[CSED233-01] Data Structure

Algorithm Analysis

Jaesik Park

POSTECH

Announcement

- Attendance
  - 9:30~9:45 - late (-1 pts from your scores)
  - 9:45~ - absent (-3 pts from your scores)
  - 3 absent without proper reason – you may get failure
- Office hour
  - Tuesday and Thursday
  - 1PM~2PM, via Online (meeting link will be announced on PLMS)
- Programming assignment #1
  - Assignment was announced: 3/2
  - Due date: 3/21 midnight
  - It will include basic concepts we've learned so far
  - Don't be afraid
    - We will provide template code and instructions
    - Will be easy to follow
    - DO NOT COPY your friend's code

Can we determine which algorithm is better?

Algorithm

- A step-by-step procedure for solving a problem in a finite amount of time

- How to compare two algorithms?
- One measure is efficiency
  - Running time
    - Time complexity
  - Space requirements
    - Space complexity

- Two ways of comparison
  - Empirical studies (programming & testing)
  - Theoretical analysis

Empirical Studies

- Programming & testing
  - Write a program implementing the algorithm
  - Run the program with inputs of varying size
  - Measure the efficiency

- Limitations
  - Much effort to implement the algorithm
  - Results may not apply to other inputs which are not included in the experiment
  - For fair comparison, the same H/W and S/W environments must be used

Theoretical Analysis

- High-level description of the algorithm instead of an implementation
  - Running time – as a function of the input size n
  - Consider all possible inputs

- Limitations
  - You input size might be naturally constrained, so don't need to think!

- Allow us to evaluate the speed of an algorithm independent of the HW & SW environments

Best, Worst, and Average Cases

- Different inputs of a given size can require different amount of running time
  - Best case
    - At least, takes this much
  - Average case
    - Usually, it takes this much
    - Difficult to determine, and often *infeasible*
  - Worst case
    - It could take up to this
    - *Easier* to analyze
    - Crucial to *interactive* applications

- Focusing on the worst-case running time, here

Asymptotic Analysis

- Asymptotic analysis, also known as asymptotics, is a method of describing limiting behavior.
  - If $f(n) = n^2 + 3n$, then as n becomes very large, the term 3n becomes insignificant compared to n^2.
  - The function f(n) is said to be "asymptotically equivalent to n^2, as $n \to \infty$".

- Let T(n) the running-time function that maps an input size **N** to a running time **R**

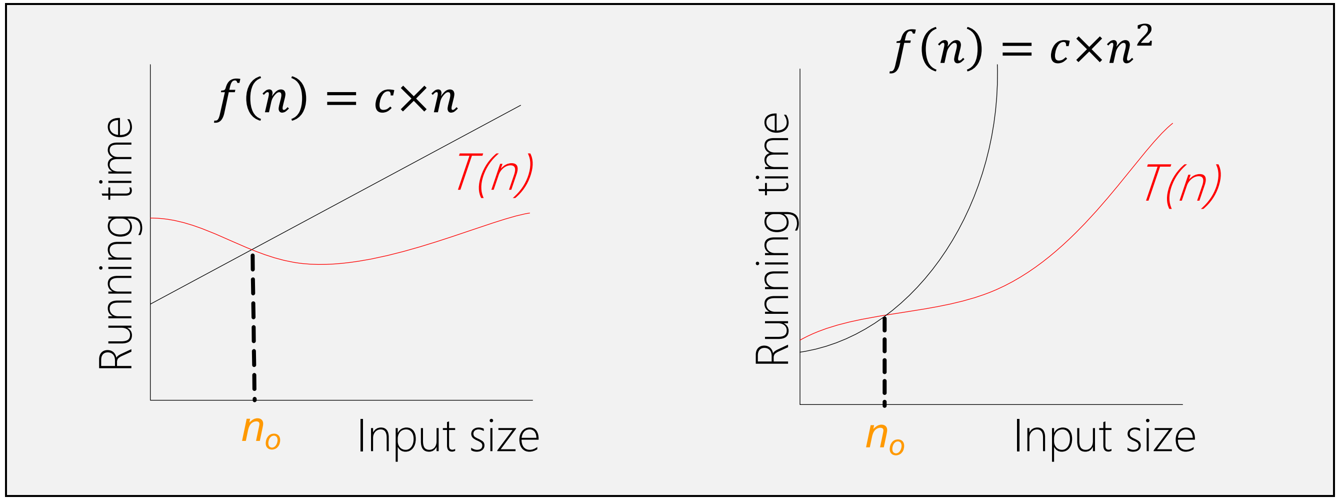
- To capture the growth rate behavior of *T(n)* in the long run
  - Worst case ➔ upper bound: Big-Oh
  - Average case ➔ Equal: Big-Theta
  - Best case ➔ Lower bound: Big-Omega

Big-O Notation

- An algorithm is $O(f(n))$ if there exist a constant $c > 0$ & an integer constant $n_0 \geq 1$ such that

$$T(n) \leq c \cdot f(n) \text{ for all } n \geq n_0$$

  - Then, we write $T(n) \in O(f(n))$, or $T(n) = O(f(n))$
- Upper bound on the growth rate of $T(n)$



