Algorithms and Lab (CSE130)

Introduction to Algorithms

Muhammad Tariq Mahmood

tariq@koreatech.ac.kr
School of Computer Science and Engineering

Note: These notes are prepared from the following resources.

- (main text)Foundations of Algorithms, by Richard Neapolitan and Kumarss Naimipour
- Python Algorithm (파이썬 알고리즘) by Y.K. Choi (2021) (Korean)
- Introduction to the Design and Analysis of Algorithms (3rd Edition) by Anany Levitin
- Introduction to Algorithms, (3rd Edition) by By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
- https://www.geeksforgeeks.org

Contents

Course Overview

Course Objectives

- Understand algorithms for the most common problems(searching, sorting, shortest path, string manipulations, scheduling) .
- Analyze the performance of algorithms (asymptotically).
- Apply methods for algorithm analysis.
- Learn important algorithmic design paradigms.
- Apply important algorithmic design paradigms in problem solving.
- Develop efficient algorithms for common science and engineering problems.

Course Overview (cont...)

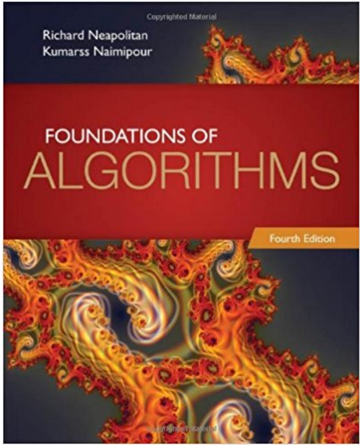
Course Prerequisites: It is assumed that you already have studied the following courses

- Data Structures
- Programming C/C++ or JAVA or Python
- Discrete mathematics

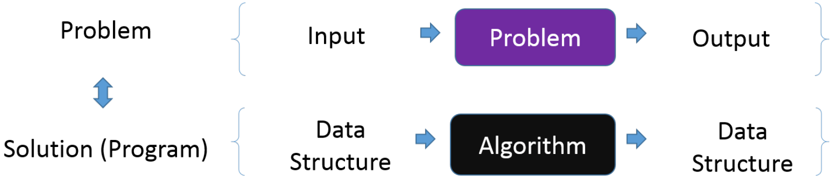
Grading Scheme

- Midterm Exam : 30%
- Final Exam : 30%
- Exercises/Homework's/Labs : 30%
- Participation, Q/A sessions : 10%

- Main Textbook: Foundations of Algorithms, Fourth Edition, Richard Neapolitan, Kumarss Naimipour
- Additional References:
- Python Algorithm (파이썬 알고리즘) by Y.K. Choi (2021) (Korean)
- Introduction to Algorithm by Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, 3rd ed. Cambridge, MA: MIT Press.
- Introduction to the design and analysis of algorithms by Anany Levitin, Pearson Addison Wesley.
- https://www.geeksforgeeks.org



What is an Algorithm?

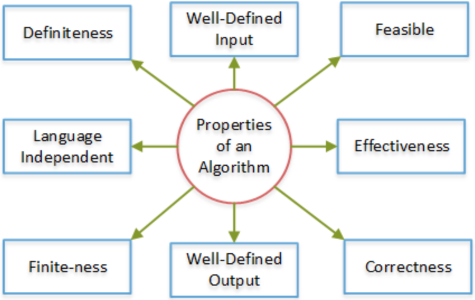


- A computer **algorithm** is a detailed **step-by-step method** for solving a **problem** using a computer.
- An algorithm is a **sequence of unambiguous instructions** for solving a **problem**, i.e., for obtaining a required output for any **legitimate input** in a finite amount of time.
- An algorithm is a **well-defined computational procedure** that takes some value or a set of values as **input** and produces some value or a set of values as **output**.
- Applying a technique to solve a **problem** results in a **step by step procedure**. This step by step procedure is called an algorithm for the problem.

What is an Algorithm? (cont...)

