ECE430.217 Data Structures

Asymptotic Analysis

Weiss Book Chapter 2

Byoungyoung Lee
https://compsec.snu.ac.kr
byoungyoung@snu.ac.kr

Lecture slides were prepared based on Douglas W. Harder's notes (**dwharder@alumni.uwaterloo.ca).**

Outline

In this topic, we will look at:

- Justification for analysis
- Quadratic and polynomial growth
- Counting machine instructions
- Landau symbols
- Big-Θ as an equivalence relation
- Little-o as a weak ordering

Goal: Algorithm Analysis

Suppose we have two algorithms, how can we tell which is better?

We could implement both algorithms, run them both

- Implementation Cost
  - How long should I run? How many times should I run?
- Input dependency
  - Depending on input, your algorithm may take a different time

Preferably, we should analyze them mathematically

- *Algorithm analysis*

Approach: Analyzing Algorithms w.r.t. a variable

In general, we will always analyze algorithms with respect to one or more variables

One variable:

- The number of items n currently stored in an array or other data structure
- The number of items n expected to be stored in an array or other data structure
- The dimensions of an $n \times n$ matrix

Multiple variables:

- Dealing with n objects stored in m memory locations
- Multiplying a $k \times m$ and an $m \times n$ matrix
- Dealing with sparse matrices of size $n \times n$ with m non-zero entries

Find Maximum Value in an Array

For example, the time taken to find the largest object in an array of n random integers will take n operations

```
int find_max( int *array, int n ) {
    int max = array[0];

    for ( int i = 1; i < n; ++i ) {
        if ( array[i] > max ) {
            max = array[i];
        }
    }

    return max;
}
```

Find Maximum Value in an Array (STL)

```
#include <vector>

int find_max( std::vector<int> array ) {
    if ( array.size() == 0 ) {
        throw underflow();
    }

    int max = array[0];

    for ( int i = 1; i < array.size(); ++i ) {
        if ( array[i] > max ) {
            max = array[i];
        }
    }

    return max;
}
```

Linear and Binary Search

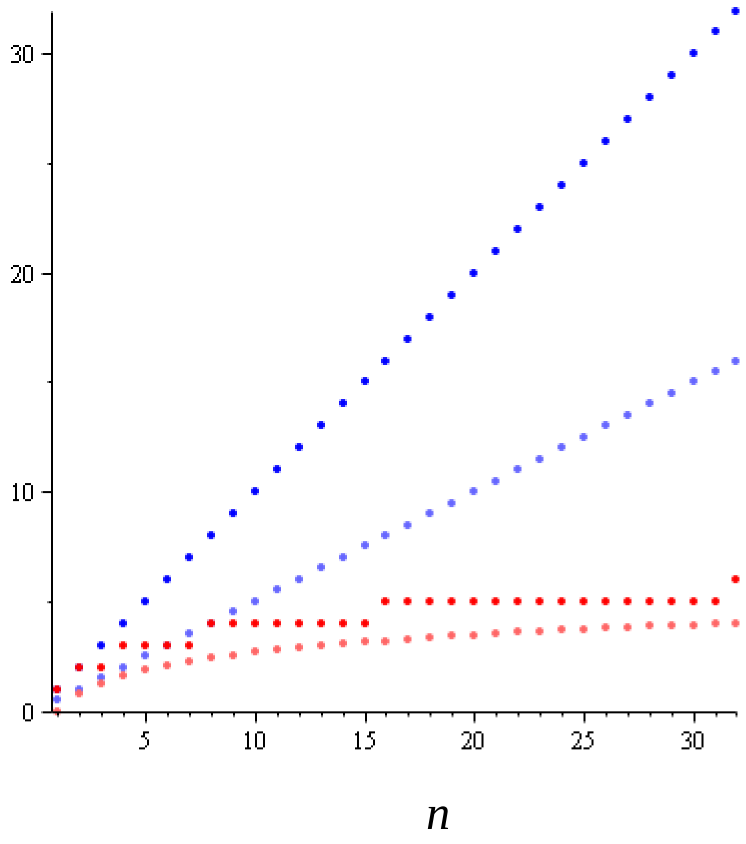
There are other algorithms which become significantly faster as the problem size increases

This plot shows maximum and average number of comparisons to find an entry in a sorted array of size n

- Linear search
- Binary search

Q. Maximum number?

Average number?



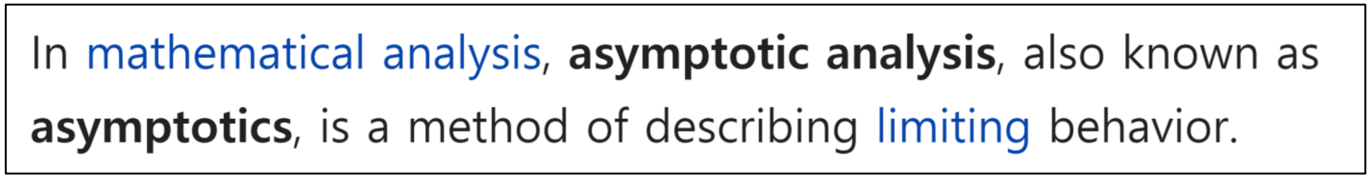
Asymptotic Analysis

Given an algorithm:

- We need to be able to describe max/avg values mathematically
- We need well-defined descriptions of the algorithm
- We need to do this in a machine-independent way

For this, we use **Landau symbols**

and the associated **asymptotic analysis**



In mathematical analysis, **asymptotic analysis**, also known as **asymptotics**, is a method of describing limiting behavior.

