# Chapter 3 Section 2

Mohamed Elmahy
Nowshin Sayara
Fariha Rashid

October 15, 2020

## 1 Introduction

Algorithms can be beneficial when trying to perform a task or solve a problem. Programmers need to choose the best-suited algorithm for any given program to optimize the performance better. There are many vital components to what makes an algorithm a good one. Notably, we look at the speed and number of steps involved. We can look at an algorithm's efficiency through asymptotic notations: Big O, Big Omega, and Big Theta.

Big O describes an algorithm's worst-case by choosing a polynomial function's highest order, ignoring the constants. In contrast, Big Omega describes the best-case run time of an algorithm by selecting the polynomial function's lowest order, ignoring the constants. Finally, Big Theta describes both the best case and the worst-case run-time of an algorithm, which means the algorithm has to be both Big O and Big Omega simultaneously.

## 2 Asymptotic Analysis

### 2.1 Importance

Asymptotic analysis defines the run-time performance of an algorithm. Using asymptotic analysis, we can mathematically conclude the best case, average case, and worst case scenario of an algorithm.

The time required by an algorithm falls under three types:

- Best Case - Minimum time required for program execution.
- Average Case - Average time required for program execution.
- Worst Case - Maximum time required for program execution.

# 3 Big-O Notation

## 3.1 Definition:

We will need to let $f$ and $g$ be functions from $Z \to R$. We then say that $f(x)$ is $O(g(x))$. This is read as "$f(x)$ is big-oh of $g(x)$." If there are constants $C$ and $k$ such that $|f(x)| \leq C|g(x)|$, whenever $x > k$. To find a pair of witnesses, select a value of $k$ for which the size of $|f(x)|$ can be readily estimated when $x > k$ and then see whether we can use this estimate to find a value of $C$ for which $|f(x)| \leq C|g(x)|$, whenever $x > k$.

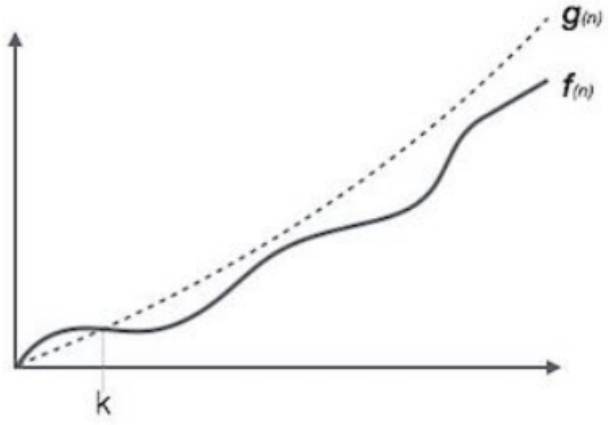


Figure 1: Big-O

## 3.2 Importance of Big-O

One of the advantages of using big-O notation is that we can estimate a function's growth without worrying about constant multipliers or smaller order terms. This means that, using big-O notation, we do not have to worry about the hardware and software used to implement an algorithm. The big-O notation is used extensively to estimate the number of operations an algorithm uses as its input grows. With the help of this notation, we can determine whether it is practical to use a particular algorithm to solve a problem as the input size increases.

### 3.3 Example

**Question:** Show that $7x^3$ is O($x^2$)

*Proof.* Assume that $7x^3$ is O($x^2$), so by definition $\exists\ c\ \geq\ 1$, and $k \geq 0$, such that

$$\frac{7x^3}{x^2} \leq \frac{cx^2}{x^2}, \forall x > k$$

This implies

$$7x \leq c,\ \forall x \geq k$$

This is clearly false since $x$ is a variable and $C$ is a fixed constant. This means that $x$ will continue to grow to infinity. But clearly,

$$7x^2\ is\ O(x^3)$$

$$x^2 \leq x^3, \forall x \geq 1$$

Without a doubt, we know this is true.

$$\Rightarrow 7x^2 \leq 7x^3, \forall x \geq 1$$

We multiply by 7 to get the original expression. By the previous equation, this is also true. So, $C = 7$ and $k = 0$ where $x > k$. We choose k=0 because x is strictly greater than k. Therefore,

$$7x^2\ \in\ O(x^3)$$

□

### 3.4 Example

**Question:** Show that $n^2$ is not O($n$)

*Proof.* Assume that $n^2 \in\ O(n)$. Then by definition, $\exists\ C \geq 1$, $k \geq 0$ *such that* $n^2\ \leq Cn$. Which implies that

$$n\ \leq\ C,\ \forall n > k$$

This is clearly false since $n$ grows infinitely. To show this, take the limit as $n \Rightarrow \infty$ on both sides of the inequality.

