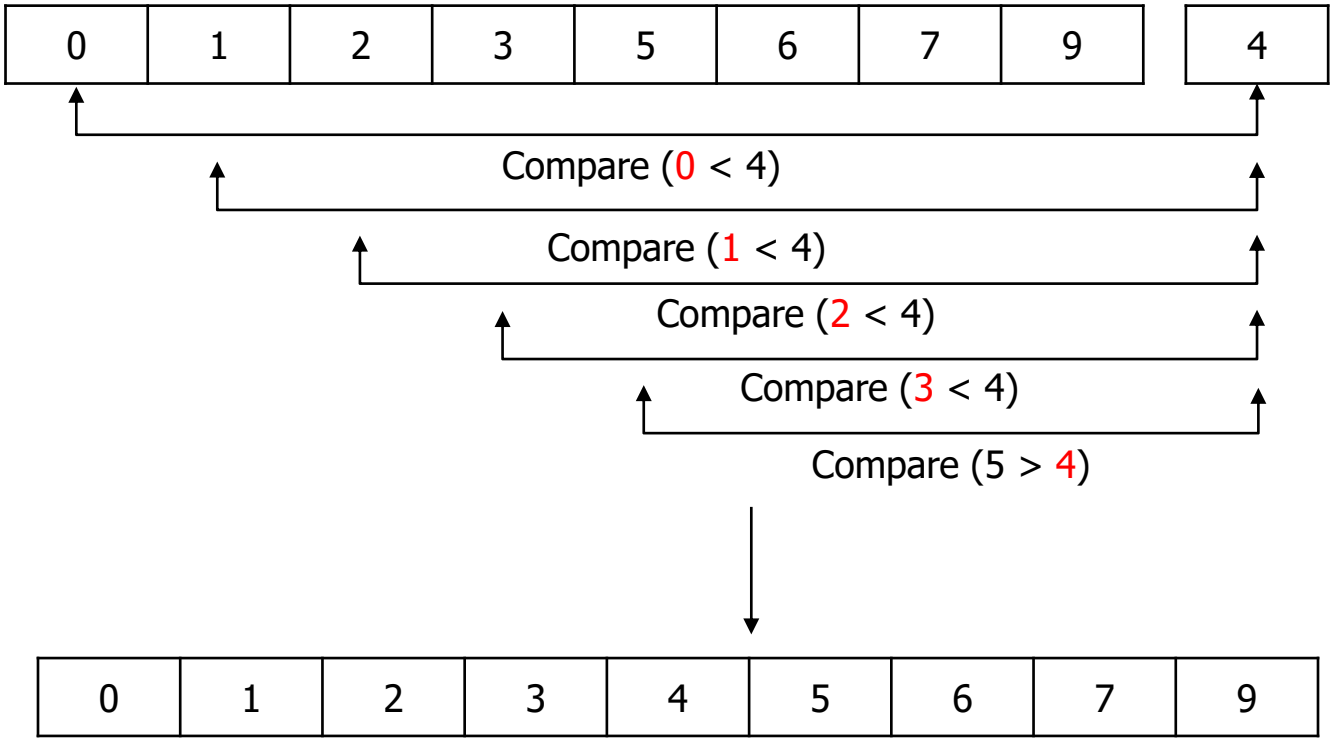
- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)



Sorted? O

Representative algorithms

- Merge sort algorithm (cont'd)
  - Example of merge sort algorithm
    - Recursively divide the list (conquer and combine)



Sorted? O

Representative algorithms

- Matrix multiplication algorithm
  - It divides the target matrices into submatrices
  - Then, it calculates the product of target matrices based on the solutions of these submatrices
  - Three steps:
    - Divide
      - A matrix is recursively divided into four submatrices, each of size $\frac{n}{2} \times \frac{n}{2}$
      - Base case is 1×1
    - Conquer
      - The product of the submatrices is calculated
    - Combine
      - Sum the products of the submatrices to generate the solution

Representative algorithms

- Matrix multiplication algorithm
  - Check lists
    - Are the solutions to subproblems derived using the same algorithm?
      - Yes (matrix multiplication)
    - Are each of the subproblems independent?
      - Yes

