Lecture 24. The Growth of Functions: Big-$\mathcal{O}$, Big-Ω and Big-Θ

- Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n. If an algorithm has to scale, it should compute the result within a finite and practical time bound even for large values of n. For this reason, complexity is calculated asymptotically as n approaches infinity.

- Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n. If an algorithm has to scale, it should compute the result within a finite and practical time bound even for large values of n. For this reason, complexity is calculated asymptotically as n approaches infinity.
- In other words, when studying the **complexity of an algorithm**, we are concerned with the growth in the number of operations required by the algorithm as the size of the problem increases.

- Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n. If an algorithm has to scale, it should compute the result within a finite and practical time bound even for large values of n. For this reason, complexity is calculated asymptotically as n approaches infinity.
- In other words, when studying the **complexity of an algorithm**, we are concerned with the growth in the number of operations required by the algorithm as the size of the problem increases.
  - In order to get a handle on its complexity, we first look for a function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to which the algorithm is applied.
  - We then try to compare values of this function, for large n, to the values of some known function, such as a **power function**, **exponential function**, or **logarithm function**.
  - Thus, the growth of functions refers to the relative size of the values of two functions for large values of the independent variable.

- Algorithmic complexity is a measure of how long an algorithm would take to complete given an input of size n. If an algorithm has to scale, it should compute the result within a finite and practical time bound even for large values of n. For this reason, complexity is calculated asymptotically as n approaches infinity.
- In other words, when studying the **complexity of an algorithm**, we are concerned with the growth in the number of operations required by the algorithm as the size of the problem increases.
  - In order to get a handle on its complexity, we first look for a function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n, to which the algorithm is applied.
  - We then try to compare values of this function, for large n, to the values of some known function, such as a **power function**, **exponential function**, or **logarithm function**.
  - Thus, the growth of functions refers to the relative size of the values of two functions for large values of the independent variable.
- Analysis of an algorithm's complexity is helpful when comparing algorithms or seeking improvements.

Definitions

Big-$\mathcal{O}, \Omega$ and Θ are notations representing algorithmic complexity. These measure the time and memory required for an algorithm to run.

Definitions

Big-$\mathcal{O}, \Omega$ and Θ are notations representing algorithmic complexity. These measure the time and memory required for an algorithm to run.

Big-$\mathcal{O}, \Omega$ and Θ are typically used to analyze the worst case complexity of an algorithm. If, for example, n is the size of the input data, then Big-$\mathcal{O}, \Omega$ and Θ are only interested when the input data size n grows arbitrarily large, but they are not so interested when the input is small.

Definition 24.1. $f \in \mathcal{O}(h)$ (f is Big-O of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \leq c|h(x)| \quad \text{for all } x \geq k.$$

Definition 24.1. $f \in \mathcal{O}(h)$ (f is Big-O of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \leq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.2. Calculate a big-$\mathcal{O}$ of $f(x) = 2x^2 + 4x$.

Definition 24.1. $f \in \mathcal{O}(h)$ (f is Big-O of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \leq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.2. Calculate a big-$\mathcal{O}$ of $f(x) = 2x^2 + 4x$.

$$\begin{aligned} 2x^2 + 4x &\leq 2x^2 + x^2 \quad \text{for } x \geq 4 \\ &= 3x^2 \end{aligned}$$

Therefore, $f(x) \in \mathcal{O}(x^2)$.

Definition 24.3. $f \in$ Big-$\Omega(h)$ (f is Big-Omega of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \geq c|h(x)| \quad \text{for all } x \geq k.$$

Definition 24.3. $f \in$ Big-$\Omega(h)$ (f is Big-Omega of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \geq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.4. Calculate a Big-Ω of $f(x) = 2x^2 + 4x$.

Definition 24.3. $f \in$ Big-$\Omega(h)$ (f is Big-Omega of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \geq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.4. Calculate a Big-Ω of $f(x) = 2x^2 + 4x$.

$$2x^2 + 4x \quad \geq \quad 2x^2 \quad \text{for all } x \geq 1$$

Therefore, $f(x) \in \Omega(x^2)$.

Big-Θ Notation

Definition 24.5. Let $h : \mathbb{R} \to \mathbb{R}$ be a function. Then $\Theta(h)$ is the set of all functions f such that

$$c_1|h(x)| \leq |f(x)| \leq c_2|h(x)|$$

for some positive constants c_1, c_2 and for all $x \geq k$ for some $k > 0$. In other words, $\Theta(h) = \mathcal{O}(h) \cap \Omega(h)$. If $f \in \Theta(h)$, we also say that "f is big-theta of h", "f is of order h", or "f grows as fast as h."

Big-Θ Notation

Definition 24.5. Let $h : \mathbb{R} \to \mathbb{R}$ be a function. Then $\Theta(h)$ is the set of all functions f such that

$$c_1|h(x)| \leq |f(x)| \leq c_2|h(x)|$$

for some positive constants c_1, c_2 and for all $x \geq k$ for some $k > 0$. In other words, $\Theta(h) = \mathcal{O}(h) \cap \Omega(h)$. If $f \in \Theta(h)$, we also say that "f is big-theta of h", "f is of order h", or "f grows as fast as h."