Big-O Notation: Examples

- Example: $T(n) = (n+1)^2$ is $O(n^2)$

$$
\begin{aligned}
T(n) &= (n+1)^2 = n^2 + 2n + 1 \\
&\le n^2 + 2n^2 + n^2 = 4n^2 \text{ for all } n \ge 1 \\
\end{aligned}
$$

$$\text{Thus pick } c = 4 \text{ and } n_0 = 1$$

- More examples:

$$3n^3 \in O(n^3)\text{: tight bound} \Leftrightarrow 3n^3 \in O(n^4)\text{: loose bound}$$

$$3n^3 + 2n^2 + 8 \in O(n^3)$$

$$2^{100} \in O(1)$$

$$3\log(n) + 5 \in O(\log(n))$$

Properties of Big-Oh

- Addition rule:

$$T_1(n) \in O(f(n)) \text{ and } T_2(n) \in O(g(n)) \Rightarrow T_1(n) + T_2(n) \in O(\max\{f(n), g(n)\})$$

- Product rule:

$$T_1(n) \in O(f(n)) \text{ and } T_2(n) \in O(g(n)) \Rightarrow T_1(n) \cdot T_2(n) \in O(f(n) \cdot g(n))$$

- Others

$$\text{For any constant } a > 0,\ T(n) \in O(f(n)) \Rightarrow a \cdot T(n) \in O(f(n))$$

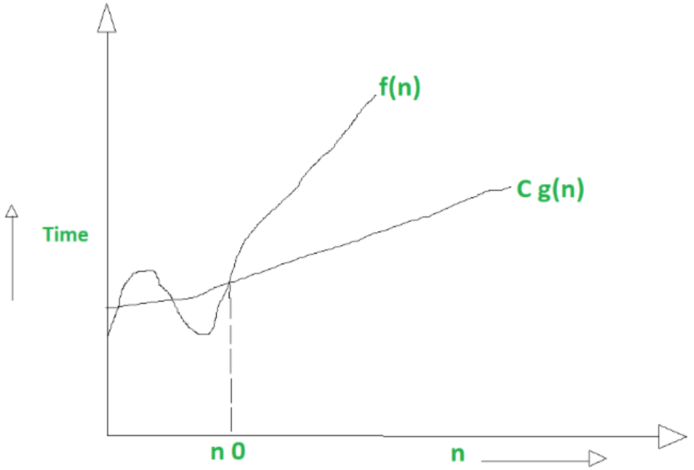
$$T(n) \in O(f(n)) \text{ and } f(n) \in O(g(n)) \Rightarrow T(n) \in O(g(n))$$

$$T(n): \text{polynomial of degree } d \Rightarrow T(n) \in O(n^d)$$

Big-Omega & Big-Theta

- *T(n)* is *Ω(f(n))* if there exist a constant $c > 0$ & an integer constant $n_0 \geq 1$ such that

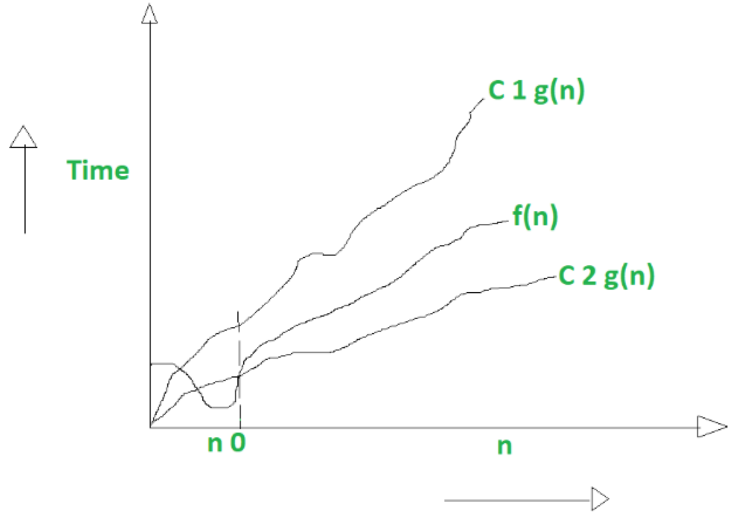
$$T(n) \geq c \cdot f(n) \text{ for all } n \geq n_0$$



  - Lower bound - asymptotically greater than or equal to *f(n)*

- *T(n)* is *Θ(f(n))* if T(n) is *O(f(n))* and *Ω(f(n))*

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n) \text{ for all } n \geq n_0$$