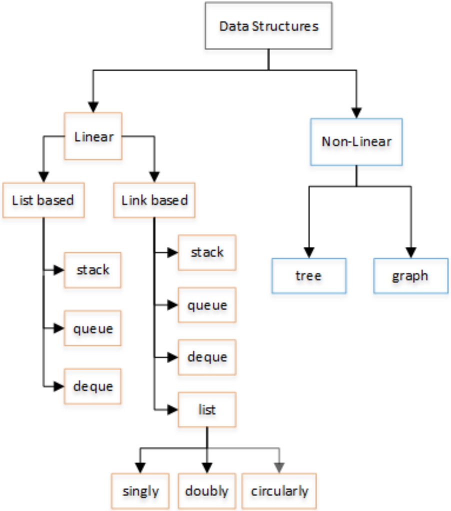
Properties of an algorithm:

- Well-Defined Input - An algorithm takes zero or more number of input values.
- Well-Defined Output - An algorithm produces an output as result.
- Definiteness - Every statement/instruction in an algorithm must be clear and unambiguous
- Finite-ness - For all different cases, the algorithm must produce result within a finite number of steps.
- Effectiveness - Every instruction must be basic enough to be carried out
- Feasible: The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources.
- Correctness: Correct set of output values must be produced from the each set of inputs.
- Language Independent: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language



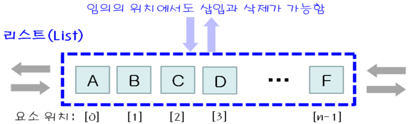
Relationship between Data structures and Algorithms

- Data structures are entities designed to hold the data that is used by the algorithms.
- Each data structure has a way to store the elements in memory and functions that will help you to manipulate the data stored efficiently.
- Some data structures are better than others to be used to solve determinate problems efficiently.
- Note that, $pragram = DataStructure + Algorithm$

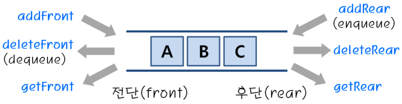


Linear Data Structures

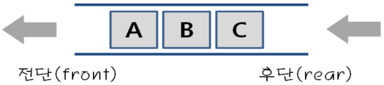
- List is a more general data structure. Items can be inserted and deleted at any position. Deque, Queue and Stack are special cases of List data structure



- Deque is a specific data structure. Items can only be inserted and deleted at rear and front ends



- Queue is a more specific data structure. Items can only be inserted at rear end and can be deleted at front end

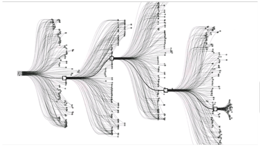


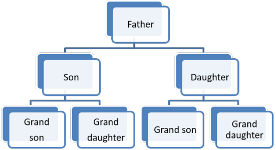
- Stack is also a more specific data structure. Items can be inserted and deleted one end

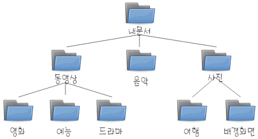
What is an Algorithm? (cont...)

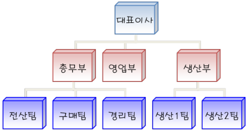
Tree Data Structures

- There are a number of applications where linear data structures are not appropriate.
- For example, applications that require searching, linear data structures are not suitable.
- A linear linked list will not be able to capture the tree-like relationship with ease.
- Tree data structure is useful to organize complex data and it has numerous applications
- It has a hierarchical structure





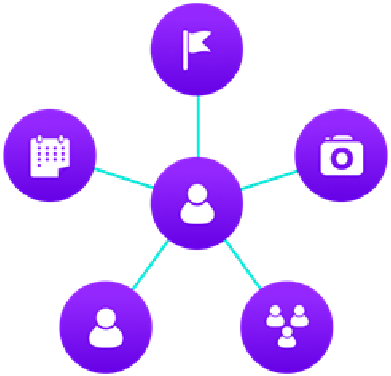




  - **Tree** is finite set of one or more nodes such that
    1. there is a special node called root
    2. remaining nodes are partitioned into $n \geq 0$ disjoint trees $T_1, T_2, \cdots, T_n$ where each of these is a tree; we call each T_i subtree of the root
  - A tree is **Acyclic graph**(a graph that contains no cycle)

Graph Data Structures

- A graph data structure is a collection of nodes that have data and are connected to other nodes.
- Example: On facebook, everything is a node including User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.
- Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.



