빅 데 이 터 혁 신 공 유 대 학

파이썬으로 배우는 데이터 구조

한동대학교 전산전자공학부

김영섭 교수

교육부 NRF 한국연구재단

Data Structures in Python
Chapter 2 - 2

- Performance Analysis
- **Big-O Notation**
- Big-O Properties
- Growth Rates
- Growth Rates Examples

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

그러므로 나의 사랑하는 자들아 너희가 나 있을 때 뿐 아니라 더욱 지금 나 없을 때에도 항상 복종하여 두렵고 떨림으로 너희 구원을 이루라 (Continue to work out your salvation with fear and trembling.) 빌2:12

나는 인애를 원하고 제사를 원하지 아니하며 번제보다 하나님을 아는 것을 원하노라 (호6:6)
하나님은 모든 사람이 구원을 받으며 진리를 아는데에 이르기를 원하시느니라 (딤전2:4)

그런즉 너희가 먹든지 마시든지 무엇을 하든지 다 하나님의 영광을 위하여 하라 (고전10:31)

Agenda & Reading

- **Big-O Notation**
  - **Asymptotic Analysis**
- Big-O Properties
  - Calculating Big-O

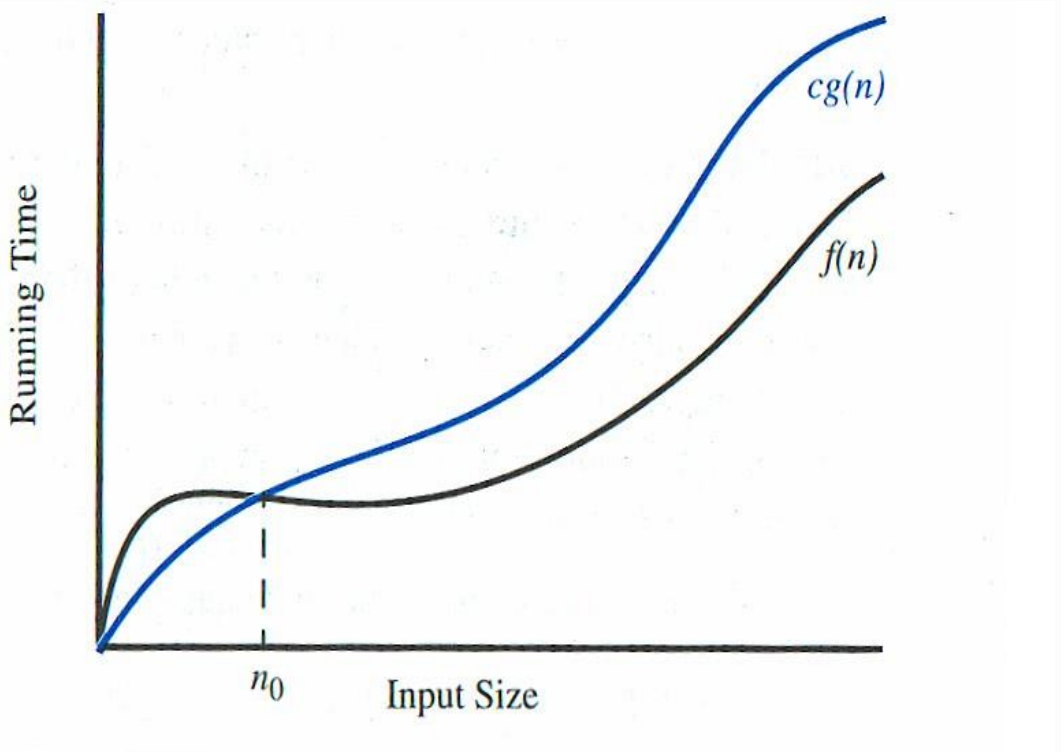
- References:
  - Textbook: Problem Solving with Algorithms and Data Structures
    - Chapter 3. Analysis
  - Textbook: www.github.idebtor/DSpy
    - Chapter 2.1 ~ 3

3 Big-O Definition

- Let $f(n)$ and $g(n)$ be functions that map non-negative integers to real numbers. We say that $\boldsymbol{f(n)}$ **is** $\boldsymbol{O(g(n))}$ if there is a real constant c, where $c > 0$ and an integer constant n, where $n_0 \geq 1$ such that $f(n) \leq c * g(n)$ for every integer $n \geq n_0$.

  - $f(n)$ describe the actual time of the program
  - $g(n)$ is a much simpler function than $f(n)$
  - With assumptions and approximations, we can use $g(n)$ to describe the complexity i.e., $\boldsymbol{O(g(n))}$

Big-O Notation is a mathematical formula that best describes an algorithm's performance.