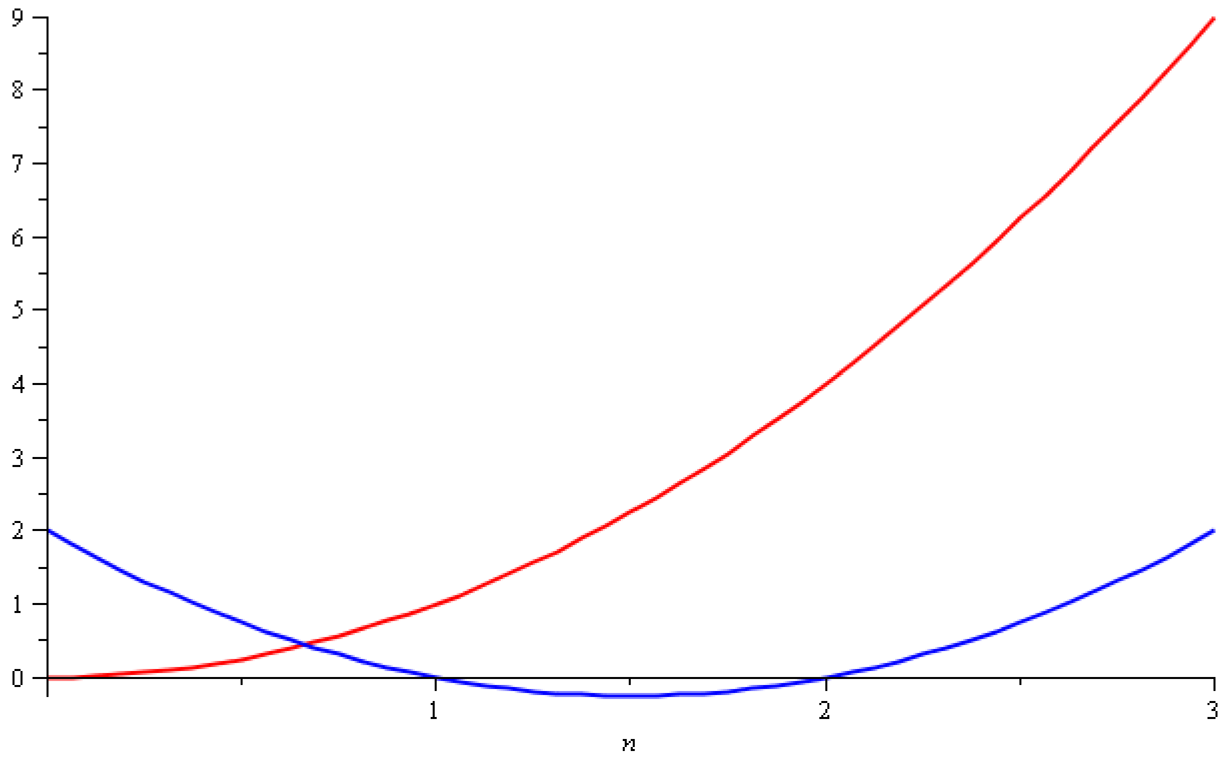
https://en.wikipedia.org/wiki/Asymptotic_analysis

