Asymptotic Analysis of Algorithms
(Still Under Development)

1. Motivation

- Often there are several ways (algorithms) to solve a problem.
- Algorithms can (greatly) vary in terms of efficiency (how fast they run and how much RAM memory they need).
- WIth today's fast computers, running time or memory requirement may **not be an issue** when we are dealing with small data (**small input data size**)
- However, **when input size grows**, even the fastest computers may need **YEARS** to finish running an **inefficient** algorithm on a **very large data set**! Similarly, an inefficient algorithm may require huge amount of **memory beyond what is available** (have you ever gotten an out-of-memory error?).
- Therefore we need to be able to **MEASURE** and **COMPARE** the **efficiency** of different algorithms for solving the same problem.
- We **only care** about algorithms' efficiency for **very large input sizes (i.e., when n→∞).** Thus we call this **Asymptotic** Analysis.

2. Measuring Time and Space Requirements

Let's start with some important questions and think about their answer:

Question1: **Given a specific input,** how would you measure the **speed** of an algorithm that processes (e.g., sorts) that input? **Note that algorithms are codes and are independent of any machine or programming language**.

1. **Running time (milliseconds)**
2. Number of **lines** of code executed
3. Number of **elementary CPU operations executed**
4. Number of **function calls**

Question2: **Given a specific input**, how do you measure the **memory** requirement for a given algorithm?

1. **Total** #bytes it uses on the **RAM** memory **from start to finish**
2. **Minimum** #bytes it uses **at any given time** during its execution
3. **Maximum** #bytes it uses **at any given time** during its execution
4. **Average** #bytes it uses **at any given time** during its execution

Question3: Can we judge the speed and memory requirements of an algorithm based on **one sample input case**?

1. **Yes**, if the input size is **very large**
2. **Yes**, if we consider the **worst case** scenario, **regardless of input size**
3. **Yes**, if we consider the **worst case** scenario for a **very large** input size
4. **No**, we need to analyze the the algorithm 's performance for inputs with **different sizes** (small, medium, large, very large) and take the **average**

Question4: What is the **best case** when dealing with a **sorting** algorithm? What is the **worst** case?

Question5: What determines the **total number of elementary operations** executed by an algorithm?

1. Input size (n)
2. Input itself (e.g., for sorting, are all elements out of order? Is the input already sorted? Is it ordered reversely?)
3. Number of CPUs
4. Amount of memory used
5. Both 1 and 2

Answers at the end of this page.

2.1. Running time as a function of input size

From now on, when we say "run time" of an algorithm, we mean #elementary operations executed by that algorithm. What determines #elementary operations executed by an algorithm? input size (n) for sure. Therefore **we can represent the running time of an algorithm as f(n) : a function of input size.**

2.2. Best Case, Worst Case and Average Case

In addition to input size, the input itself (the values and the structure of the elements in the input) also determines the running time of an algorithm.
For any algorithm, one can come up with a best case, a worst case and something in between (average case). When doing Big-O analysis, we are concerned with the WORST CASE scenario only.

3. Big-O Notation and growth rate

We saw that the running time of a given algorithm is a **function of the input size, n; That is f(n).** However, we are not interested in the running time for small input sizes. We are interested

in the running time for very large input sizes. More precisely, we want to know the running time when **input size approaches infinity**. That's why we say **asymptotic**.
"In computer science, big O notation is used to **classify algorithms** according to **how their run time or space requirements grow as the input size grows**."

Next we are going to examine the growth rates of several functions.

3.1. Growth rate of Polynomial, Exponential and Logarithmic functions

Please watch the following video:
Comparing Linear, Exponential, and Quadratic Functions

Experiment and compare for yourself:

Compare the growth rate of different functions using a function plotter. Here is a good one which allows you to plot and compare several functions:

https://www.desmos.com/calculator

Using the above tool, **plot the following pairs of functions** and **compare their growth for small, large and very large x values (by zooming in or out).**

- Plot Y=X (**linear**) and Y=X^2 (**quadratic**)
- Plot Y=X^2 and Y=X^3 (**polynomials with different degrees**)
- Plot Y=X^2 (**polynomial**) and Y=2^X (**exponential**)
- Plot Y=X^10 (**polynomial with higher degree**) and Y=2^X (**exponential**)
- Plot Y=log(X) (**logarithmic**) and Y=X (**linear**)

3.2. Growth Rates and Derivatives

From calculus, we know that the **growth rate** of a function y=f(x) **at a certain point x=a** can be obtained by calculating its **derivative at point x=a** which is **y'=d/dx f(a)**.