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Task: multiply the following matrices A and B
      - $n = 4$

$$A = \begin{bmatrix} 1 & 3 & 1 & 2 \\ 0 & 2 & 1 & 3 \\ 2 & 0 & 3 & 4 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 & 1 & 3 \\ 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Naïve approach: $O(n^3)$

$$\begin{bmatrix} 1 & 3 & 1 & 2 \\ 0 & 2 & 1 & 3 \\ 2 & 0 & 3 & 4 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad X \quad \begin{bmatrix} 2 & 1 & 1 & 3 \\ 1 & 4 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Naïve approach: $O(n^3)$

```
MATRIX MULTIPLY(A, B):
  n = A.rows
  C is a (n × n) empty matrix
  for i = 1 to n:
    for j = 1 to n:
      c_ij = 0
      for k = 1 to n:
        c_ij = c_ij + a_ik · b_kj
  return C
```

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Recursively divide the target matrices (divide)
      - How?

$$A = \begin{bmatrix} A_{11} & A_{12} \\ \begin{matrix}1 & 3\\ 0 & 2\end{matrix} & \begin{matrix}1 & 2\\ 1 & 3\end{matrix} \\ \begin{matrix}2 & 0\\ 0 & 1\end{matrix} & \begin{matrix}3 & 4\\ 0 & 1\end{matrix} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ \begin{matrix}2 & 1\\ 1 & 4\end{matrix} & \begin{matrix}1 & 3\\ 1 & 0\end{matrix} \\ \begin{matrix}0 & 0\\ 1 & 1\end{matrix} & \begin{matrix}1 & 3\\ 1 & 0\end{matrix} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$
$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$
$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$$

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Recursively divide the target matrices (divide)

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$

$$A_{11} = \begin{bmatrix} a_{11} & a_{12} \\ 1 & 3 \\ 0 & 2 \\ a_{21} & a_{22} \end{bmatrix} \quad B_{11} = \begin{bmatrix} b_{11} & b_{12} \\ 2 & 1 \\ 1 & 4 \\ b_{21} & b_{22} \end{bmatrix} \quad C_{11} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$A_{12} = \begin{bmatrix} a_{13} & a_{14} \\ 1 & 2 \\ 1 & 3 \\ a_{23} & a_{24} \end{bmatrix} \quad B_{21} = \begin{bmatrix} b_{13} & b_{14} \\ 1 & 3 \\ 1 & 0 \\ b_{23} & b_{24} \end{bmatrix}$$

Representative algorithms

- Matrix multiplication algorithm
  - Example of matrix multiplication algorithm
    - Recursively multiply the submatrices and sum the productions (conquer and combine)

$$c_{11} = a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{13}+a_{14}b_{23}$$
$$c_{12} = a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{14}+a_{14}b_{24}$$
$$c_{21} = a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{13}+a_{24}b_{23}$$
$$c_{22} = a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{14}+a_{24}b_{24}$$

Is it faster than the naïve approach?

If not, why?

Representative algorithms

- Strassen algorithm
  - It reduces the number of required multiplications
    - Remember that the product of A and B can be represented as the result of eight submatrix multiplications
    - Addition (+) has a lower complexity compared to multiplication (×)
    - Then, how to reduce the number of multiplications?

$A_{11} * B_{11}$ $A_{12} * B_{21}$

$A_{11} * B_{12}$ $A_{12} * B_{22}$

$A_{21} * B_{11}$ $A_{22} * B_{21}$

$A_{21} * B_{12}$ $A_{22} * B_{22}$

Representative algorithms

- Strassen algorithm
  - How to reduce the number of multiplications? (cont'd)
    - Define ten matrices derived from the submatrices of A and B

$$S_1 = B_{12} - B_{22}$$
$$S_2 = A_{11} + A_{12}$$
$$S_3 = A_{21} + A_{22}$$
$$S_4 = B_{21} - B_{11}$$
$$S_5 = A_{11} + A_{22}$$
$$S_6 = B_{11} + B_{22}$$
$$S_7 = A_{12} - A_{22}$$
$$S_8 = B_{21} + B_{22}$$
$$S_9 = A_{21} - A_{11}$$
$$S_{10} = B_{11} + B_{12}$$