Quadratic Growth

Consider the two functions

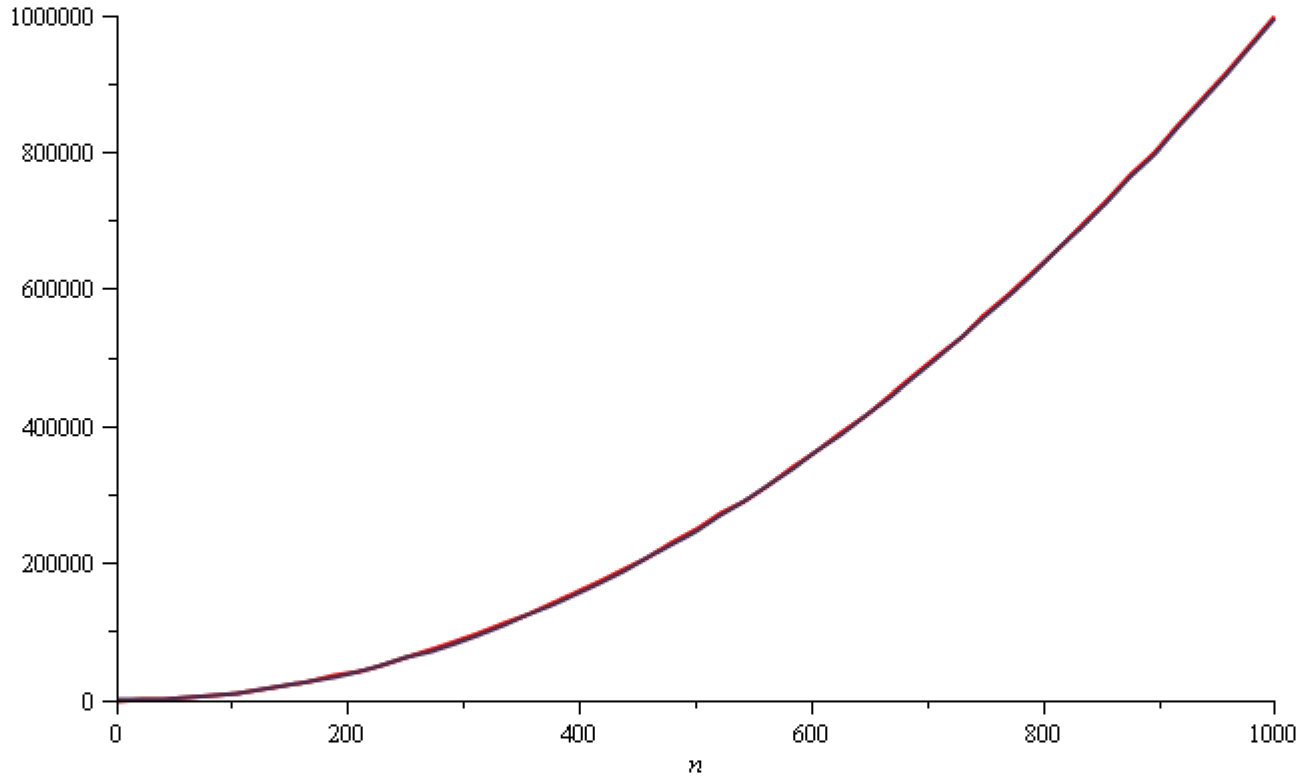
$f(n) = n^2$ and $g(n) = n^2 - 3n + 2$

Around $n = 0$, they look very different



Quadratic Growth

In the range $n = [0, 1000]$, they are (relatively) indistinguishable:



Quadratic Growth

The absolute difference is large though:

$$f(1000) = 1\ 000\ 000$$

$$g(1000) = \quad 997\ 002$$

The relative difference is very small

$$\left| \frac{f(1000) - g(1000)}{f(1000)} \right| = 0.002998 < 0.3\%$$

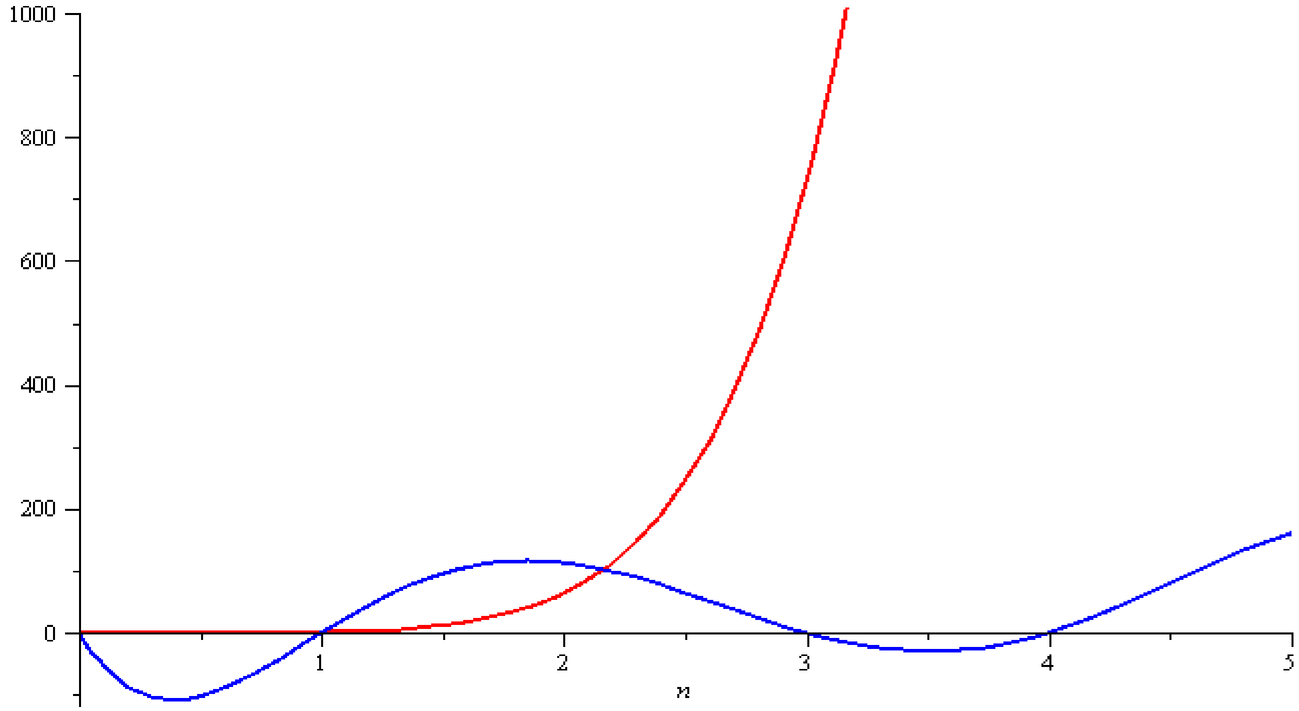
and this difference goes to zero as $n \to \infty$