For example, the growth rate for y=x^2 at point x=3 is 6 because derivative of y=x^2 is : y'=2*x, and at x=3, we have y'=2*3=6.

Here are the derivative formulas for some common functions:

- **derivative of (x^a) =a x^{a-1}**
- **derivative of (a^x) =a^x ln a**
- **derivative of ($\log_a x$) = 1 / (x ln a)**

Based on this, **calculate** is the growth rate (derivative) of the following functions when **x=10, x=1000 and x= infinity:**

- Q6: Linear functions: y=**x**, y= **2x**, y= **1000x** or generally y=**ax**
- Q7: Quadratic and higher-order: y=$\mathbf{x}^2$, y= $\mathbf{x}^3$, or generally y=$\mathbf{x}^a$

- Q8: Exponential: Y= $\mathbf{2}^{x}$ or generally Y= $\mathbf{a}^{x}$
- Q9: Logarithmic: Y =log **x**,

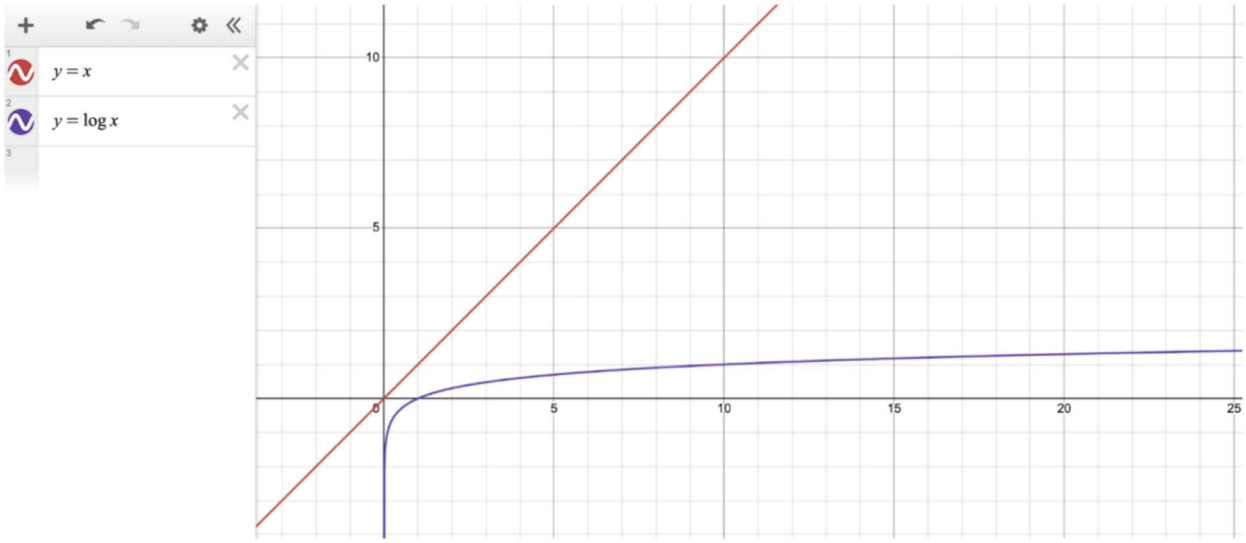
Answers at the end of this page

Do your calculated growth rates suggest big differences between the growth rate of the above function types (linear, quadratic, exponential and logarithmic)?