3 Big-O Notation

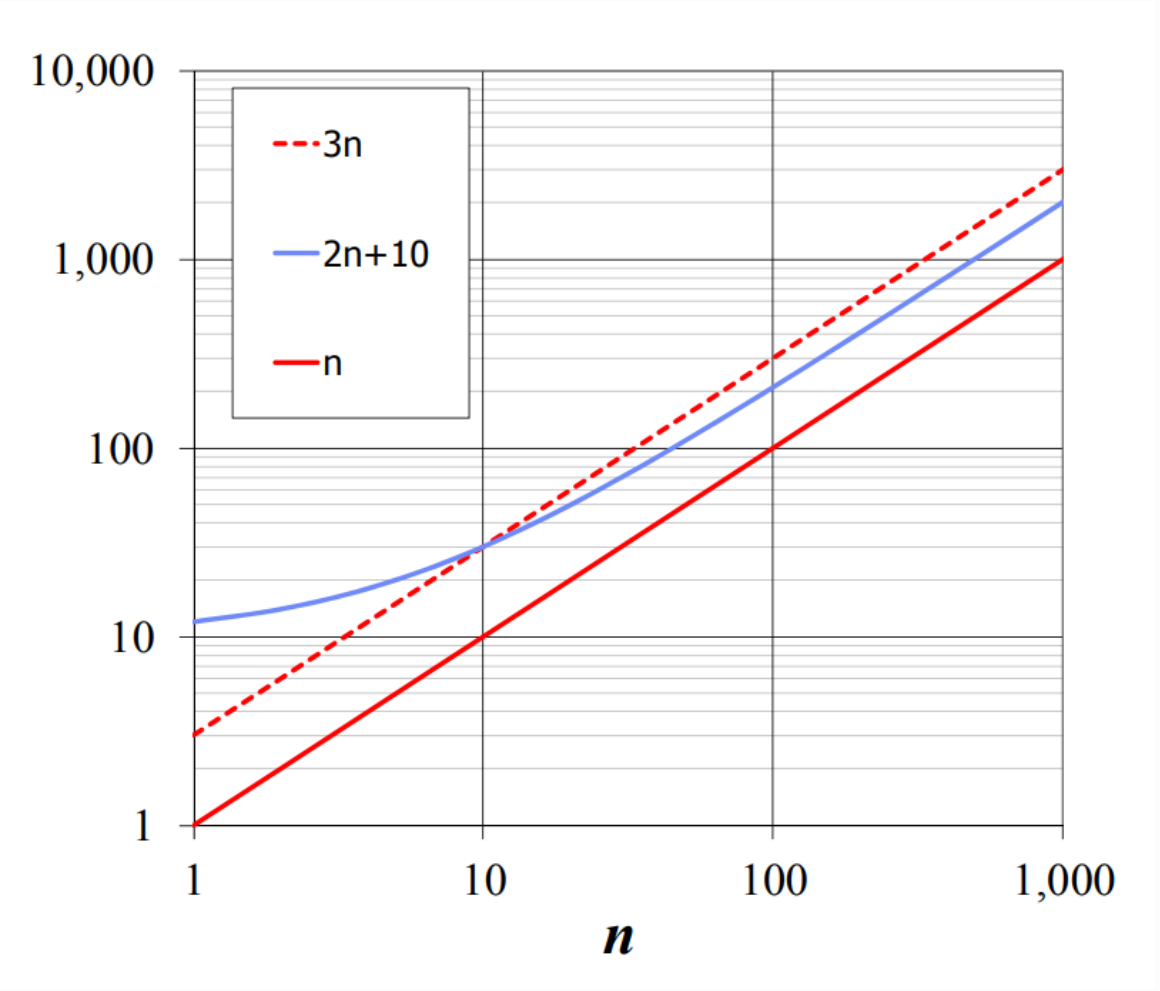
- We use Big-O notation (capital letter O) to specify the order of complexity of an algorithm.
  - $e.g.,\ O(n^2),\ O(n^3),\ O(n)$
  - If a problem of size n requires time that is directly proportional to n, the problem is $O(n)$ - that is, order n.
  - If the time requirement is directly proportional to n^2, the problem is $O(n^2)$, etc.

3 Big-O Examples

- Given functions $f(n)$ and $g(n)$, we say that **$f(n)$ is $O(g(n))$** if there are positive constants, c, and n_0 such that $f(n) \leq c * g(n)$ for every integer $n \geq n_0$.