Polynomial Growth

To demonstrate with another example,

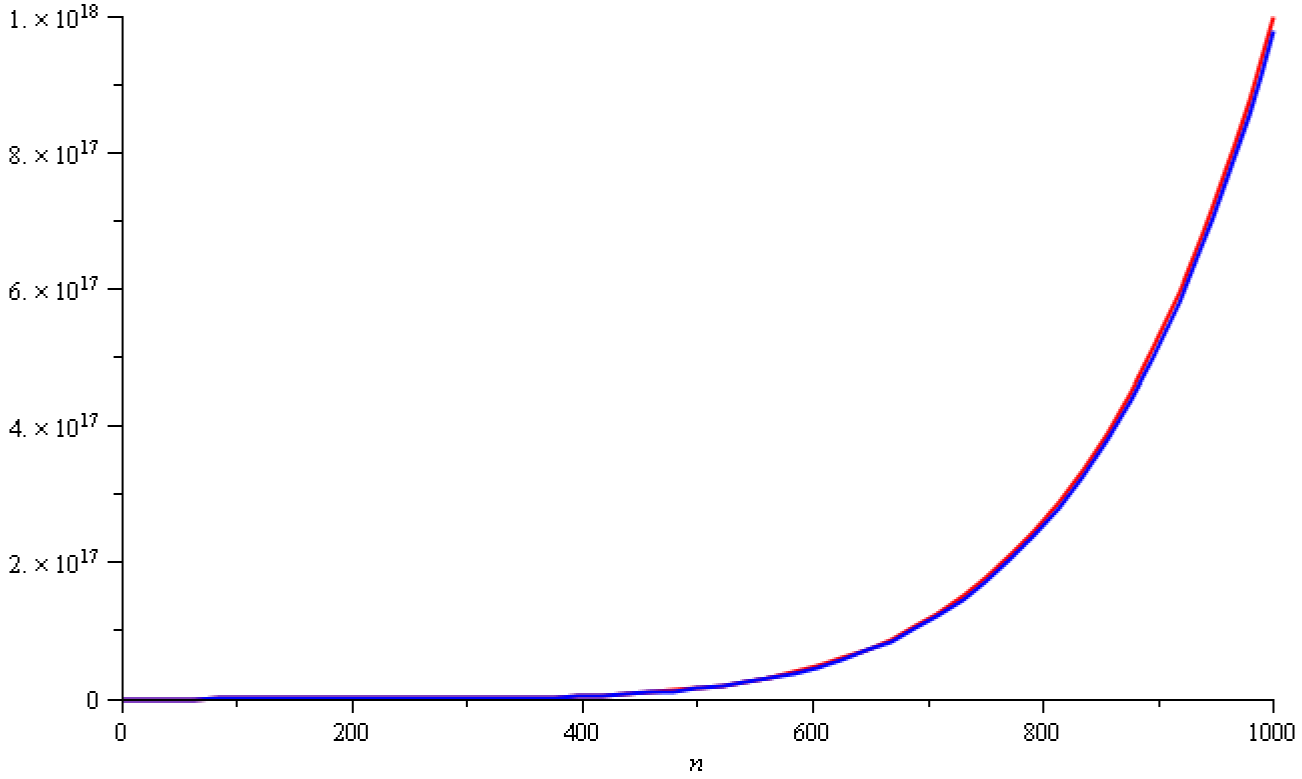
$f(n) = n^6$ and $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$

Around $n = 0$, they are very different



Polynomial Growth

Still, around $n = 1000$, the relative difference is less than 3%



Polynomial Growth

The justification for both pairs of polynomials being similar is that, in both cases, they each had **the same leading term**:

n^2 in the first case, n^6 in the second

Suppose however, **that the coefficients of the leading terms were different**

- In this case, both functions would exhibit the same rate of growth, however, one would always be **proportionally larger**

Examples

We will now look at an example:

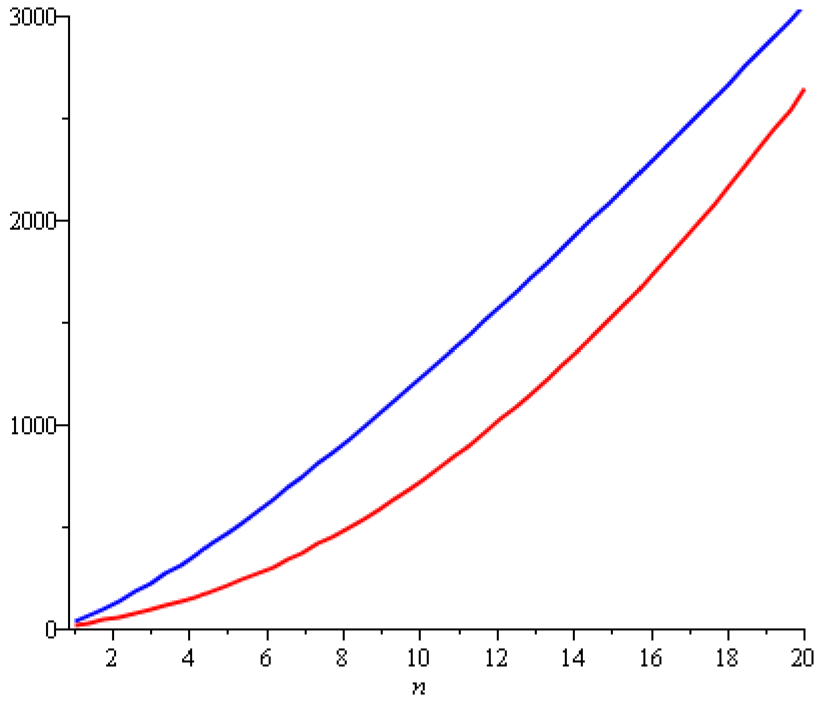
- A comparison of **insertion sort** and **quicksort**

Counting Instructions

Insertion sort is growing at a rate of n^2

while quicksort grows at a rate of $n \lg(n)$

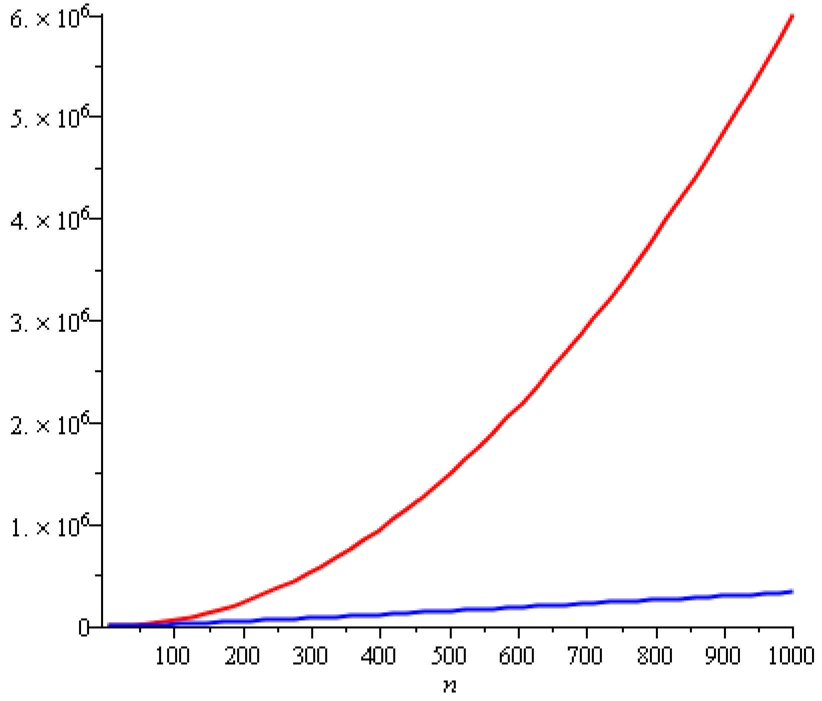
- The plot suggests it is more useful to use insertion sort when sorting small lists—quicksort has a more overhead



Counting Instructions

If the size of the list is large (greater than 20), the additional overhead of quicksort quickly becomes insignificant

- The **quicksort** algorithm becomes significantly more efficient



Weak ordering

Consider the following definitions:

– We will consider two functions to be equivalent, $f \sim g$, if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where} \quad 0 < c < \infty$$

– We will state that $f < g$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

For functions we are interested in, these define a weak ordering

Landau Symbols: Big-Oh

A function $\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n))$ if there exists N and c such that

$$\mathrm{f}(n) < c\,\mathrm{g}(n)$$

for all $n > N$

- The function $\mathrm{f}(n)$ has **a rate of growth no greater than** that of $\mathrm{g}(n)$

Landau Symbols

Assumptions before we begin:

- Our functions will describe the time or memory required to solve a problem of size n
- We conclude we are restricting ourselves to certain functions:
  - They are defined for $n \geq 0$
  - They are strictly positive for all n
    - In fact, $\mathrm{f}(n) > c$ for some value $c > 0$
    - That is, any problem requires at least one instruction and byte
  - They are increasing (monotonic increasing)

Landau Symbols: Big-Theta

Another Landau symbol is Θ

A function $f(n) = \Theta(g(n))$ if there exist positive N, c_1, and c_2 such that

$$c_1\, g(n) < f(n) < c_2\, g(n)$$

for all $n > N$

- The function $f(n)$ has **a rate of growth equal to** that of $g(n)$

Landau Symbols : Big-Theta

Claim If $\lim_{n\to\infty}\frac{\mathrm{f}(n)}{\mathrm{g}(n)} = c$ where $0 < c < \infty$, it follows that $\mathrm{f}(n) = \Theta(\mathrm{g}(n))$.

