Big O notation

Data Structure, second stage, computer science department, college of science for women, university of Baghdad.

Dr. Amer Al-Mahdawi

What is Rate of Growth?

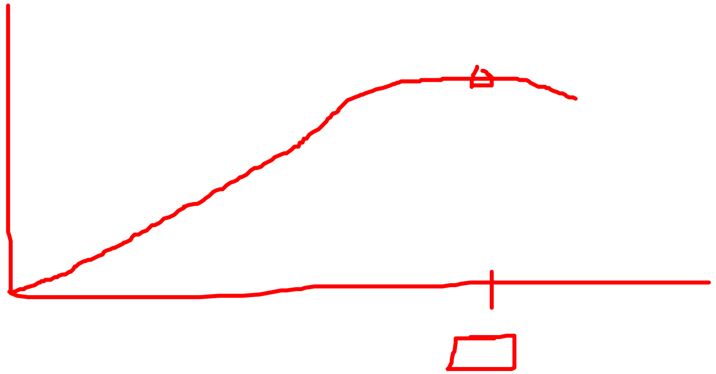
- The rate at which the running time increases as a function of input is called *rate of growth.*
- Let us assume that you go to a shop to buy a car and a bicycle.
- If your friend sees you there and asks what you are buying, then in general you say *buying a car.*
- This is because the cost of the car is high compared to the cost of the bicycle (approximating the cost of the bicycle to the cost of the car).
    - *Total Cost = cost_of_car + cost _of _bicycle*
    - *Total Cost = cost_of _car (approximation)*
- For the above-mentioned example, we can represent the cost of the car and the cost of the bicycle in terms of function, and for a given function ignore the low order terms that are relatively insignificant (for large value of input size, n). As an example, in the case below, n^4 , $2n^2$, 100n and 500 are the individual costs of some function and approximate to n^4 since n^4 is the highest rule of growth.
    - $n^4 + 2n^2 + 100n + 500 \gtrsim n^4$

Commonly Used Rates of Growth

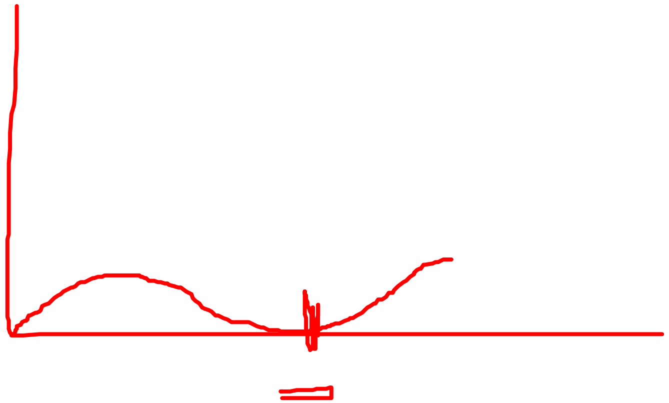
Time Complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
$logn$	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
$nlogn$	Linear Logarithmic	Sorting n items by 'divide-and-conquer' - Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

Types of Analysis

- To analyze the given algorithm, we need to know with which inputs the algorithm takes less time (performing well) and with which inputs the algorithm takes a long time.
- We have already seen that an algorithm can be represented in the form of an expression.
- That means we represent the algorithm with multiple expressions: one for the case where it takes less time and another for the case where it takes more time.
- In general, the first case is called the *best case* and the second case is called the *worst case* for the algorithm.
- To
- To analyze an algorithm, we need syntax, and that forms the base for asymptotic analysis/notation.
- There are three types of analysis:
- **Worst case:**
- Defines the input for which the algorithm takes a long time.
- Input is the one for which the algorithm runs the slowest.