Example 22.4. From Example 22.2 and 22.4, $f(x) = 2x^2 + 4x \in \Theta(x^2)$

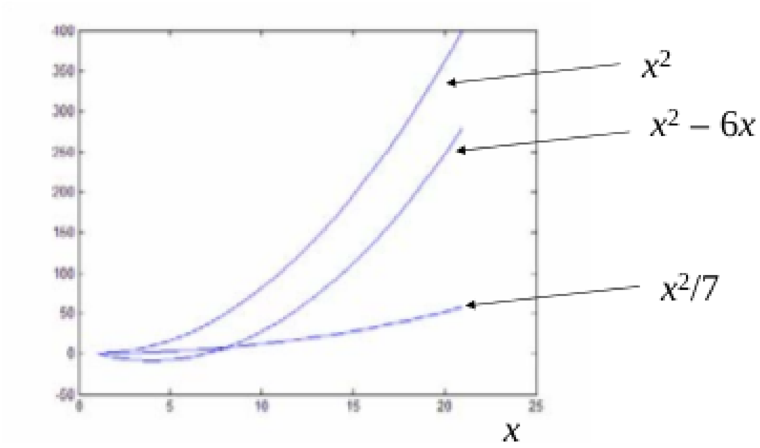
Example 24.6. Prove that $x^2 - 6x$ is $\Theta(x^2)$.

Example 24.6. Prove that $x^2 - 6x$ is $\Theta(x^2)$.

- We can prove easily that $x^2 - 6x \in \mathcal{O}(x^2)$ We can show that $x^2 - 6x$ is $\Omega(x^2)$ as following:
- Note that $x^2 - 6x = x^2(1 - 6/x)$.
- Since $(1 - 6/x) > 0$ for $x \geq 7$,

$$|x^2 - 6x| = |x^2|(1 - 6/x) \geq |x^2|(1 - 6/7) \quad \text{for } x \geq 7$$

- So, choosing $k = 7$ and $c = 1/7$, we have that $x^2 - 6x \in \Omega(x^2)$.



x^2
$x^2 - 6x$
$x^2/7$
x
400
350
300
250
200
150
100
50
0
-50
0
5
10
15
20
25

$f \in \mathcal{O}(h)$ (f is Big-O of h) if $\exists c > 0$ and $\exists k > 0$ such that

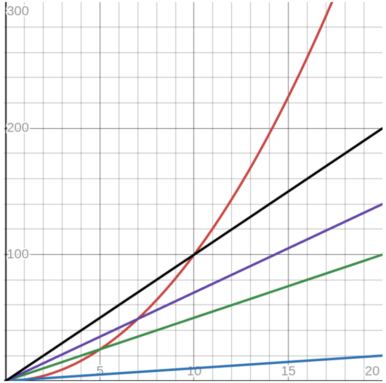
$$|f(x)| \leq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.7. Show that n^2 is not $\mathcal{O}(n)$.

$f \in \mathcal{O}(h)$ (f is Big-O of h) if $\exists c > 0$ and $\exists k > 0$ such that

$$|f(x)| \leq c|h(x)| \quad \text{for all } x \geq k.$$

Example 24.7. Show that n^2 is not $\mathcal{O}(n)$.
Graphically, x^2 (red), x (blue), $5x$ (green), $7x$ (purple), $10x$ (black):



Mathematically,

- To show that n^2 is not $\mathcal{O}(n)$, we must show that no pair of c and k exist such that $n^2 \leq cn$ whenever $n > k$.
- We use a proof by contradiction to show this.
  - Suppose that there are constants c (fixed) and k (fixed) for which $n^2 \leq cn$ whenever $n > k$.
  - Observe that when $n > 0$ we can divide both sides of the inequality $n^2 \leq cn$ by n to obtain the equivalent inequality $n \leq c$.
  - However, no matter what c and k are, the inequality $n \leq c$ cannot hold for all n with $n > k$.
  - In particular, once we set a value of k, we see that when n is larger than the maximum of k and c (for example, $n = \max\{k+1, c+1\}$), it is not true that $n \leq c$ even though $n > k$.
  - This contradiction shows that n^2 is not $\mathcal{O}(n)$.

Big-$\mathcal{O}$ Estimates for Polynomials

- **Theorem 24.7**. Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, where $\forall i (a_i \in \mathbb{R})$. Then $f(x) \in \mathcal{O}(x^n)$.

Big-$\mathcal{O}$ Estimates for Polynomials

- **Theorem 24.7**. Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, where $\forall i(a_i \in \mathbb{R})$. Then $f(x) \in \mathcal{O}(x^n)$.
- **Proof**. For this proof, we will use the "triangle inequality",

$$\forall x \forall y \quad (|x+y| \leq |x| + |y|).$$

  - Using this property

$$\begin{aligned} |f(x)| &\leq |a_n x^n| + |a_{n-1}x^{n-1}| + \cdots + |a_1 x| + |a_0| \\ &= |x^n|(|a_n| + |a_{n-1}|/|x| + \cdots + |a_1|/|x^{n-1}| + |a_0|/|x^n|) \\ &\leq |x^n|(|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|) \text{ for } \; x > 1 \end{aligned}$$

  - So using $c = |a_n| + \cdots + |a_0|$ and $k = 1$ yields the conclusion.