Proof Suppose that $\mathrm{f}(n)$ and $\mathrm{g}(n)$ satisfy $\lim_{n\to\infty}\frac{\mathrm{f}(n)}{\mathrm{g}(n)} = c$

From the definition, this means given $c > \varepsilon > 0$, there

exists an $N > 0$ such that $\left|\frac{\mathrm{f}(n)}{\mathrm{g}(n)} - c\right| < \varepsilon$ whenever $n > N$

That is,

$$c - \varepsilon < \frac{\mathrm{f}(n)}{\mathrm{g}(n)} < c + \varepsilon$$

$$\mathrm{g}(n)\left(c - \varepsilon\right) < \mathrm{f}(n) < \mathrm{g}(n)\left(c + \varepsilon\right)$$

Landau Symbols

We have a similar definition for $\mathbf{O}$:

If $\lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = c$ where $0 \le c < \infty$, it follows that $\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n))$

There are other possibilities we would like to describe:

If $\lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = 0$, we will say $\mathrm{f}(n) = \mathbf{o}(\mathrm{g}(n))$

– The function $\mathrm{f}(n)$ has a rate of growth less than that of $\mathrm{g}(n)$

We would also like to describe the opposite cases:

– The function $\mathrm{f}(n)$ has a rate of growth greater than that of $\mathrm{g}(n)$
– The function $\mathrm{f}(n)$ has a rate of growth greater than or equal to that of $\mathrm{g}(n)$

Landau Symbols

We will at times use five possible descriptions

$$\mathrm{f}(n) = \mathbf{o}(\mathrm{g}(n)) \qquad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = 0$$

$$\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n)) \qquad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} < \infty$$

$$\mathrm{f}(n) = \mathbf{\Theta}(\mathrm{g}(n)) \qquad 0 < \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} < \infty$$

$$\mathrm{f}(n) = \mathbf{\Omega}(\mathrm{g}(n)) \qquad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} > 0$$

$$\mathrm{f}(n) = \boldsymbol{\omega}(\mathrm{g}(n)) \qquad \lim_{n\to\infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} = \infty$$

Landau Symbols

For the functions we are interested in, it can be said that

$\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n))$ is equivalent to $\mathrm{f}(n) = \mathbf{\Theta}(\mathrm{g}(n))$ or $\mathrm{f}(n) = \mathbf{o}(\mathrm{g}(n))$

and

$\mathrm{f}(n) = \mathbf{\Omega}(\mathrm{g}(n))$ is equivalent to $\mathrm{f}(n) = \mathbf{\Theta}(\mathrm{g}(n))$ or $\mathrm{f}(n) = \boldsymbol{\omega}(\mathrm{g}(n))$

Landau Symbols

Some other observations we can make are:

$$f(n) = \mathbf{\Theta}(g(n)) \Leftrightarrow g(n) = \mathbf{\Theta}(f(n))$$

$$f(n) = \mathbf{O}(g(n)) \Leftrightarrow g(n) = \mathbf{\Omega}(f(n))$$

$$f(n) = \mathbf{o}(g(n)) \Leftrightarrow g(n) = \boldsymbol{\omega}(f(n))$$

Big-Θ as an Equivalence Relation

If we look at the first relationship, we notice that
$f(n) = \Theta(g(n))$ seems to describe an equivalence relation:

1. $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
2. $f(n) = \Theta(f(n))$
3. If $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$, it follows that $f(n) = \Theta(h(n))$

Consequently, we can group all functions into equivalence classes, where all functions within one class are big-theta Θ of each other

Big-Θ as an Equivalence Relation

For example, all of

$$n^2 \qquad 100000\, n^2 - 4\, n + 19 \qquad n^2 + 1000000$$

$$323\, n^2 - 4\, n \ln(n) + 43\, n + 10 \qquad 42n^2 + 32$$

$$n^2 + 61\, n \ln^2(n) + 7n + 14 \ln^3(n) + \ln(n)$$

are big-Θ of each other

E.g., $42n^2 + 32 = \Theta(\, 323\, n^2 - 4\, n \ln(n) + 43\, n + 10\,)$

Big-Θ as an Equivalence Relation

The most common classes are given names:

$\Theta(1)$	constant
$\Theta(\ln(n))$	logarithmic
$\Theta(n)$	linear
$\Theta(n \ln(n))$	“n log n”
$\Theta(n^2)$	quadratic
$\Theta(n^3)$	cubic
$2^n, e^n, 4^n, \ldots$	exponential

Logarithms and Exponentials

Recall that all logarithms are scalar multiples of each other

- Therefore $\log_b(n) = \boldsymbol{\Theta}(\ln(n))$ for any base b

Alternatively, there is no single equivalence class for exponential functions:

- If $1 < a < b$, $\lim\limits_{n\to\infty} \frac{a^n}{b^n} = \lim\limits_{n\to\infty} \left(\frac{a}{b}\right)^n = 0$
- Therefore $a^n = \mathbf{o}(b^n)$

Note: we will see that it is almost universally undesirable to have an exponentially growing function!

