

Asymptotic notation

Saturday, March 12, 2022 16:41

For the glory of God

Why do we need asymptotic notation?

Before taking a deep dive into what asymptotic notation is, I think it's better to start off talking about the necessity of asymptotic notation.

Let's say that :

- A **shortest path problem** is given from a company.

↳ Given origin (e.g., home) and destination (e.g., Handong Global University),

which path is going to be minimizing travel time?

- One student develops an algorithm, namely X algorithm, to address the aforementioned problem.
- The other student develops another algorithm, namely Z algorithm, to address the same problem.
- They have the same answer (i.e., the shortest path) ; however, they have different running time to solve the computational problem.

e.g., The X algorithm takes 10 minutes while the Z algorithm takes 3 minutes.

↳ However, it is worth mentioning that they use different computers or laptops.

The question is then as follows.

" How would we be able to come up with a way that does not depend on a machine architecture? "

Asymptotic notation allows us to meaningfully compare algorithms by implementing the following idea :

- Ignore machine dependent constants

e.g., $\boxed{3n^3} = 3n^3 = O(n^3)$; where O represents Big-O notation

↳ $T(n)$ denotes the running time of an algorithm on input of size n .

- Look at **growth** of $T(n)$ as $n \rightarrow \infty$

↳ This intuitively means that we can say an algorithm is fast when the running time grows slowly.

(Also, it means that we would like to see the running time especially as the input size gets very large)

↳ We are asked to change a way of thinking in terms of algorithm running time such as :

" How much time does it take to run an algorithm? ⇒ How does the running time of an algorithm grow? "

okay, we have discussed about the necessity of asymptotic notation so far. Let's move on the next topic.

What is asymptotic notation?

- Basically, it is mathematical notation that describes algorithm efficiency.

- Again, asymptotic notation allows us to analyze the running time of an algorithm by identifying its behavior as input size for the algorithm increases.

· There are three different types of asymptotic notation :

a) Big-O notation : The worst case, e.g. $O(n)$

b) Big- Ω notation : The best case, e.g. $\Omega(n^2)$

c) Θ notation : Average case, e.g. $\Theta(\log n)$

· please note that the Big-O notation has been widely used in practice as people tend to talk about the worst case.

· Let's then talk a little bit more about the Big-O notation.

Big-O notation

· Let's say that :

- A shortest path problem is given from a company.

- A student develops Y algorithm to address the problem. \rightarrow which is denoted by $T(n)$

- With asymptotic notation, the student notices the running time of the algorithm as follows :

$$T(n) = pn^2 + qn + r ; \text{ where } p, q, r \text{ are all positive constants}$$

- Here, the Big-O notation comes in, allowing us to claim the following argument :



$$T(n) = O(n^2)$$

- now, we can say that the complexity of the Y algorithm is $O(n^2)$, which can be compared with other algorithms in a consistent manner for solving the shortest path problem.

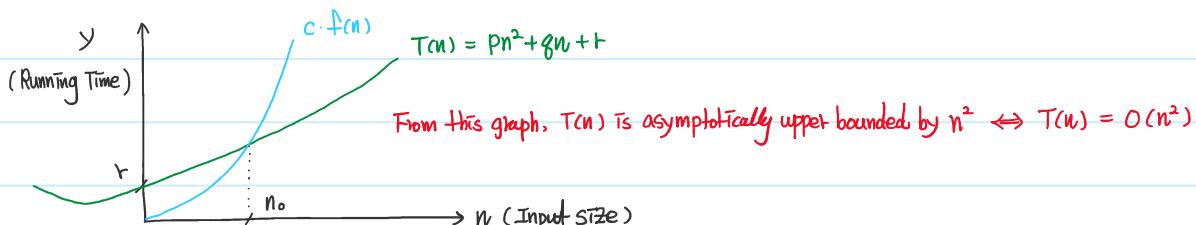
· The question is then as follows :

" Where does $O(n^2)$ come from? What is the rigorous mathematical definition for Big-O notation? "

· The Big-O notation is mathematically defined as follows :

$T(n)$ is $O(f(n))$ if $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$; where c is constant

↳ what it means, graphically.

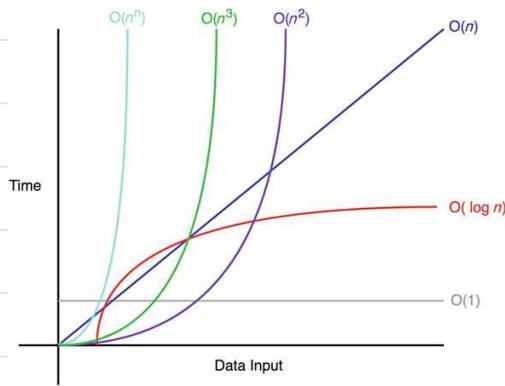


< Notational Sketch >

; Here, note that n represents a relatively big input size

↳ This indicates that asymptotic notation is rarely used for small input size.

- For your reference, take a look at the following Big-O complexity graph.



Why do we need to pay attention to an algorithm efficiency?

- So far, we are mainly talking about speed, efficiency, fast, slow, and so forth of an algorithm.
 - In fact, algorithm performance (i.e., efficiency) was less important than other factors such as:
 - Correctness
 - Robustness
 - Maintainability
 - User friendliness
- ⋮
- However, with the advent of big data recently, we have encountered large-scale problems.
 - Designing an algorithm in a more efficient manner becomes very important as performance is highly associated with money.