- All of facebook is then a collection of these nodes and edges. This is because facebook uses a graph data structure to store its data.

Problems

What is a Problem?

1. A problem is a question to which we seek an answer.
2. A problem may contain variables that are not assigned specific values in the statement of the problem.
3. These variables are called parameters to the problem.
4. A solution to an **instance of a problem** is the **answer (algorithm)** to the question asked by the problem.

Examples

- Is the key x in the array S of n keys? (Searching Problem)
- Add all the numbers in the array S of n numbers (Arithmetic Problem)
- Sort n keys in nondecreasing order (Sorting Problem)
- Matrix Multiplication $\Rightarrow$ Determine the product of two n x n matrices (Matrix Multiplication Problem)
- Determine whether x is in the sorted array S of n keys. (Sorting Problem)
- Determine the nth term in the Fibonacci sequence (Arithmetic Problem)

Problems (cont...)

- Determine whether x is in the sorted array S of size n. (Searching Problem)
- Sort n keys in nondecreasing order (Sorting Problem)
- Merge two sorted arrays into one sorted array.(Merging Problem)
- Partition $\Rightarrow$ Partition the array S for Quicksort (Partition Problem)
- Determine the product of two n x n matrices where n is a power of 2. (Matrix Multiplication Problem)
- Large Integer Multiplication $\Rightarrow$ Multiply two large integers, u and v. (Integer Manipulation Problem)
- Compute the binomial coefficient (Computing Binomial Coefficients)
- Compute the shortest paths from each vertex in a weighted graph to each of the other vertex. The weights are nonnegative numbers (Shortest Path Problem)
- Print the intermediate vertices on a shortest path from one vertex to another vertex in a weighted graph. (Shortest Path Problem)

Problems (cont...)

- Determining the minimum number of elementary multiplications needed to multiply n matrices (Matrix Multiplication Problem)
- Print the optimal order for multiplying n matrices. (Matrix Multiplication Problem)
- Determine the node containing a key in a binary search tree. It is assumed that the key is in the tree (Searching Problem)
- Determine an optimal binary search tree for a set of keys, each with a given probability of being the search key. (Searching Problem)
- Build an optimal binary search tree (Searching Problem)
- Determine an optimal tour in a weighted, directed graph. The weights are nonnegative numbers. (Optimization Problem)
- Determine a minimum spanning tree (Minimum spanning Tree Problem)
- Determine the shortest paths from v1 to all other vertices in a weighted, directed graph. (Shortest Path Problem)

Problems (cont...)

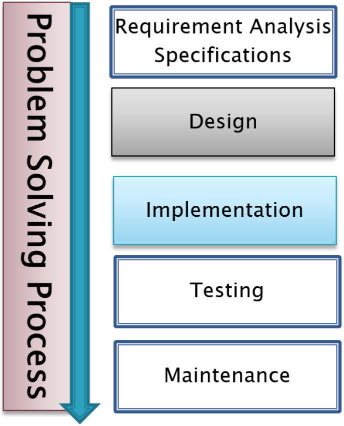
- Determine the schedule with maximum total profit given that each job has a profit that will be obtained only if the job is scheduled by its deadline. (Scheduling Problem)
- Determine a binary character code by constructing a binary tree corresponding to an optimal code (Encoding Problem)
- Position n queens on a chessboard so that no two are in the same row, column, or diagonal. (Searching Problem)
- Given n positive integers (weights) and a positive integer W, determine all combinations of the integers that sum to W. (Searching Problem)
- Determine all ways in which the vertices in an undirected graph can be colored, using only m colors, so that adjacent vertices are not the same color. (coloring Problem)
- Determine all Hamiltonian Circuits in a connected, undirected graph. (Searching Problem)
- Let n items be given, where each item has a weight and a profit. The weights and profits are positive integers. Furthermore, let a positive integer W be given. Determine a set of items with maximum total profit, under the constraint that the sum of their weights cannot exceed W. (Optimization Problem)