Logarithms and Exponentials

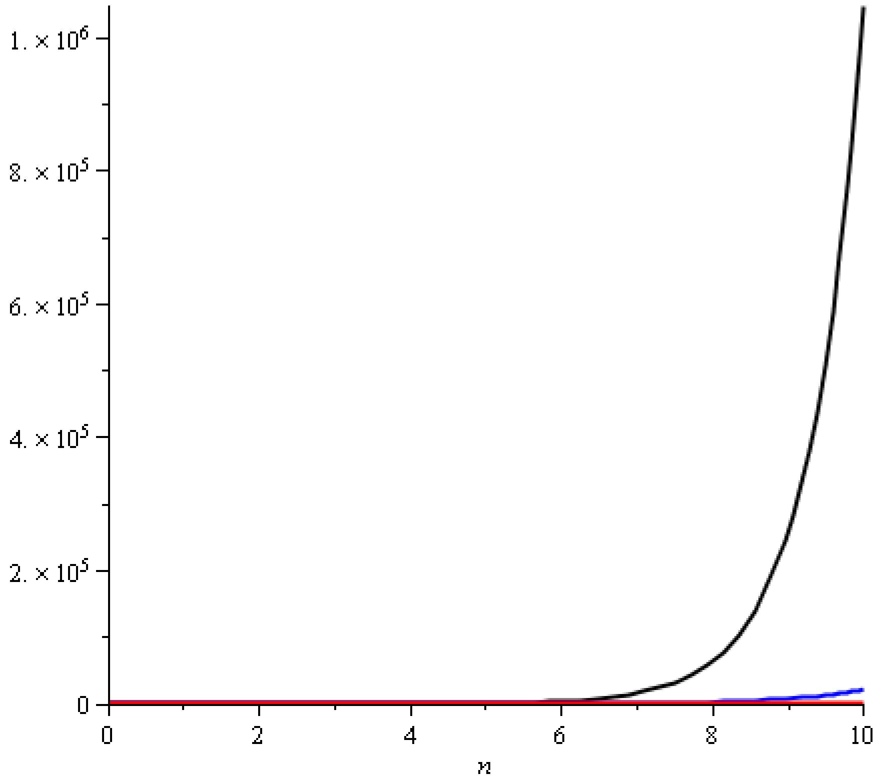
Plotting 2^n, e^n, and 4^n on the range $[1, 10]$ already shows how significantly different the functions grow

Note:

$2^{10} = 1024$

$e^{10} \approx 22\,026$

$4^{10} = 1\,048\,576$



Little-**o** as a Weak Ordering

We can show that, for example

$\ln(n) = \mathbf{o}(n^p)$

for any $p > 0$

Proof: Using l'Hôpital's rule, we have

$$\lim_{n\to\infty}\frac{\ln(n)}{n^p} = \lim_{n\to\infty}\frac{1/n}{pn^{p-1}} = \lim_{n\to\infty}\frac{1}{pn^p} = \frac{1}{p}\lim_{n\to\infty}n^{-p} = 0$$

Little-o as a Weak Ordering

Other observations:

- If p and q are real positive numbers where $p < q$, it follows that

$$n^p = \mathbf{o}(n^q)$$

- For example, matrix-matrix multiplication is $\mathbf{\Theta}(n^3)$ but a refined algorithm is $\mathbf{\Theta}(n^{\lg(7)})$ where $\lg(7) \approx 2.81$

- **Exercise**: $n^p = \mathbf{o}(\ln(n)n^p)$, but $\ln(n)n^p = \mathbf{o}(n^q)$
  - n^p has a slower rate of growth than $\ln(n)n^p$, but
  - $\ln(n)n^p$ has a slower rate of growth than n^q for $p < q$

Little-$\mathbf{o}$ as a Weak Ordering

If we restrict ourselves to functions $f(n)$ which are $\mathbf{\Theta}(n^p)$ and $\mathbf{\Theta}(\ln(n)n^p)$, we note:

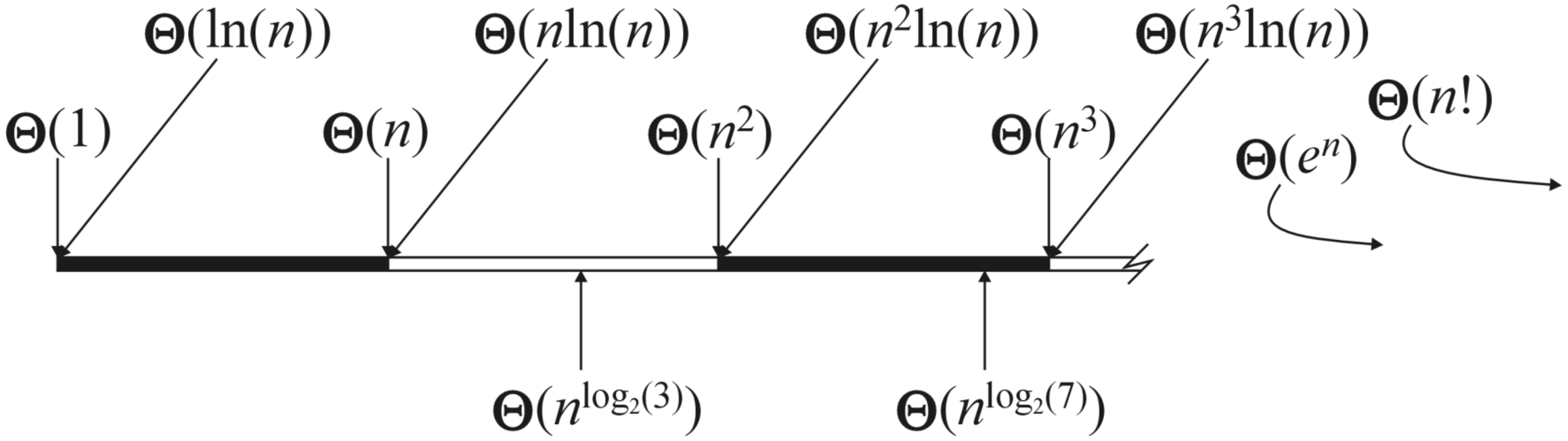
- It is never true that $f(n) = \mathbf{o}(f(n))$
- If $f(n) \neq \mathbf{\Theta}(g(n))$, it follows that either