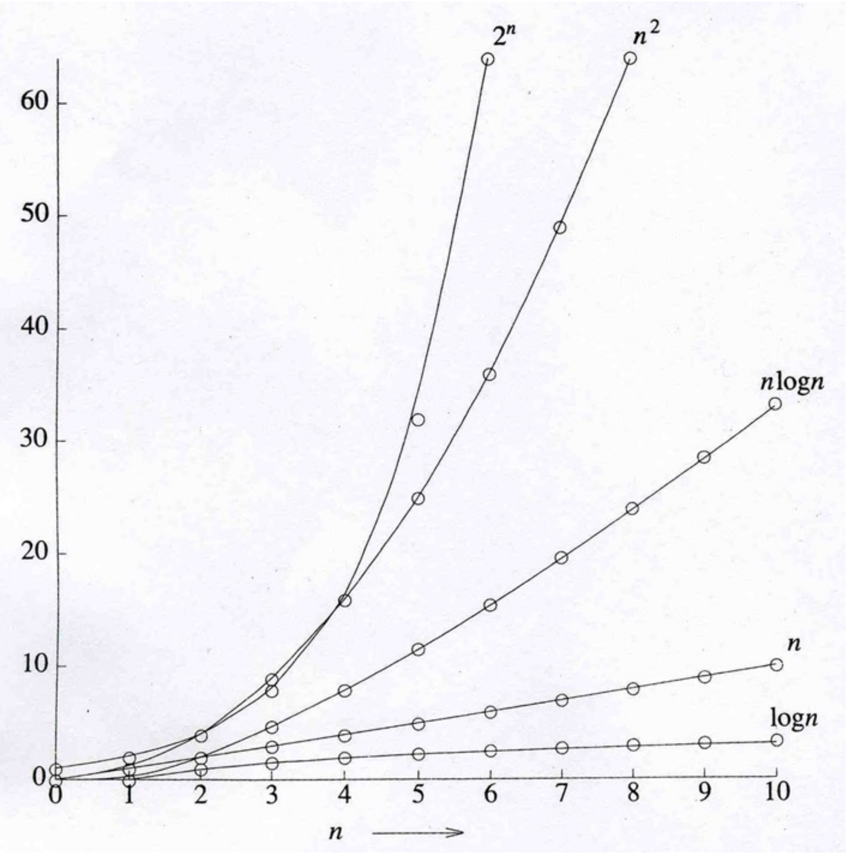


  - Asymptotically equal to *f(n)*

Asymptotic Running Time

- If $T_A(n) \in \Theta(f(n))$, we say that algorithm *A* has asymptotic running time $\Theta(f(n))$

- Typical growth rate:
  - $\Theta(1)$ – constant
  - $\Theta(\log(n))$ – logarithmic
  - $\Theta(n)$ – linear
  - $\Theta(n * \log(n))$ – log linear
  - $\Theta(n^2)$ – quadratic
  - $\Theta(n^3)$ – cubic
  - $\Theta(2^n)$ – exponential
  - $\Theta(n!)$ – factorial



Asymptotic Algorithm Analysis

- The asymptotic analysis of an algorithm determines the running time in big-Oh notation
  - Find the worst-case # of *primitive operations* executed as a function of the input size
  - Express it with big-Oh notation

- Since constant factors and lower-order terms are eventually dropped anyhow, we can disregard them when counting primitive operations

Asymptotic Algorithm Analysis: Example

program segment

```
for i:=1 to n do
    for j:=1 to n do begin
        C[i,j]:=0;
        for k:=1 to n do
            C[i,j]:=C[i,j]+A[i,k]*B[k,j]
    end
```

$$T(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}(c_1 + \sum_{k=1}^{n} c_2) = \sum_{i=1}^{n}\sum_{j=1}^{n}(c_1 + c_2 \cdot n)$$

$$= \sum_{i=1}^{n}(c_1 \cdot n + c_2 \cdot n^2) = c_1 \cdot n^2 + c_2 \cdot n^3$$

$$\Rightarrow T(n) \in O(n^3)$$

Limitations of Analysis

- Not account for *constant factors*, but constant factor may dominate
  - $1000*n$ vs. n^2 (when interested only in $n < 1000$)

- Not account for different *memory access times* at different levels of memory hierarchy
  - Cache Memory << MM << HDD

- Programs that do more computation may take less time than those that do less computation
  - Cost (fetch from *MM*) >> Cost (operation in *CPU*)
  - Memory access could take more than computation

Intuition for Asymptotic Notation

- Big-Oh
  - f(n) is O(g(n)) if f(n) is asymptotically less than or equal to g(n)
- Big-Omega
  - f(n) is Ω(g(n)) if f(n) is asymptotically greater than or equal to g(n)
- Big-Theta
  - f(n) is Θ(g(n)) if f(n) is asymptotically equal to g(n)

- Little-oh
  - f(n) is o(g(n)) if f(n) is asymptotically *strictly* less than g(n)
- Little-omega
  - f(n) is ω(g(n)) if is asymptotically *strictly* greater than g(n)

References

- Further reading list and references
  - https://www.w3schools.com/cpp/
  - https://en.wikipedia.org/wiki/Asymptotic_analysis
  - https://www.geeksforgeeks.org/difference-between-big-oh-big-omega-and-big-theta/

- Slide credit
  - Jaesik Park
  - Seung-Hwan Baek
  - Jong-Hyeok Lee