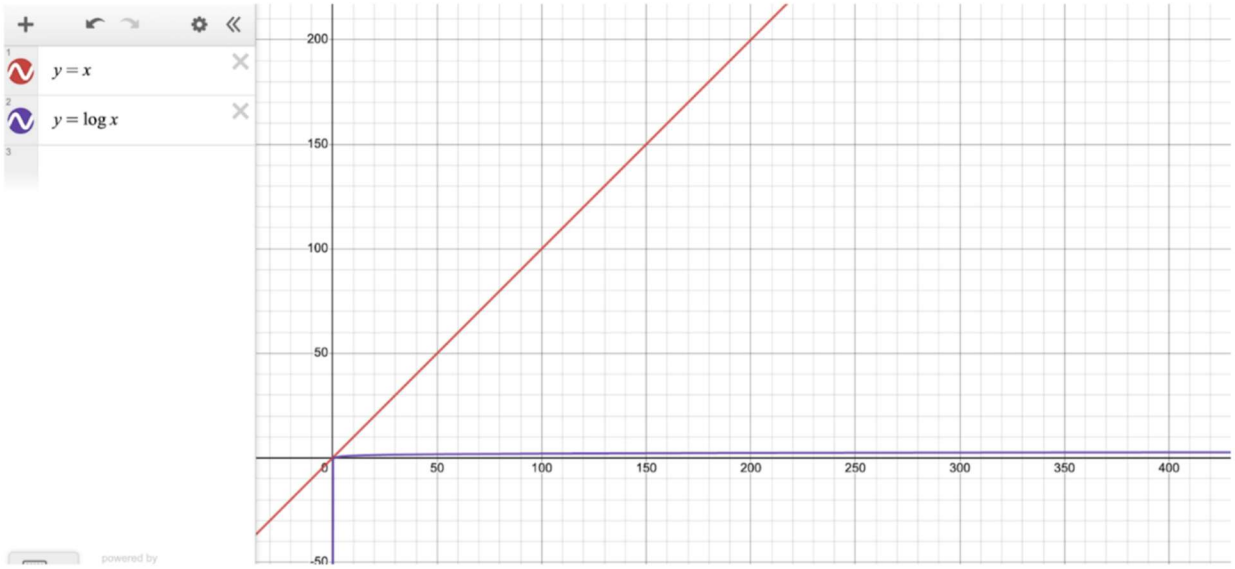
3.3. Why we love logarithmic growth

Logarithm and exponential are the opposite of each other in every way, including their growth rate! While the growth rate for exponential functions is terribly high, the growth rate of logarithmic functions is wonderfully low: as the input size (x) grows, growth **rate** y'(x)) decreases! Why? Because **the derivative of ln x is 1/x** which means it **decreases as x increases**! Same for other base numbers (for $\mathbf{log_a}$ we just multiply the x in the denominator by ln a).

Logarithmic growth is even better than linear growth. In the following graphs, the **red** line is a **linear** function and the **purple** is **log x**.



Let's zoom out (i.e., view bigger x values):



Almost no growth! Isn't that amazing? **Remember this graph to appreciate all the work we will do to come up with logarithmic-time algorithms. Isn't it worth it?**

