Lecture 2
Algorithm Analysis

고형석 교수

ko@graphics.snu.ac.kr

http://graphics.snu.ac.kr/~ko

Outline of Topics

- **What is Algorithm Analysis?**
- **Examples of Algorithm Running Times**
- **The Maximum Contiguous Subsequence Sum Problem**
  - Algorithm 1: A cubic algorithm
  - Algorithm 2: A quadratic algorithm
  - Algorithm 3: An *N* log *N* algorithm
  - Algorithm 4: A linear algorithm
- **Logarithmic Algorithms**
  - General principles
  - Binary Search

Finding the Minimum

Given an array of *N* items, find the smallest

What is Algorithm Analysis?

- **Finding out running time or space requirement**
- **Running time of an algorithm almost always depends on the amount of input: More input means more time. Thus the running time, *T*, is a function of the amount of input, *N*, or $T(N) = f(N)$.**
- **Actual running time also depends on**
  - the speed of the host machine
  - the quality of the compiler and optimizer
  - the quality of the program that implements the algorithm
  - <u>the basic fundamentals of the data structure and algorithm</u>
- **Typically, the last item is most important. Algorithm analysis usually means that.**

Finding the Minimum

Given an array of *N* items, find the smallest

- **Obvious algorithm is a sequential scan.**

Running Time:
Worst-case Vs. Average Case

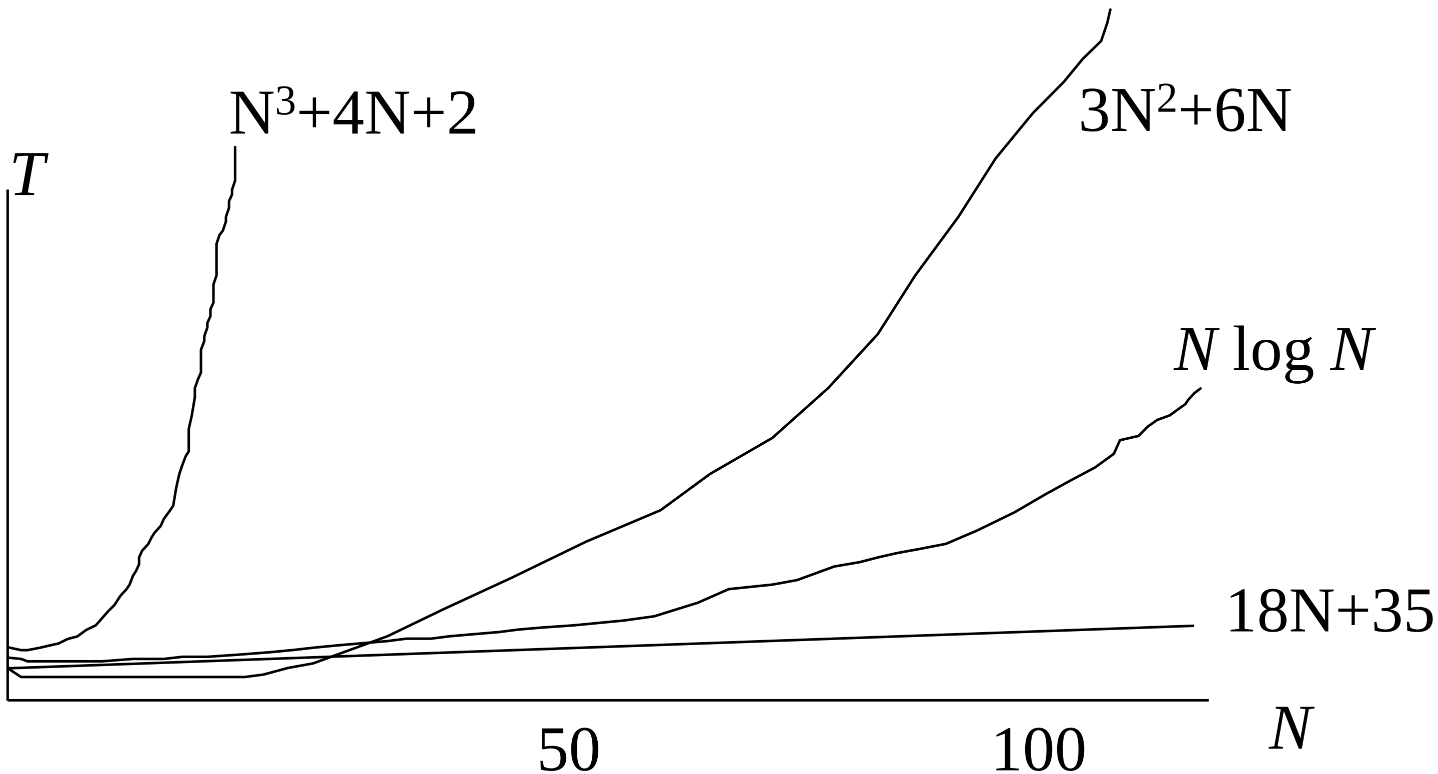
- **Worst-case running time is a bound over all inputs of size *N*. (Guarantee)**
- **Average-case running time is an average over all inputs of size *N*. (Prediction)**
  - Usually, difficult to define the distribution to compute the average cases
- **Best case running time**
  - Can be used to argue that the algorithm is really bad.