Representative algorithms

- Strassen algorithm
  - How to reduce the number of multiplications? (cont'd)
    - Calculate seven matrices by multiplying submatrices recursively

$$P_1 = A_{11}S_1 \ (= A_{11}B_{12} - A_{11}B_{22})$$

$$P_2 = S_2B_{22} \ (= A_{11}B_{22}+A_{12}B_{22})$$

$$P_3 = S_3B_{11} \ (= A_{21}B_{11} + A_{22}B_{11})$$

$$P_4 = A_{22}S_4 \ (= A_{22}B_{21} - A_{22}B_{11})$$

$$P_5 = S_5S_6 \ (= A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22})$$

$$P_6 = S_7S_8 \ (= A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22})$$

$$P_7 = S_9S_{10} \ (= A_{21}B_{11} + A_{21}B_{12} - A_{11}B_{11} - A_{11}B_{12})$$

$$S_1 = B_{12} - B_{22}$$
$$S_2 = A_{11} + A_{12}$$
$$S_3 = A_{21} + A_{22}$$
$$S_4 = B_{21} - B_{11}$$
$$S_5 = A_{11} + A_{22}$$
$$S_6 = B_{11} + B_{22}$$
$$S_7 = A_{12} - A_{22}$$
$$S_8 = B_{21} + B_{22}$$
$$S_9 = A_{21} - A_{11}$$
$$S_{10} = B_{11} + B_{12}$$

Representative algorithms

- Strassen algorithm
  - How to reduce the number of multiplications? (cont'd)
    - Combine the seven matrices to obtain the solution
      - Only seven multiplications are required for each division

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 - P_3 + P_1 + P_7$$

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$
$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$
$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22}$$

$$P_1 = A_{11}S_1 \ (= A_{11}B_{12} - A_{11}B_{22})$$

$$P_2 = S_2B_{22} \ (= A_{11}B_{22} + A_{12}B_{22})$$

$$P_3 = S_3B_{11} \ (= A_{21}B_{11} + A_{22}B_{11})$$

$$P_4 = A_{22}S_4 \ (= A_{22}B_{21} - A_{22}B_{11})$$

$$P_5 = S_5S_6 \ (= A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22})$$

$$P_6 = S_7S_8 \ (= A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22})$$

$$P_7 = S_9S_{10} \ (= A_{21}B_{11} + A_{21}B_{12} - A_{11}B_{11} - A_{11}B_{12})$$

Representative algorithms

- Strassen algorithm
  - Summary
    - First, It recursively divides target matrices into four submatrices
      - Base case is 1×1
    - Second, define ten matrices based on the submatrices (S_1, … S_{10})
    - Third, calculates seven matrices (P_1, P_2, …, P_7)
    - Fourth, obtain the solution

Is it faster than the naïve approach?

Part 3

ALGORITHM ANALYSIS USING RECURRENCE FORMULA

Algorithm analysis using recurrence formula

- Recurrence formula
  - An equation that relates the terms of a sequence
  - Representing the next step (value) based on the previous steps (values)
  - It is defined recursively
    - Thus, it is primarily used for analyzing divide and conquer algorithm
  - Example of recurrence formula
    - Fibonacci numbers
      - $F(n) = F(n-1) + F(n-2), n \geq 2, F(0) = 0, F(1) = 1$
    - Factorial
      - $F(n) = n * F(n-1)$

```
Factorial (n):
   if (n=1):
      return 1
   return n * Factorial(n-1)
```

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm
  - Assume that the time for merge sort on a list of length n is T(n)
  - Then, T(n) = 2T(n/2) + c
    - Here, c is a post-processing time
  - Post processing time
    - Merge two sublists into a single list
      - Time complexity: $\Theta(n)$
  - Recurrence formula for the time complexity of merge sort algorithm
    - $T(n) = \begin{cases} \Theta(1) & , \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & \text{if } n > 1 \end{cases}$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Recurrence relation
    - $T(n) = 2T\left(\frac{n}{2}\right) + n$