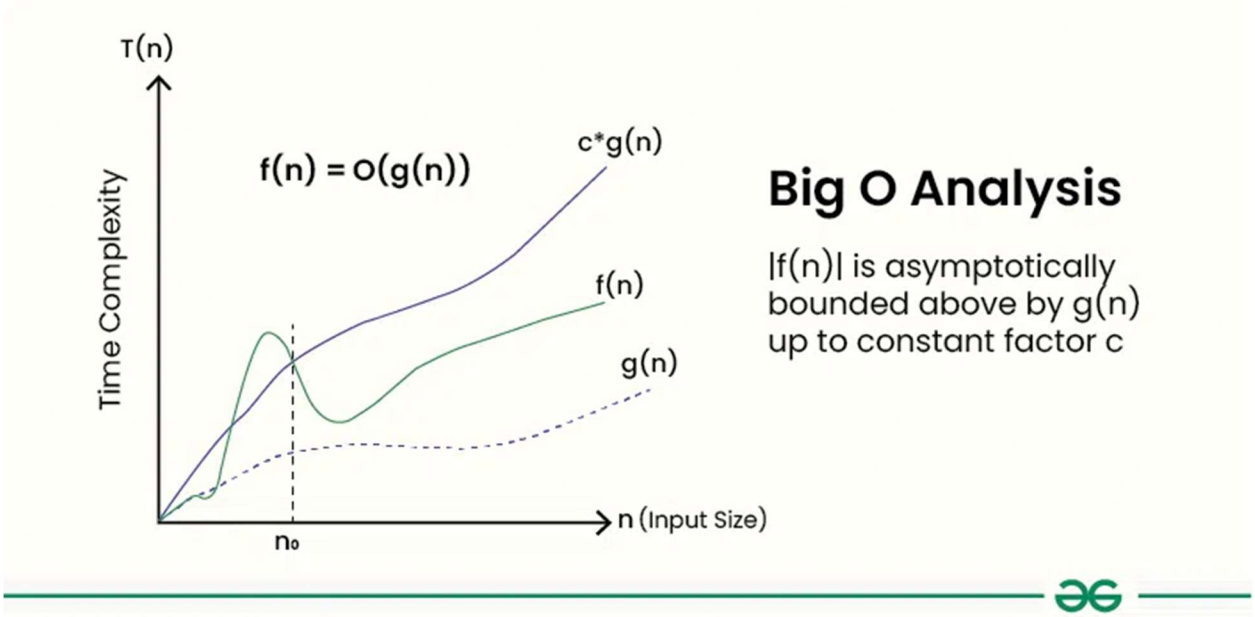
✏ **Experiment and compare for yourself:**

Using the plotting tool (https://www.desmos.com/calculator), **plot the following pairs of functions** and **compare their growth for very large x values (by zooming out):**

- Plot Y=log(X) (**logarithmic**) and Y=2^X (**exponential**)
- Plot Y=log(X) (**logarithmic**) and Y=X^2 (**quadratic**)
- Plot Y=log(X) (**logarithmic**) and Y=X (**linear**)
- Plot Y=log(X) (**logarithmic**) and Y=Sqrt(X)

4. Big-O definition

Now we are ready for the definition of Big-O:
Given two functions f(n) and g(n), we say that f(n) is O(g(n)) if there exist constants c > 0 and n_0 ≥ 0 such that f(n) ≤ c*g(n) for all n ≥ n_0.
In simpler terms, f(n) is O(g(n)) if f(n) grows no faster than c*g(n) for all n≥ n_0 where c and n_0 are constants.



In Big-O analysis, we are not interested in the precise running time function f(n). We are interested in knowing the upper bound of its growth rate in the worst case: is it constant? logarithmic? linear? quadratic? exponential?
"Big O notation characterizes functions according to their growth rates: different functions with the **same asymptotic growth rate** may be represented using the same O notation."

That is, in Bi-O analysis, the following functions are within the same group (linear growth): f(n)= x or f(n)= 2x or f(n)= 100x+1000 . we say that f(n) ∈ O(n)

Similarly, all the following functions are considered quadratic growth: f(n)= x^2 or f(n)= $10x^2$ or f(n)= $10x^2+1000x + 10^{100}$. we say that f(n) ∈ O(x^2).

Big-O: Why we can omit the constants and lower order polynomials

In typical usage the *O* notation is **asymptotical, that is, it refers to very large x.** In this setting, **the contribution of the terms that grow "most quickly" will eventually make the other ones irrelevant**. As a result, the following **simplification rules** can be applied:

- **If *f*(*x*) is a sum of several terms, if there is one with largest growth rate, it can be kept, and all others omitted.**
- **If *f*(*x*) is a product of several factors, any constants (factors in the product that do not depend on x) can be omitted.**

For example, let $f(x) = 6x^4 - 2x^3 + 5$, and suppose we wish to simplify this function, using *O* notation, to describe its growth rate as x approaches infinity. This function is the sum of three terms: $\mathbf{6x^4}$, $\mathbf{-2x^3}$, and **5.** Of these three terms, the **one with the highest growth rate** is the one **with the largest exponent as a function of x**, namely $\mathbf{6x^4}$. Now one may apply the second rule: $6x^4$ is a **product** of **6** and $\boldsymbol{x^4}$ in which **the first factor does not depend on *x***. Omitting this factor results in the **simplified form** $\boldsymbol{x^4}$. Thus, we **say that *f*(*x*) is a "big O" of** $\boldsymbol{x^4}$. Mathematically, we can write $\boldsymbol{f(x) = O(x^4)}$.