Problems ⇔ Algorithms

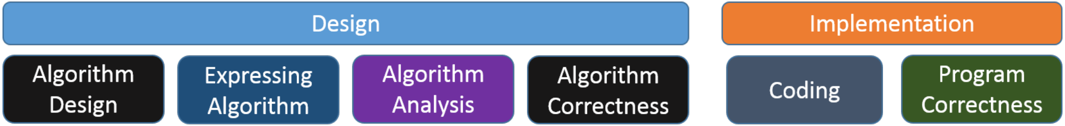
- Searching (Sequential Search, Binary Search)
- Sorting (exchange Sort, bubble sort, heap sort, quick sort, merge sort)
- Fibonacci Terms (Recursive and iterative algorithms)
- Binomial Coefficients (Divide-and-Conquer algorithm, Dynamic Programming algorithm)
- Shortest Paths (Floyd's Algorithm, Dijkstra's Algorithm)
- Sum-of-Subsets(Backtracking Algorithm, Branch-and-Bound)
- Minimum spanning Tree (Prim's Algorithm, Kruskal's Algorithm)
- Encoding Problems (Huffman's Algorithm)
- Traveling Salesperson Problem (Greedy Algorithm, Backtracking Algorithm, Branch-and-Bound)
- Knapsack Problem(Dynamic Programming Approach, Grady Approach, Backtracking Algorithm, Branch-and-Bound)

Problem Solving Process

- Requirements analysis and specifications: It involves in determining 1) input and output required to solve the problem. 2) other information, such as software usage, feasibility of the solution
- Design: It involves in 1) selecting appropriate data structures to organize the input/output data and designing algorithms, needed to process the data efficiently
- Implementation: It involves in selecting appropriate programming language and translating algorithms into well-structured, well-documented, readable code
- Testing: It involves in executing the software and correcting errors and proving correctness of algorithms and modifying algorithms
- Maintenance: It involves in modifying software to improve performance and adding new features, etc



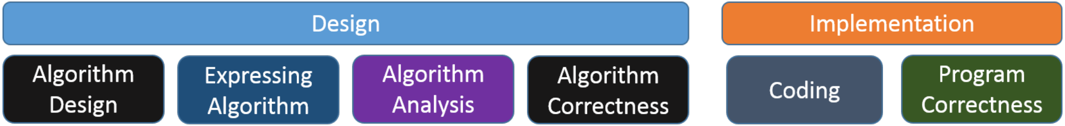
Design and Implementation Phases



Algorithm design techniques

- There are general approaches to construct efficient solutions to problems. They provide templates suited to solving a broad range of diverse problems.
  - Brute Force
  - Divide and conquer
  - Dynamic programming
  - Greedy approach
  - State space search techniques

Design and Implementation Phases

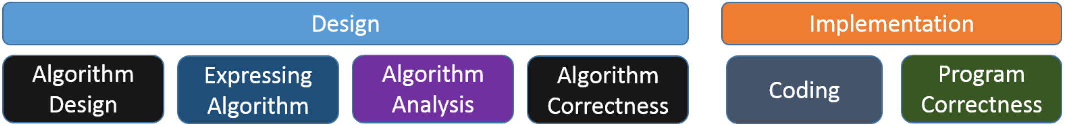


Expressing Algorithms (Pseudocode)

- Pseudocode specifies the steps required to accomplish the algorithm.
- Pseudocode cannot be compiled nor executed, and there are no syntax rules.
- Can be written any human language
- Example: getting and printing records from a file

```
procedure Example()
    open subscriber file
    Get a record
    while more records do
        if count > 3 then
            Output the record
        else
            Get another record
        end if
    end while
end procedure
```