$$\lim_{n\to\infty} n\ \leq \lim_{n\to\infty} C$$

Thus, $\infty \leq C$, This is a contradiction, so $\exists\ x$, $k$ such that $n^2 \leq\ Cn$, $\forall n > k$. Therefore, $n^2 \notin\ O(n)$

## 4 Big-O estimates for some important functions

Polynomials can often be used to estimate the growth of functions. To do that we use Theorem 1 . It shows that the leading term of a polynomial dominates its growth by asserting that a polynomial of degree n or less is ($O(x^n)$).

## 4.1 Theorem 1

$f(x) = a_n x^n + \cdots + a_1 x + a_0$ where, $a_i$ are real numbers, then $f(x)$ is $O(x^n)$. This means that largest exponent in the polynomial equation defines the Big O.

*Proof.* Using triangle inequality, if $x > 1$, we have,

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1} x^{n-1} + a_1 x + a_0| \\ &\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x + |a_0| \\ &= x^n (\ |a_n| x^n + |a_{n-1}| x + \cdots + |a_1| x^{n-1} + \frac{|a_0|}{x^n}) \\ &\leq\ x^n (|a^n| + |a_{n-1}| + \cdots + |a_1| + |a_0|) \end{aligned}$$

This shows that, $|f(x)| \ \leq \ Cx^n$ where $C = (|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|)$ whenever $x \ > \ 1$. Hence, the witnesses $C = (|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|)$ and $k = 1$ show that $f(x)$ *is* $O(x^n)$.

□

## 4.2 Example

**Question:** Give big-O estimates for the factorial function and the logarithm of the factorial function, where the factorial function $f(n) = n!$ is defined by $n! = 1 \cdot 2 \cdot 3 \cdots n$ whenever $n$ is a positive integer, and $0! = 1$.

*Proof.* A big-O estimate for $n!$ can be obtained by noting that each term in the product does not exceed $n$. Hence,

$$n! = 1 \cdot 2 \cdot 3 \cdots n \ \leq \ n \cdot n \cdot n \cdots n \ = \ n^n$$

Each term can be upper bounded by $n$ since they are less than or equal to $n$. The total number of inputs is n. Hence, we get $n^n$. This inequality shows that $n!$ is $O(n^n)$, taking $C = 1$ and $k = 1$ as witnesses. Taking logarithms of both sides of the inequality established $n!$, we obtain

$$\log \ n! \ \leq \ \log \ n^n \ = \ n \log n$$

This implies that $\log \ n!$ *is* $O(n \log n)$. We choose $C = 1$ and $k = 1$ as witnesses. □

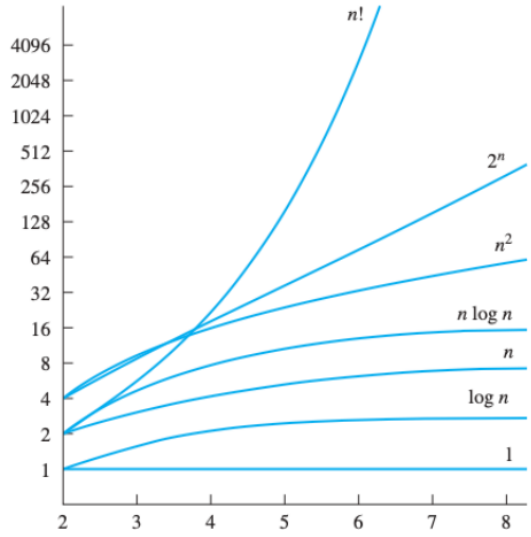


Figure 2: A general hierarchy of functions, ordered by their growth complexity commonly used in big-O estimates: $1 \leq \log(n) \leq n^c \leq n^c \log(n) \leq n^{c+\varepsilon} \leq d^n \leq n! \leq n^n$ where $c, \varepsilon \in (0,\infty)$ and $d \in (1,\infty)$.