Finding the Minimum

Given an array of *N* items, find the smallest

- **Obvious algorithm is a sequential scan.**
  - What is worst, average, best case running time?

Examples of Running Time Functions



Notations for Time Bounds

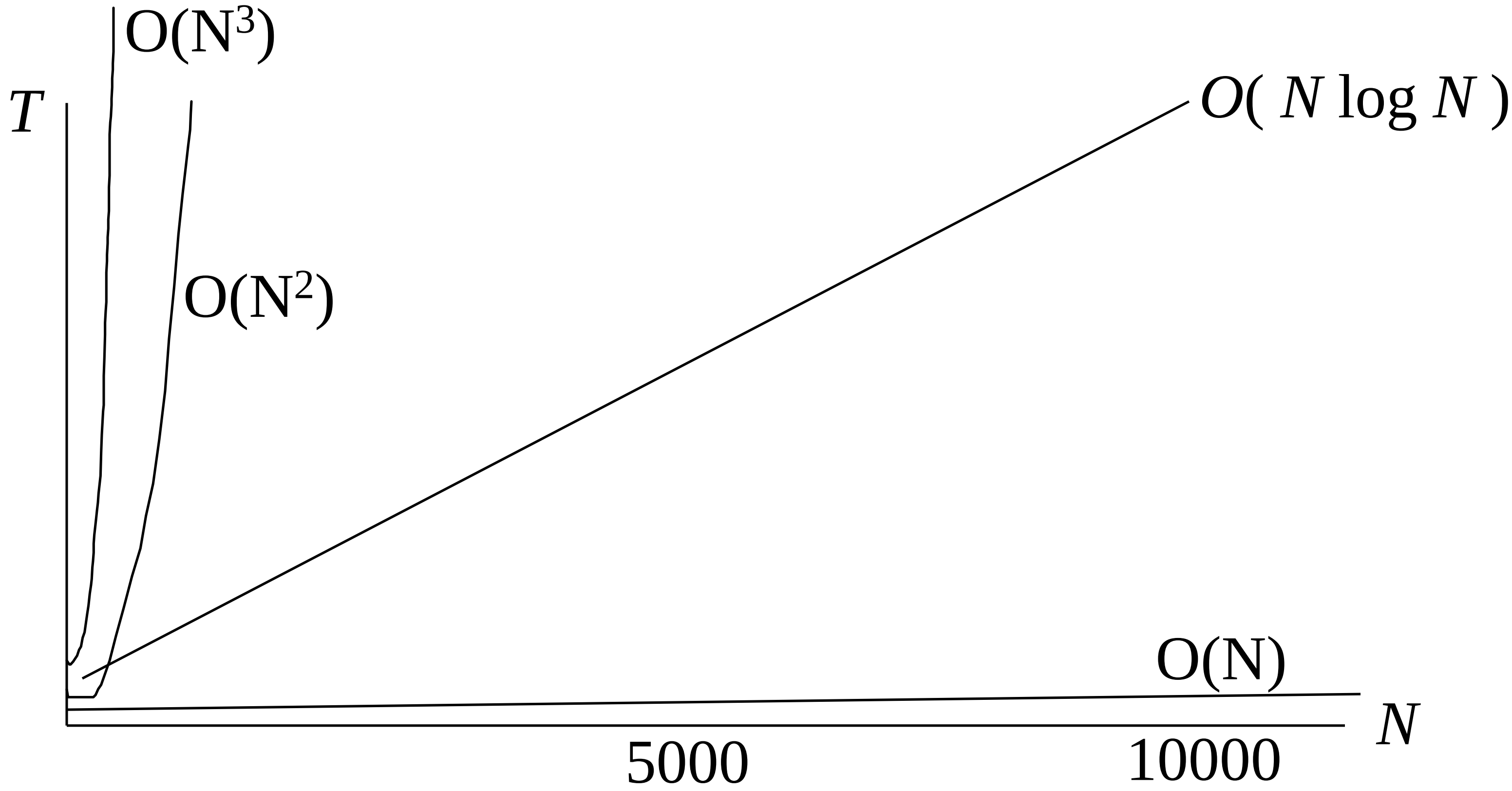
- **Big-Oh for upper bound:**
  - $T(N) = O(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq c \cdot f(N)$ when $N \geq n_0$.
- **Big-Omega for lower bound:**
  - $T(N) = \Omega(g(N))$ if there are positive constants c and n_0 such that $T(N) \geq c \cdot g(N)$ when $N \geq n_0$.
- **Big-Theta for precise bound:**
  - $T(N) = \Theta(h(N))$ iff $T(N) = O(h(N))$ and $T(N) = \Omega(h(N))$.
- **Small-Oh for UUUUPPER bound:**
  - $T(N) = o(p(N))$ iff $T(N) = O(p(N))$ and $T(N)$ is not $\Theta(p(N))$.

Finding the Minimum

Given an array of *N* items, find the smallest

- **Obvious algorithm is a sequential scan.**
  - What is worst, average, best case running time?
- **Running time is *O*(*N*) because we have to do a fixed amount of work for each element in the array.**
- **A linear algorithm is as good as we can hope for. Why?**
  - We have to examine every element in the array, a process that requires a linear time.

Running Time Functions



For large inputs, some running time functions are completely unusable.

Dominant Term Matters

- **Suppose we estimate $350N^2 + N + N^3$ with N^3.**
- **For $N = 10000$:**
  - Actual value is 1,003,500,010,000
  - Estimate is 1,000,000,000,000
  - Error in estimate is 0.35%, which is negligible.
- **For large *N*, dominant term is usually indicative of algorithm's behavior.**
- **For small *N*, dominant term is not necessarily indicative of behavior,** BUT**, typically programs on small inputs run fast, so we don't care anyway.**

Properties of Big-Oh

- **If T1(N) = O(f(N)) and T2(N) = O(g(N)), then**
  - T1(N) + T2(N) = O(max(f(N), g(N)))
    - Lower-order terms are ignored
  - T1(N)*T2(N) =O(f(N)*g(N))
- **O(c f(N)) = O(f(N)) for any constant c**
  - Constants are ignored!
- **In reality, constants and lower-order terms may matter, especially when the input size is small.**
- **Can you prove the above properties with the definitions?**

Lecture 2
Algorithm Analysis

The End