- Example:
  T(n) = 2n + 10
  T(n) is O(n)

- Question:

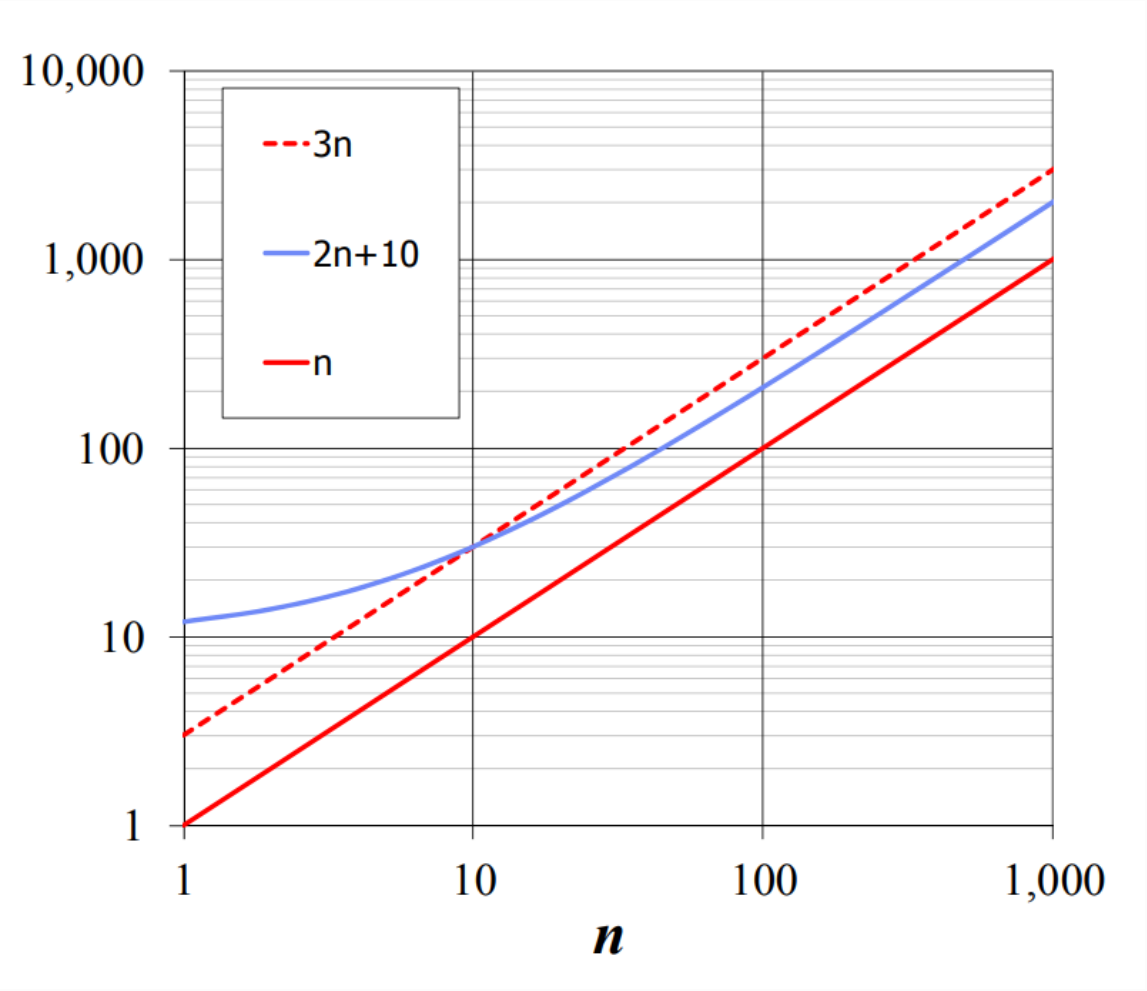


3 Big-O Examples

- Given functions $f(n)$ and $g(n)$, we say that $\boldsymbol{f(n)}$ **is** $\boldsymbol{O(g(n))}$ if there are positive constants, c, and n_0 such that $f(n) \leq c * g(n)$ for every integer $n \geq n_0$.