      $= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n \; = 2^2T\left(\frac{n}{2^2}\right) + 2n$

      $= 2^2\left(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n \; = 2^3T\left(\frac{n}{2^3}\right) + 3n$

      …

      $= 2^kT\left(\frac{n}{2^k}\right) + kn$

      - Here, k denotes the number of recurrence
      - For the simplicity of asymptotic analysis, assume $n = 2^k$ $(k > 0)$
      - Then, $k = \log_2 n$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Recurrence relation
    - $T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$

$$= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n \log_2 n$$

$$= nT(1) + n \log_2 n$$

$$= n + n \log_2 n$$

$$\cong O(n \log_2 n)$$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: substitution method

> Asymptotic complexity of T(n)=2T(n/2) + n is O(nlogn)
>
> For sufficiently large n, there exists a positive constant c s.t. $T(n) \leq c\ n\log n$ is satisfied

- Boundary condition
  - There exists c that satisfies $T(2) \leq c\ 2\log 2$
  - T(1) is not possible because $c\ 1\log 1 = 0$

- Inductive assumption
  - Assume that $T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2}\right)\log\frac{n}{2}$ is satisfied

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: substitution method
    - $T(n) \leq 2T\left(\frac{n}{2}\right) + n$

$$\leq 2c\frac{n}{2}\log_2\frac{n}{2} + n = cn\log n - cn\log 2 + n$$

$$= cn\log n + (-c\log 2 + 1)n \quad (\because c \geq \frac{1}{\log 2})$$

$$\leq cn\log_2 n$$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: master method
    - It can immediately calculate the complexity of recurrence relation that follows a formal format by master theorem
    - What is the formal format?
      - Solving the 'n/b' problem 'a' times with a f(n) overhead
      - For merge sort algorithm, a and b are 2, and f(n) = n
    - Complexity depends on the constants, a, b, and the fuction f(n)

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: master method
    - Master theorem
      - Let $n^{\log_b a}$ be $h(n)$

① For $\varepsilon > 0$, if $\frac{f(n)}{h(n)} = O\left(\frac{1}{n^{\varepsilon}}\right)$, then $T(n) = \Theta(h(n))$

② For $\varepsilon > 0$, if $\frac{f(n)}{h(n)} = \Omega(n^{\varepsilon})$ and $af\left(\frac{n}{b}\right) \leq cf(n)$ is satisfied with $c < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$

③ If $\frac{f(n)}{h(n)} = \Theta(1)$, then $T(n) = \Theta(h(n) \log n)$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: master method
    - Approximated Master theorem
      - Let $n^{\log_b a}$ be $h(n)$

① If $\lim_{n\to\infty} \frac{f(n)}{h(n)} = 0$, then $T(n) = \Theta(h(n))$

② If $\lim_{n\to\infty} \frac{f(n)}{h(n)} = \infty$ and $af\left(\frac{n}{b}\right) \le f(n)$ is satisfied with sufficiently large n, then $T(n) = \Theta(f(n))$

③ If $\frac{f(n)}{h(n)} = \Theta(1)$, then $T(n) = \Theta(h(n)\log n)$

Algorithm analysis using recurrence formula

- Analysis of merge sort algorithm (cont'd)
  - Other approach: master method
    - Meaning of Master theorem
      - Let $n^{\log_b a}$ be $h(n)$

① If h(n) is more complex than f(n), then h(n) determines the complexity

② If f(n) is more complex than h(n), then f(n) determines the complexity

③ If h(n) and f(n) are similar, then the complexity is h(n)logn

Summary

- Divide and conquer
  - Three steps
    - Divide
      - Base case
    - Conquer
    - Combine

- Representative algorithms
  - Merge sort algorithm, Strassen algorithm

- Recurrence formula
  - Recurrence relation, Substitue method, Master method

Questions?

SEE YOU NEXT TIME!