# 5 Useful things to know

1) If $d > c > 1$, then $n^c \in O\left(n^d\right)$ but $n^d \in O\left(n^c\right)$. This states that polynomials that have a higher power can be a good upper bound for a general polynomial.

2) Whenever $b > 1$ and $c, d > 0$, then $(\log_b n)^c = \log\binom{c}{b}\ n \in O\left(n^d\right)$ but $n^d \notin O\left((\log_b n)^c\right)$. This states that poly-logarithmic functions are less than polynomial functions.

3) When $c > b > 1$, we have $b^n \notin O\ \left(c^n\right)$ but $c^n \notin O\left(b^n\right)$. This states that the exponential function that has the bigger base is greater.

4) If $c > 1$, we have that, $n^c$ is $O\left(n!\ \right)$ but $n!$ is not $O\left(c^n\right)$. This states that factorial functions grow faster than exponential functions.

# 6 The growth of combinations of Functions

A lot of algorithms use two or more separate sub-procedures. The number of steps used by a computer to solve a problem with the input of a specified size using such an algorithm is the sum of the number of steps used by these sub-procedures. To give a big-O estimate for the number of steps needed, it is necessary to find big-O estimates for the number of steps used by each sub-procedure and then combine these estimates.

## 6.1 Theorem 2

Suppose that $f_1(x)$ is $O(g_1(x))$ and that $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(g(x))$, where $g(x) = (max(|g_1(x)|,\ |g_2(x)|)$ for all $x$.

### 6.2 Theorem 3

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.

### 6.3 Example

**Question:** Give a big-O estimate for $f(n) = 3n\log(n!) + (n^2+3)\log n$, where $n$ is a positive integer.

*Proof.*

$$\begin{aligned} f(n) &= 3n\log(n!) + (n^2+3)\log n \\ f_1(n) &= 3n\log(n!) \\ f_2(n) &= (n^2+3)\log n \end{aligned}$$

First, the product $3n\log(n!)$ will be estimated. From example 4.2, we know that $\log(n!)$ is $O(n\log n)$. Also, recall that $3n$ is $O(n)$ where $k = 1$ and $C = 3$. By using Theorem 3, we get

$$f_1(n) = (n^2+3)\log n \;\leq\; O((n)(n\log n)) \;=\; O(n^2\log n)$$

Next, we find the Big-O of $(n^2+3)$ by upper bounding both terms.

$$n^2 + 3 \leq n^2 + 3n^2 = 4n^2$$

Therefore, $n^2+3$ is $O(n^2)$, where $k = 1$ and $C = 4$. By Theorem 3 it follows that

$$f_2(n) = (n^2+3)\log n \;\leq\; O((n^2)(\log n)) \;=\; O(n^2\log n)$$

Use Theorem 2 to combine the two big-O estimates of $f_1(n)$ and $f_2(n)$ shows that

$$f(n) \leq O(\max(f_1,\, f_2)) = O(\max(n^2\log n,\, n^2\log n)) = O(n^2\log n)$$

Therefore, $f(n)$ is $O(n^2\log n)$. □

## 7 Big-Omega Notation, Ω

Although big-O notation is used to describe the growth of a function, it does have some limitations to it. We have the upper bound when f(x) is O(g(x)) in terms of g(x), for the size of f(x) for large values of x, but we don't have the lower bound. This is why we use big-Omega notation. It gives us the lower bound, which is why it is used to describe the best case running time of an algorithm.

### 7.1 Definition

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ *is* Ω $(g(x))$, which is read as "$f(x)$ is big-Omega of $g(x)$". If there are constants C and k with $C > 0$ such that

$$|f(x)| \;\geq\; C\;|g(x)|,\; \textit{whenever}\;\; x > k$$

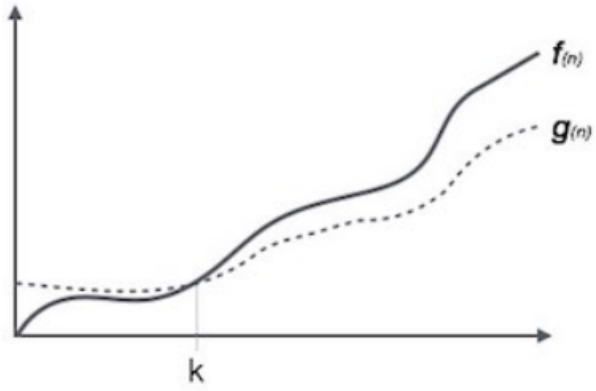


Figure 3: Omega Notation, $\Omega$

### 7.2 Example

**Question:** Show that $f(x) = 8x^3 + 5x^2 + 7 \in \Omega(x^3)$

*Proof.* We need to find a C value for which f(x) is greater.