  $f(n) = \mathbf{o}(g(n))$ or $g(n) = \mathbf{o}(f(n))$
- If $f(n) = \mathbf{o}(g(n))$ and $g(n) = \mathbf{o}(h(n))$, it follows that $f(n) = \mathbf{o}(h(n))$

This defines a weak ordering!

Little-o as a Weak Ordering

Graphically, we can shown this relationship by marking these against the real line



Algorithms Analysis

We will use Landau symbols to describe the complexity of algorithms

An algorithm is said to have ***polynomial time complexity***
if its run-time may be described by $\mathbf{O(n^d)}$ for some fixed $d \geq 0$

- We will consider such algorithms to be *efficient*

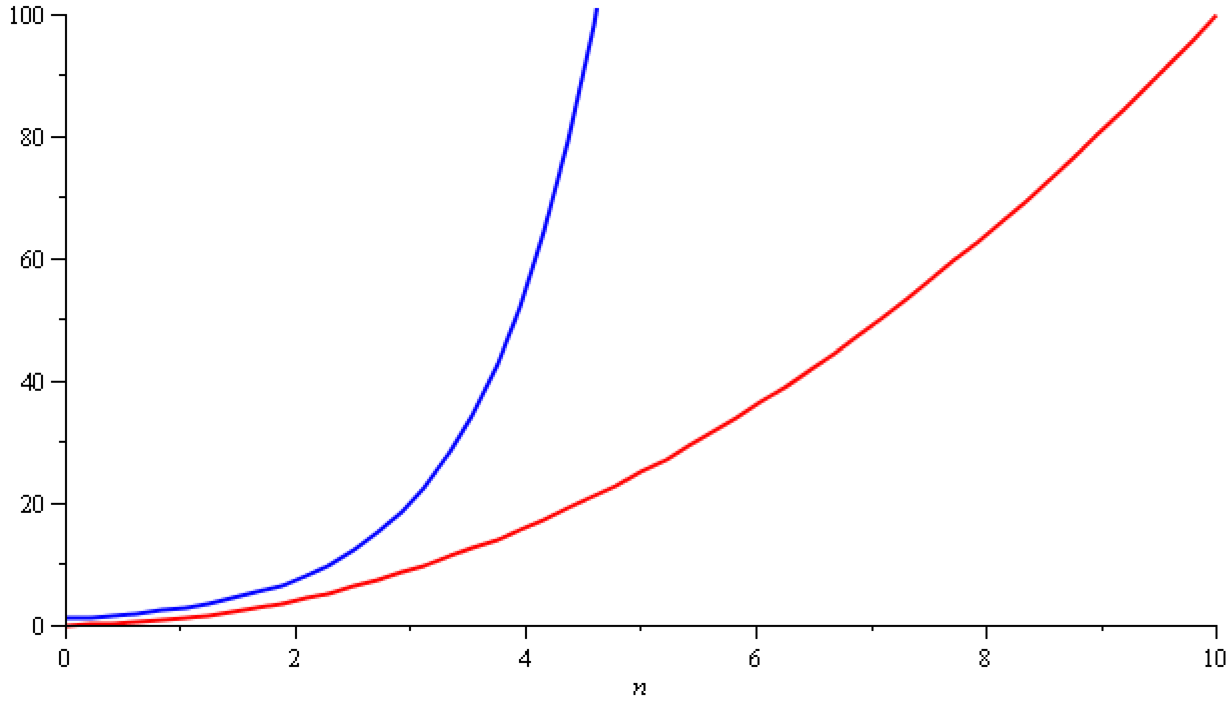
Problems that have **no known polynomial-time algorithms** are said to be ***intractable***

- Traveling salesman problem: find the shortest path that visits n cities
- Best run time: $\Theta(n^2 2^n)$
- Typically called as **NP (Nondeterministic Polynomial time)**

Algorithm Analysis

In general, you don’t want to implement exponential-time or exponential-memory algorithms

– Warning: don’t call a quadratic curve “exponential”



Summary

In this class, we have:

- Discussed Landau symbols: o O Θ Ω ω
- Discussed how to use these
- Looked at the equivalence relations

References

Wikipedia, https://en.wikipedia.org/wiki/Mathematical_induction