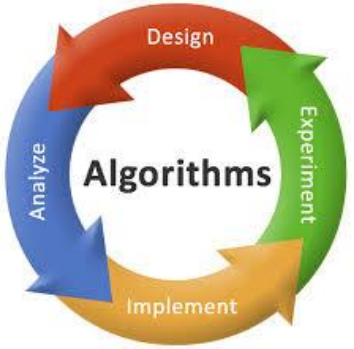
Java Programming AP Edition
U5C14 Searching, Sorting and Program Analysis

OVERVIEW OF ALGORITHMS

ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

Algorithm

How to solve a problem?



A systematic process consisting of an ordered sequence of steps, each step depending on the outcome of the previous one.

Methodology (Problem Solving)

Resource Estimation

- Time Complexity
- Memory Complexity

Algorithm Study

How Efficiently the algorithm can solve a problem?

Problem Solving

- Sorting
- Approximate Methods (ad hoc solutions)
- Randomized Solutions
- Dynamic Programming
- Linear Programming
- Cryptography
- Fourier Transform
- Computational Geometry
- Graph Theory

Algorithm Efficiency Analysis (Time Complexity)

Big-O Notation

NP-Completeness

Data Structure (Memory Complexity/Time Complexity Trade-offs)

Binary

B-Tree and other Tree

Stack, Queue, List, Map, Set, Heap, Sparse Matrix

Executing Time

Suppose two algorithms perform the same task such as search (linear search vs. binary search) and sorting (selection sort vs. insertion sort). Which one is better? One possible approach to answer this question is to implement these algorithms in Java and run the programs to get execution time. But there are two problems for this approach:

- First, there are many tasks running concurrently on a computer. The execution time of a particular program is dependent on the system load.
- Second, the execution time is dependent on specific input. Consider linear search and binary search for example. If an element to be searched happens to be the first in the list, linear search will find the element quicker than binary search.

Growth Rate

It is very difficult to compare algorithms by measuring their execution time. To overcome these problems, a theoretical approach was developed to analyze algorithms independent of computers and specific input. This approach approximates the effect of a change on the size of the input. In this way, you can see how fast an algorithm's execution time increases as the input size increases, so you can compare two algorithms by examining their **growth rates**.

Memory Complexity is Similar to Time Complexity Except for the Goal Function

T(n): Time complexity function — Time requirement when *n* grows

M(n): Memory complexity function — Memory requirement when *n* grows

Why Study Algorithm?

Remember the recursive versus iterative algorithm for Fibonacci number?

Poor algorithm leads to time-consuming solution and waste of memory.

Good algorithm leads to good solution which meets program latency requirement and does not take too much resource.

Computer Scientist's job is to find good solution for problems.