- Example:
  T(n) = 2n + 10
  T(n) is O(n)

- Question:
  - n_0
  - c
  - g(n)
  - $f(n) \leq c * g(n)$
  - $\boldsymbol{f(n)}$ **is** $\boldsymbol{O(g(n))}$

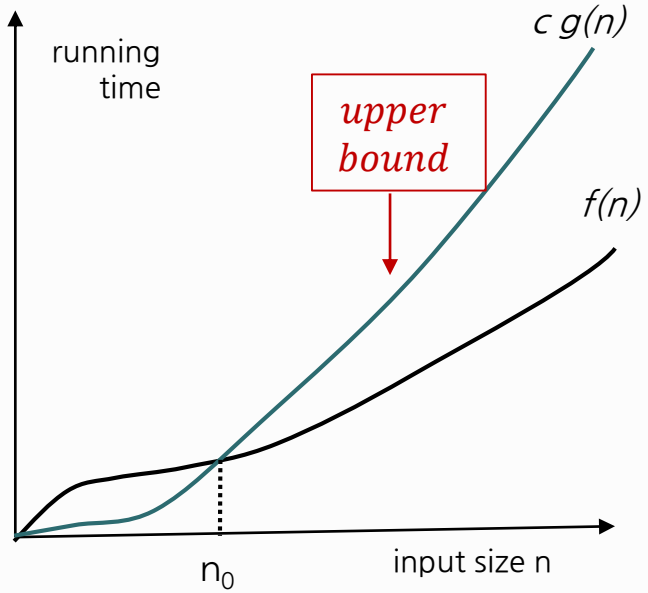


3 Big-O Examples

- Find c and n_0 to justify that the function $7n + 5$ is $O(n)$.

We must find c and n_0 such that

$7n + 5 \leq c\,n \qquad for\ n \geq n_0$



3 Big-O Examples

- Find c and n_0 to justify that the function $7n + 5$ is $O(n)$.

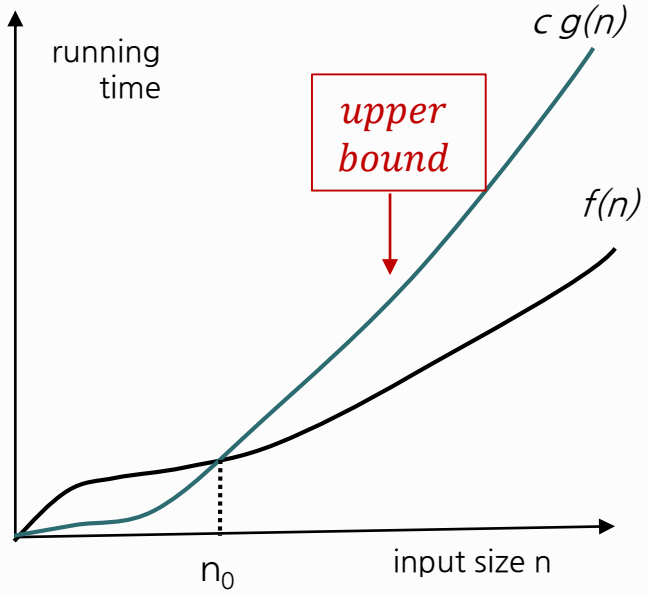
We must find c and n_0 such that

$7n + 5 \leq c\,n$ $\quad for\ n \geq n_0$

$7n + 5 \leq 7\,n + n$

$7n + 5 \leq 8\,n$ $\quad for\ n \geq n_0 = 5$

Therefore, $7n + 5 \leq c\,n$ for $c = 8$ and $n_0 = 5$, $g(n) = n$ and $O(n)$



3 Big-O Examples

- Find c and n_0 to justify that the function $7n + 5$ is $O(n)$.

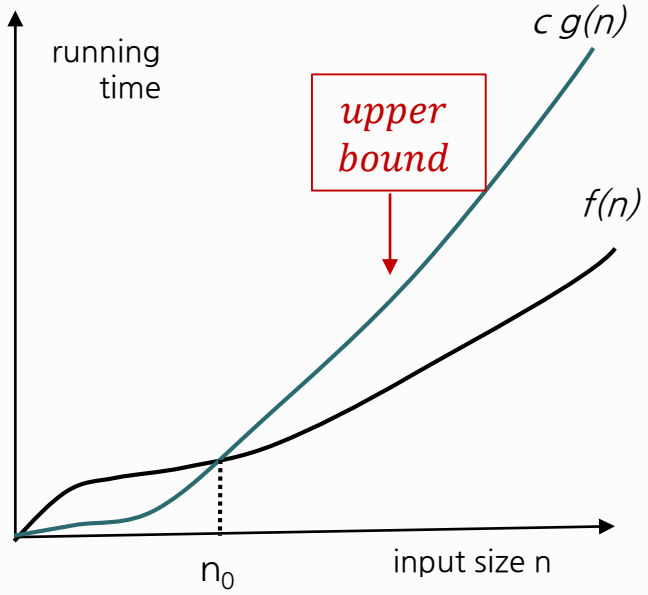
We must find c and n_0 such that

$7n + 5 \leq c\,n$ $\quad for\ n \geq n_0$

$7n + 5 \leq 7\,n + n$

$7n + 5 \leq 8\,n$ $\quad for\ n \geq n_0 = 5$

Therefore, $7n + 5 \leq c\,n$ for $c = 8$ and $n_0 = 5$, f(n) is O(n)



$7n + 5 \leq c\,n$ $\quad for\ n \geq n_0$

$7n + 5 \leq 12\,n$ $\quad for\ n \geq n_0 = 1$

Therefore, $7n + 5 \leq c\,n$ for $c = 12$ and $n_0 = 1$

g(n) = n, f(n) is O(n)

3 Big-O Examples

- Find c and n_0 to justify that the function $f(n) = 27n^2 + 16n$ is $O(n^2)$.