$$8x^3 + 5x^2 + 7 \geq Cx^3$$

Obviously

$$8x^3 + 5x^2 + 7 > 8x^3, \forall x \geq 0$$

Choose $C = 8$ and $k = 0$. Therefore, $f(x) \in \Omega(g(x))$. □

## 8 Theta Notation, $\theta$

When we want to give both an upper and a lower bound on the size of a function $f(x)$, relative to a reference function $g(x)$, we use big-theta ($big - \theta$) notation. Big-theta is a stronger statement than big-O and big-omega because it is the most accurate and is true for all inputs.

### 8.1 Definition

Let $f$ and $g$ be functions. We say that

$$f(x) \text{ is } \theta(g(x)) \text{ if } f(x) \in O(g(x)) \text{ and } f(x) \in \Omega(g(x)).$$

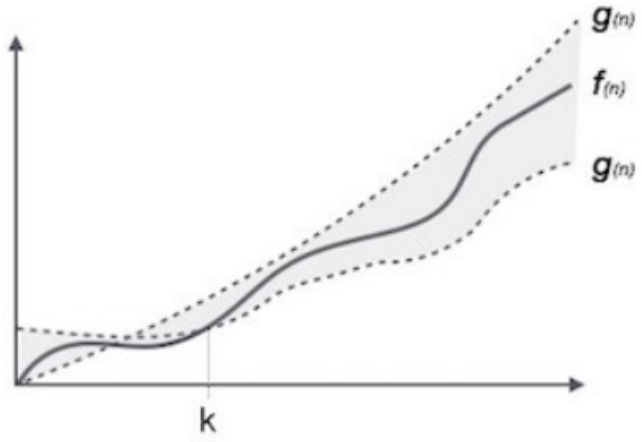


Figure 4: Theta notation, $\theta$

## 8.2 Example

**Question:** Show that $\sum_{i=1}^{n} i \ \in \theta\left(n^2\right)$

*Proof.* First, we show that $\sum_{i=1}^{n} i \ \in \ \Omega(n^2)$

$$\begin{aligned} 1+2+\cdots+n &\geq \frac{n}{2}+(\frac{n}{2}+1)+(\frac{n}{2}+2)+\cdots+n \\ &\geq \frac{n}{2}+\frac{n}{2}+\cdots\cdot+\frac{n}{2} \\ &\geq \frac{n}{2}(\frac{n}{2})=\frac{1}{4}n^2 \ \in \ \Omega\ (n^2) \end{aligned}$$

First, we upper bound each term by $(\frac{n}{2})$ because it is halfway between 1 and n. Next, since we have $(\frac{n}{2})$ terms in total, we multiply $(\frac{n}{2})$ with $(\frac{n}{2})$.

Next, we show that $\sum_{i=1}^{n} i \ \in \ O\left(n^2\right)$

$$\begin{aligned} \sum_{i=1}^{n} i \ &= \ 1+2+\cdots+n \\ &\leq n+n+\cdots+n \\ &= n*n=n^2 \in O(n^2) \end{aligned}$$

We upper bound each term by n. Since the total inputs is n, we multiply n by n. We know that

$\sum_{i=1}^{n} i \ \in O\ \left(n^2\right)$ *and* $\sum_{i=1}^{n} i \ \in \Omega\ \left(n^2\right)$. Therefore, $\sum_{i=1}^{n} i \ \in \theta\ \left(n^2\right)$ □

# 9 Citation

1) Rosen, K. H. (2019). Discrete mathematics and its applications. New York, NY: McGraw-Hill.

2) Roberts. (n.d.). Discrete Math: Big-O Examples and Theorems. https://www.youtube.com/watch?v=ahTJ-TWCrrglist=WLindex=3t=573sab$_{c}hannel = Dr.RobertsatGGC$

3) *Data Structures – Asymptotic Analysis*. (*n.d.*).*Retrieved October* 16, 2020, $from$ https://www.tutorialspoint.com/data$_{s}tructures_{a}lgorithms/asymptotic_{a}nalysis.htm$