Types of Analysis

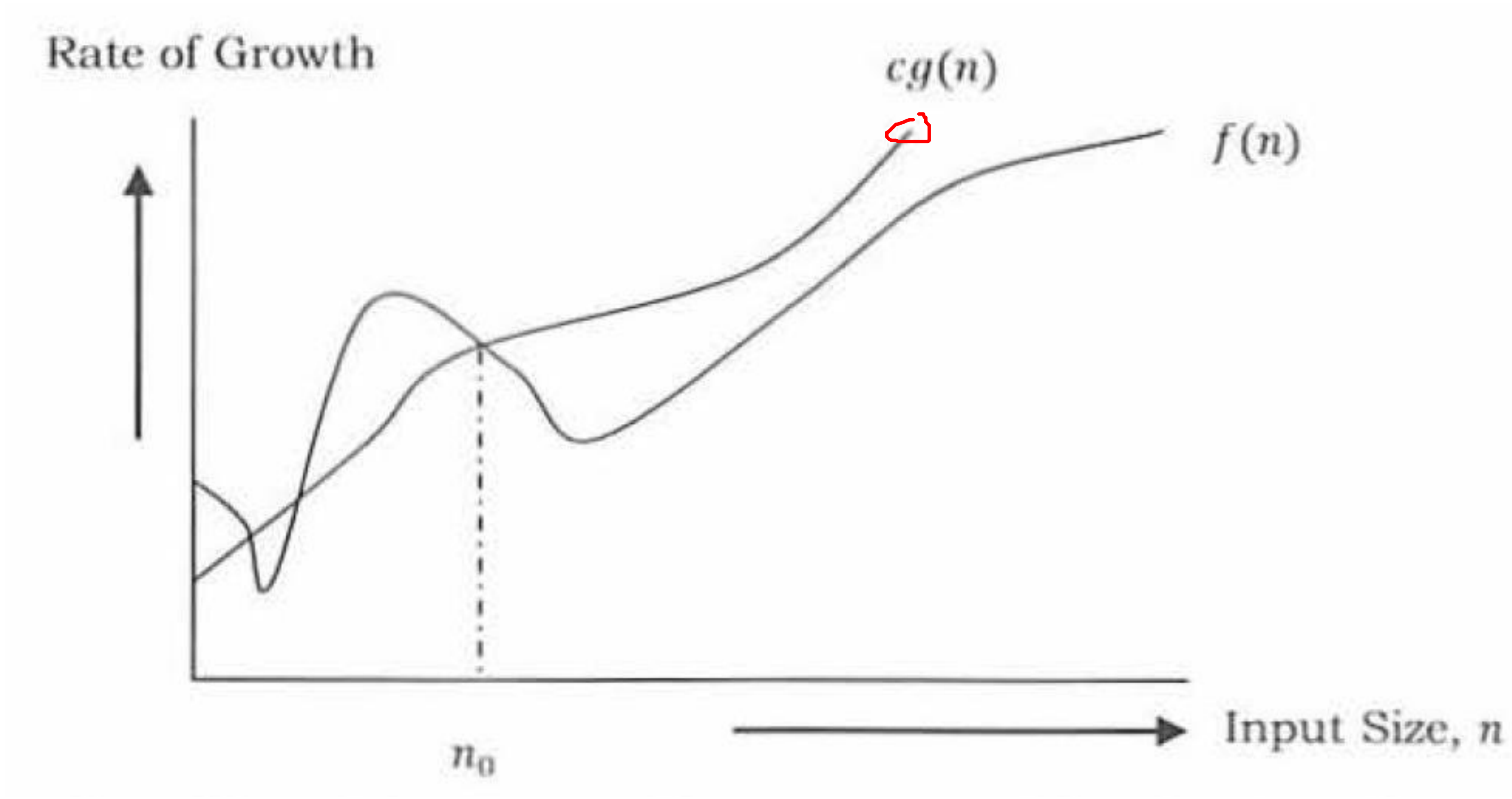


- **Best case:**
- Defines the input for which the algorithm takes the least time.
- Input is the one for which the algorithm runs the fastest.
- **Average case:**
- Provides a predict ion about the running time or the algorithm.
- Assumes that the input is random.
    - *lower Bound* <= *Average Time* <= *Upper Bound*
- For a given algorithm, we can represent the best. worst and average cases in the form of expressions.
- As an example, let $f(n)$ be the function which represents the given algorithm.
    - $f(n) = n^2 + 500$, for worst case
    - $f(n) = n + 100n + 500$, for best case

Types of Analysis

- Similarly for the average case. The expression defines the inputs with which the algorithm takes the average
- running time (or memory).
- Big-0 Notation
- This notation gives the *tight* upper bound of the given function.
- Generally, it is represented as $f(n) = O(g(n))$.
- Thot means, at larger values of n, the upper bound or $f(n)$ is g(n) .
- For example, if $f(n) = n^4 + 100n^2 + 10n + 50$ is the given algorithm, then n^4 is g(n).
- That means g(n) gives the maximum rate of growth for f(n) at larger values of n..

Types of Analysis



Types of Analysis

- Let us sec the O - notation with a little more detail.
- O - notation defined as O(g(n)) = {f(n): there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$is an asymptotic tight upper bound for $f(n)$.
- *g(n)* is an asymptotic tight upper bound for $f(n)$.
- Our objective is to give the smallest rate of growth *g(n)* which is greater than or equal to the given algorithms' rate or growth $f(n)$.
- Generally, we discard lower values of *n*. That means the rate of growth at lower values of *n* is not important.
- In the figure, n_0 is the point from which we need to consider the rate of growth for a given algorithm.
- Below n_0, the rate of growth could be different. n_0 is called threshold for the given function.

Types of Analysis

- **Big-0 Visualization**
- *O(g(n))* is the set of functions with smaller or the same order of growth as *g(n).* For example; $O(n^2)$ includes O(1), O(n), $O(nlogn)$, etc.
- **Note:** Analyze the algorithms at larger values of *n* only. What this means is, below n_0 we do not care about the rate of growth.