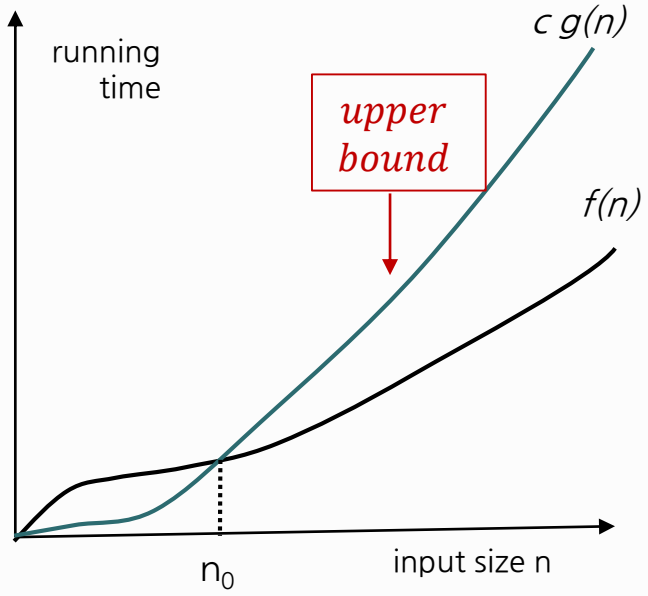
We must find c and n_0 such that

For $16n \leq n^2$

$27n^2 + 16n \leq 27n^2 + n^2$

$27n^2 + 16n \leq 28n^2$ $\quad for\ n \geq n_0 = 16$

Hence, c = 28 and n_0 = 16, Therefore, $g(n) = n^2, f(n)$ **is** $O(n^2)$.



$27n^2 + 16n$ is $O(n^2)$, we must find c and n_0 such that

$27n^2 + 16n \leq 43n^2$

$27n^2 + 16n \leq 43n^2$ $\quad for\ n \geq n_0 = 1$

Hence, c = 43 and n_0 = 1, Therefore, $g(n) = n^2, f(n)$ **is** $O(n^2)$.

3 Big-O Examples

- Suppose an algorithm requires
  - T(n) = 7n-2 operations to solve a problem of size n

$f(n) \leq c * g(n)$ for every integer $n \geq n_0$

7n-2 ≤ 7 * n for all $n_0 \geq 1$
i.e., c = 7, $n_0 = 1$

O(n)

  - T(n) = n^2- 3 * n + 10 operations to solve a problem of size n

n^2 - 3 * n + 10 < 3 * n^2 for all $n_0 \geq 2$
i.e., c = 3, $n_0 = 2$

O(n^2)

  - T(n) = $3n^3 + 20n^2 + 5$ operations to solve a problem of size n

$3n^3 + 20n^2 + 5 < 4 * n^3$ for all $n_0 \geq 21$
i.e., c = 4, $n_0 = 21$

O(n^3)

3 Big-O Examples

1) $3n + 2 =$

2) $3n + 3 =$

3) $100n + 6 =$

4) $10n^2 + 4n + 2 =$

5) $6 * 2^n + n^2 =$

6) $3n + 3 =$

7) $10n^2 + 4n + 2 =$

✖ *8)* $3n + 2 \neq O(1)$ *as* $3n + 2$ *is* ***not*** $\leq c$ *for any* c *and all* $n, n \geq n_0$.

✖ *9)* $10n^2 + 4n + 2 \neq O(n)$

Summary

- Big-O Notation is a mathematical formula that best describes an algorithm's performance.
- Big-O notation is often called the asymptotic notation **(점근적 표기법)** since it uses so-called the **asymptotic analysis (점근적 분석)** approach.
- Normally **we assume worst-case analysis**, unless told otherwise.
- In some cases, it may need to consider the best, worst and/or average performance of an algorithm

Data Structures in Python
Chapter 2 - 2

- Performance Analysis
- **Big-O Notation**
- Big-O Properties
- Growth Rates
- Growth Rates Examples

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University