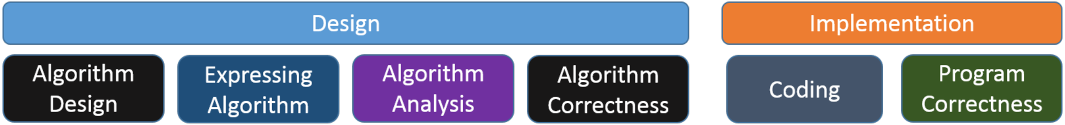
Design and Implementation Phases



Algorithm Analysis

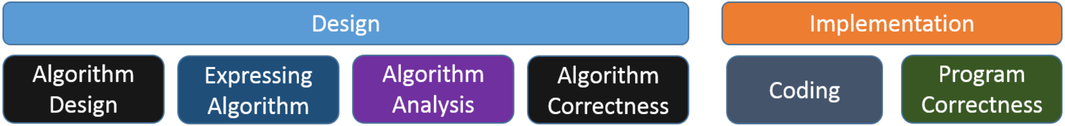
- The analysis of algorithms is to determine the computational complexity in terms of time and storage.
- Two approaches: Empirical/Experimental analysis and Theoretical/formal analysis
- Empirical/Experimental analysis:
  - It compares the time of execution for various algorithms.
  - It requires implementation of algorithms before analysis for a specific problem
  - It depends on
    - Actual number of CPU cycles (Computer)
    - Number of instructions
    - Programming language
    - Programmer

Design and Implementation Phases



Algorithm Analysis

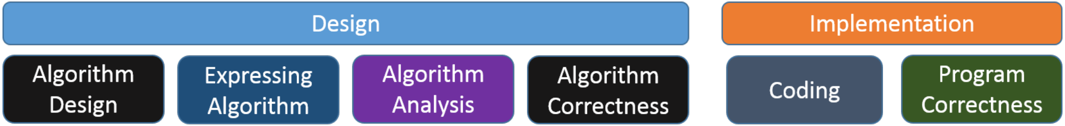
- The analysis of algorithms is to determine the computational complexity in terms of time and storage.
- Two approaches: Empirical/Experimental analysis and theoretical/formal analysis
- Theoretical/Formal analysis:
  - It involves determining a function (the time complexity function) that relates the algorithm's **input size** to the number of steps it takes or the number of storage locations it uses (its space complexity function).
  - It is common to estimate algorithm complexity in the asymptotic sense i.e., using order notations, Big-O notation, Big-omega(Ω) notation and Big-theta(Θ) notation
  - It dose not depend on
    - Actual number of CPU cycles (Computer)
    - Number of instructions
    - Programming language
    - Programmer



Algorithm Correctness Analysis

- This means to verify if the algorithm leads to the correct solution of the problem after a finite number of processing steps.
- Two common strategies: Experimental analysis and Formal analysis
- **Experimental analysis (Program Correctness):**
  - The main advantage of this approach is its simplicity.
  - The main disadvantage is the fact that testing cannot cover always all possible instances of input data (it is difficult to know how much testing is enough).

Design and Implementation Phases



Algorithm Correctness Analysis

- This means to verify if the algorithm leads to the correct output of the problem instance (specific input) after a finite number of processing steps.
- Two common strategies: Experimental Analysis and Formal Analysis
- **Formal Analysis (Algorithm Correctness):**
    - The main advantage of this approach is that if it is rigorously applied it guarantees the correctness of the algorithm.
    - The main disadvantage is the difficulty of finding a proof, mainly for complex algorithms.
    - A common technique for proving correctness is to use **mathematical induction** because an algorithm's iterations provide a natural sequence of steps needed for such proofs

Example-1: Add Array Members

- **Problem:** Add all the numbers in the array S of n numbers.
- **Inputs:** positive integer n, array of numbers S indexed from 1 to n.
- **Outputs:** *sum*, the sum of the numbers in S.
- Pseudo-code

Algorithm 1 Add Array Members (Pseudo-code)

```
procedure sumArray(integer n, number S[])
    integer i
    number sum
    sum = 0
    for (i = 1; i <= n; i++) do
        sum = sum + S[i]
    end for
    return sum
end procedure
```

Examples (cont...)