✎ **Experiment and compare for yourself:**

Using the plotting tool (https://www.desmos.com/calculator), **plot the following pairs of functions** and **compare their growth for very large x values (by zooming out):**

- Role of **coefficient**: Plot Y=X and Y=**100**X
- Role of **constant**: Plot Y=X and Y=X**+1000**
- Role of coefficient and constant: Plot Y=**10**X+**100** and Y=X
- **Can a high coefficient and constant make linear function grow faster than quadratic?** Plot **Y=1000X+1000 and Y=X^2**

F(n) is O(x^2)

5. Big-Theta

Informally, we can say f(x) is Big-Theta of g(x) when f(x) can be sandwiched between Y= C_1.g(x) and Y=C_2.g(x)

Experiment and compare for yourself:

Using the plotting tool (https://www.desmos.com/calculator), check the following **for very large x values (by zooming out):**

- **Can you sandwich** Y=2X+100 between two functions Y=C_1X and Y=C_2X? If so, what are sample C_1 and C_2 that can sandwich Y=2X+100?
- Come up with a quadratic function and find two other quadratic functions in the form Y=C_1X^2 and Y=C_2X^2 that can **sandwich** it.
- Plot Y=X^3, Y=2X^3+ 10X^2+ 1000, Y=3X^3 and try to find out **whether the second function can be sandwiched** between the first and third? **If so, for which X values** (approximately) does it happen?
- Can you **sandwich** Y=2^X+X^2 between two functions Y=C1 * 2^X and Y=C2 * 2^X?
- Come up **with** an exponential function and find two other exponential functions in the form Y=C12^X and Y=C22^X that can **sandwich** it.

TODO: Calculation running time of algorithms as a function of input size

References

- https://blog.devgenius.io/understanding-big-o-asymptotic-analysis-4708ac0dc2f2
- https://hackr.io/blog/big-o-notation-cheat-sheet
- https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/
- https://en.wikipedia.org/wiki/Big_O_notation
- https://hendrix-cs.github.io/csci382/pub/hw-00-background.pdf
- https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

- Video explaining the **relationship between exponential and logarithmic**: https://www.youtube.com/watch?v=zzu2POfYv0Y

Answer key: Q1:3 **, Q2:** 3 **, Q3:** 3

Q4: best case: input is already sorted. Worst case: Sorted reversely.

Q5: 5

Q6: growth rate is **constant:**

- y=**x -> y'=1 for any x**
- y= **2x -> y'=2 for any x**
- y= **1000x -> y'=1000 for any x**
- y=**ax -> y'=a for any x**

Q7 Quadratic and higher-order:

- y=$\mathbf{x^2}$ **-> y'=2x** -> growth rate at x=10, 100 and ∞, is **20, 200** and ∞ respectively
- y= $\mathbf{x^3}$ **-> y'=3x**^{2} -> growth rate at x=10, 100 and ∞, is **300, 30000** and ∞ respectively
- y=$\mathbf{x^a}$ **-> y'=ax**$^{a-1}$ -> growth rate at x=10, 100 and ∞, is **a*10**$^{a-1}$**, a*100**$^{a-1}$ and ∞ respectively

Q8: Exponential:

- Y= $\mathbf{2^x}$ => y'= $\mathbf{2^x}$ **ln 2** => growth rate at x=10, 100 and ∞, is $\mathbf{2^{10}}$**ln 2,** $\mathbf{2^{100}}$**ln 2** and ∞ respectively
- Y= $\mathbf{a^x}$ => y'= $\mathbf{a^x}$ **ln a** => growth rate at x=10, 100 and ∞, is $\mathbf{a^{10}}$**ln a,** $\mathbf{a^{100}}$**ln a** and ∞ respectively

Q9: Logarithmic:

- Y =log **x** => y'= **1/ (x ln 10)** => growth rate at x=10, 100 and ∞, is **1/ (10 ln 10), 1/ (100 ln 10)** and **1/ (∞ ln 10) = 0** respectively.