Properties of Big-$\mathcal{O}$: Sums

Theorem 24.8. Let $f(x) \in \mathcal{O}(h_1(x))$ and $g(x) \in \mathcal{O}(h_2(x))$. Then

$$f(x) + g(x) \in \mathcal{O}(\max(h_1(x), h_2(x)))$$

.

Proof

- There exist k_1, k_2, c_1, c_2, such that

$$\begin{aligned} |f(x)| &\leq c_1|h_1(x)| \text{ for all } x > k_1, \quad \text{and} \\ |g(x)| &\leq c_2|h_2(x)| \text{ for all } x > k_2. \end{aligned}$$

- By the triangle inequality,

$$\begin{aligned} |f(x) + g(x)| &\leq |f(x)| + |g(x)| \\ &\leq c_1|h_1(x)| + c_2|h_2(x)| \text{ for all } x > \max(k_1, k_2) \\ &\leq c_1 \max(|h_1(x)|, |h_2(x)|) + c_2 \max(|h_1(x)|, |h_2(x)|) \\ &\leq (c_1 + c_2) \max(|h_1(x)|, |h_2(x)|) \end{aligned}$$

- Setting $c = c_1 + c_2$ and $k = \max(k_1, k_2)$ yields the result $f(x) + g(x) \in \mathcal{O}(\max(h_1(x), h_2(x)))$

Example 24.5. Given that $n!$ is $\mathcal{O}(n^n)$, provide a Big-$\mathcal{O}$ expression for $n! + 34n^{10}$. Here n is nonnegative.

Example 24.5. Given that $n!$ is $\mathcal{O}(n^n)$, provide a Big-$\mathcal{O}$ expression for $n! + 34n^{10}$. Here n is nonnegative.

- $n!$ is $\mathcal{O}(n^n)$ and $34n^{10}$ is $\mathcal{O}(n^{10})$. So, $n! + 34n^{10}$ is $\mathcal{O}(\max(n^n, n^{10}))$

Properties of Big-$\mathcal{O}$: Products

A similar theorem exists for the Big-$\mathcal{O}$ of a product of functions.

Theorem 24.6. Let $f(x) \in \mathcal{O}(h_1(x))$ and $g(x) \in \mathcal{O}(h_2(x))$. Then $f(x)\,g(x) \in \mathcal{O}(h_1(x)\,h_2(x))$.

Example 24.7. Give big-$\mathcal{O}$ estimates for $n!$ and $\log(n!)$.

Properties of Big-$\mathcal{O}$: Products

A similar theorem exists for the Big-$\mathcal{O}$ of a product of functions.

Theorem 24.6. Let $f(x) \in \mathcal{O}(h_1(x))$ and $g(x) \in \mathcal{O}(h_2(x))$. Then $f(x)\, g(x) \in \mathcal{O}(h_1(x)\, h_2(x))$.

Example 24.7. Give big-$\mathcal{O}$ estimates for $n!$ and $\log(n!)$.

- **Answer.** $n! \in \mathcal{O}(n^n)$ and $\log(n!) \in \mathcal{O}(n \log n)$

Example 24.8. Give a big-$\mathcal{O}$ estimate for $f(x) = (x+1)\log(x^2+1) + 3x^2$, where x is any real number.

Growth Functions Commonly Used in Estimations

Big-$\mathcal{O}$ notation is used to estimate the number of operations needed to solve a problem using a specified procedure or algorithm. The functions used in these estimates often include the following:

$$1, \log n, n, n \log n, n^2, 2^n, n!$$

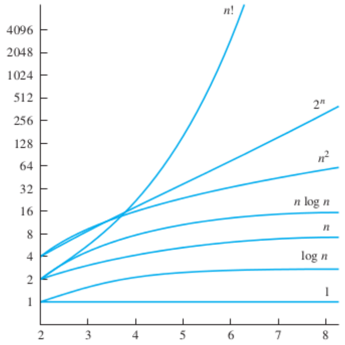


FIGURE **A Display of the Growth of Functions Commonly Used in Big-O Estimates.**

Exercise 24.9. Imagine you plan to invite a bunch of people to a dinner party. Let n be the number of people invited to the party. If you invite 10 people, you will have to shake hands ten times. If you double the number of guests, it will take you twice as long. This is a linear increase, and the time it will take you to shake everyone's hand can be expressed as $O(n)$.

Now, let's suppose everyone wants to shake hands, but for some strange reason *only one pair can shake hands at a time*. As n increases, how much longer will this meet and greet take? Give a big-O estimate for the *meet-and-greet* time and justify your answer. (Hint: How many pairs are there?)