Example-2: Exchange Sort

- **Problem:** Sort n keys in nondecreasing order.
- **Inputs:** positive integer n, array of keys S indexed from 1 to n.
- **Outputs:** the array S containing the keys in nondecreasing order.
- Pseudo-code

Algorithm 2 Exchange Sort

```
procedure EXCHANGESORT(integer n, number S[])
    integer i, j
    for (i = 1; i <= n − 1; i++) do
        for (j = i + 1; j <= n; j++) do
            if (S[j] < S[i]) then
                exchange S[i] and S[j]
            end if
        end for
    end for
end procedure
```

Examples (cont...)

Example 3: A Recursive Algorithm for computing the n-th power of 2

- Problem: Computing the n-th power of 2.
- Inputs: n , a natural number .
- Outputs: n-th power of 2 .
- Pseudo-code

Algorithm 3 n-th power of 2

```
procedure POWER2(n)
    if (n == 0) then
        return 1;
    else
        return 2 * Power2(n − 1)
    end if
end procedure
```

Examples (cont...)

Example 4: Matrix Multiplication

- **Problem:** Determine the product of two *nxn* matrices.
- **Inputs:** a positive integer n, two-dimensional arrays of numbers A and B, each of which has both its rows and columns indexed from I to n.
- **Outputs:** a two-dimensional array of numbers C, which has both its rows and columns indexed from 1 to n, containing the product of A and B.
- Pseudo-code

Algorithm 4 Matrix Multiplication

```
procedure MATRIXMULTUPLY(integer n, number A[][],number B[][])
    integer i, j, k
    number C[][]
    for (i = 1; i <= n; i++) do
        for (j = 1; j <= n; j++) do
            C[i][j] = 0
            for (k = 1; k <= n; k++) do
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
            end for
        end for
    end for
    return C
end procedure
```

Example 5: nth Fibonacci Term (Iterative)

- **Problem:** Determine the nth term in the Fibonacci sequence.
- **Inputs:** a nonnegative integer n.
- **Outputs:** fib2, the *nth* term in the Fibonacci sequence.
- Pseudo-code

Algorithm 5 Algorithm 1.7: nth Fibonacci Term (Dynamic Programming)

```
procedure FIB2(integer n)
    integer i
    integer f[0..n]
    f[0] = 0
    f[1] = 1
    for (i = 2; i <= n; i++) do
        f[i] = f[i − 2] + f[i − 1]
    end for
    return f[n]
end procedure
```

Example 6: Recursive Algorithm for nth Fibonacci term

- **Problem:** Determine the n_{th} term in the Fibonacci sequence.
- **Inputs:** a nonnegative integer n.
- **Outputs:** *fib*, the n_{th} term of the Fibonacci sequence.

Algorithm 6 Recursive Algorithm for computing Fibonacci Sequence

```
procedure FIB(integer n)
    if (n <= 1) then
        return n
    else
        return fib(n − 1) + fib(n − 2)
    end if
end procedure
```

Example 7: Sequential Search

- **Problem:** Is the key x in the array S of n keys?
- **Inputs (parameters):** positive integer n, array of keys S indexed from 1 to n, and a key x.
- **Outputs:** location, the location of x in S (0 if x is not in S.)
- Pseudo-code

Algorithm 7 Sequential Search (Iterative)

```
procedure seqSearch(integer n, number S[],number x)
    integer location
    location = 1
    while location <= n && S[location]! = x) do
        location = location + 1
        if (location > n) then
            location = 0
        end if
    end while
    return location
end procedure
```

Example 8: Binary Search (Recursive)

- **Problem:** Determine whether x is in the sorted array S of size n.
- **Inputs:** positive integer n, sorted (nondecreasing order) array of keys S indexed from 1 to n, a key x.
- **Outputs:** location, the location of x in S (0 if x is not in S).
- Pseudo-code

```
procedure LOCATION(low,high)
    integer mid
    if (low > high) then
        return 0
    else
        mid = ⌊(low+high)/2⌋
        if (x == S[mid]) then
            return mid
        else
            if (x < S[mid]) then
                return location(low, mid − 1)
            else
                return location(mid + 1, high)
            end if
        end if
    end if
end procedure
```