

Asymptotic Analysis



WATERLOO
ENGINEERING

Douglas Wilhelm Harder, M.Math. LEL
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

ece.uwaterloo.ca
dwharder@alumni.uwaterloo.ca

© 2006-2013 by Douglas Wilhelm Harder. Some rights reserved.





2.3

Background

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

We could implement both algorithms, run them both

- Expensive and error prone

Preferably, we should analyze them mathematically

- *Algorithm analysis*



2.3.1

Maximum Value

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;  
}
```



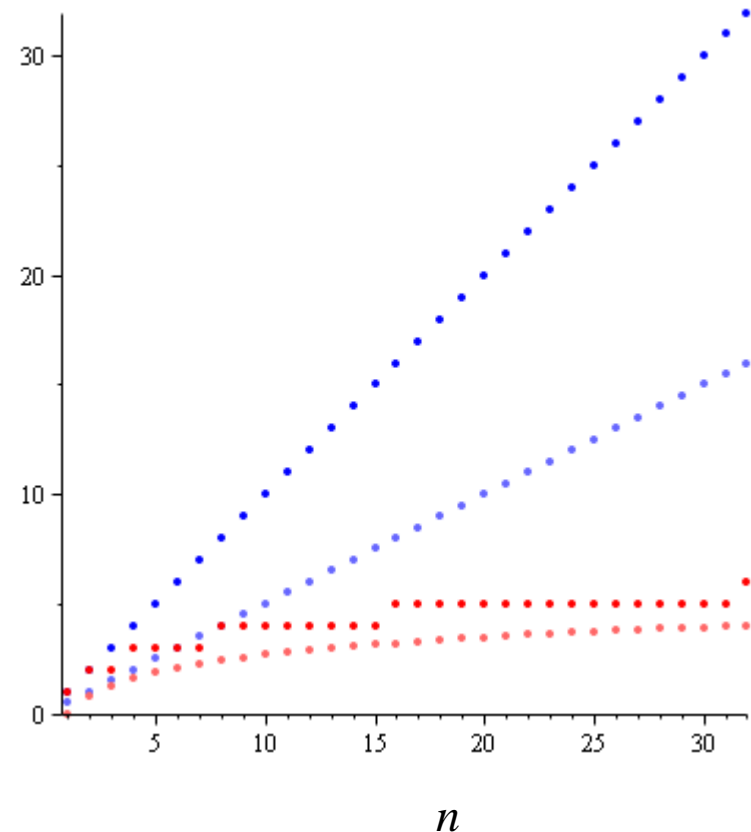
2.3.2

Linear and binary search

There are other algorithms which are 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





2.3.3

Asymptotic Analysis

Given an algorithm:

- We need to be able to describe these values mathematically
- We need a systematic means of using the description of the algorithm together with the properties of an associated data structure
- We need to do this in a machine-independent way

For this, we need Landau symbols and the associated asymptotic analysis



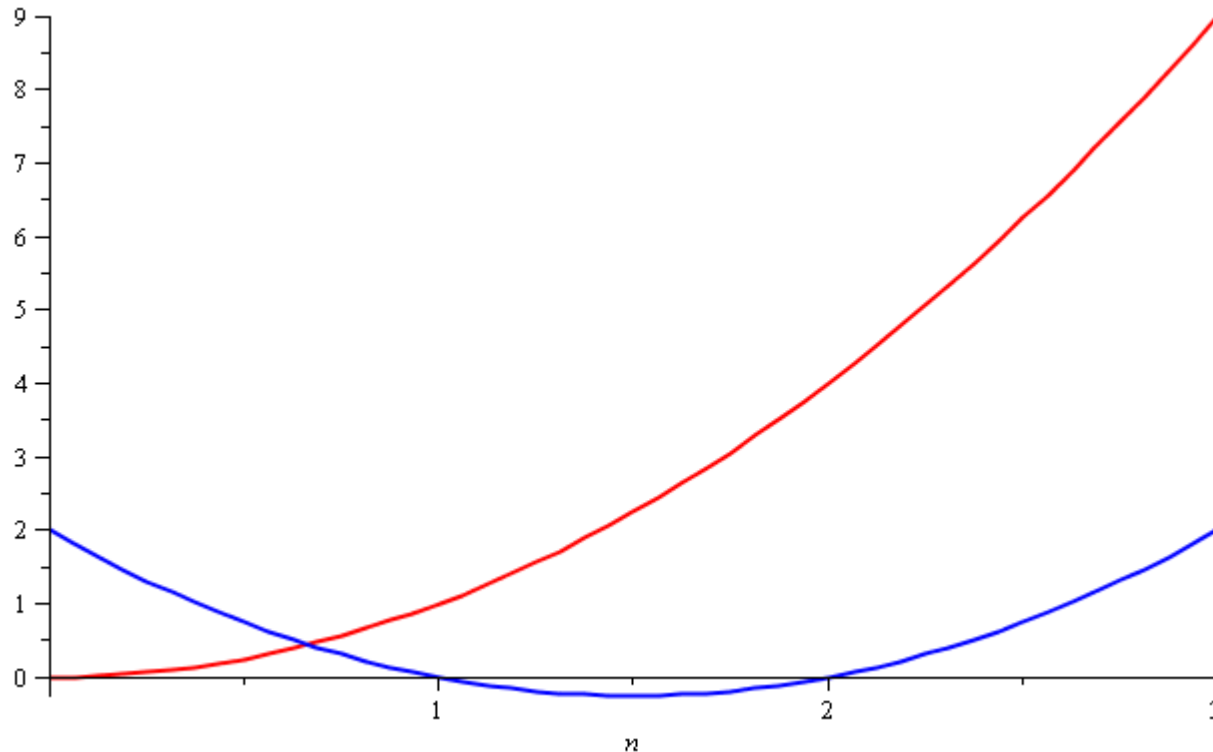
2.3.3

Quadratic Growth

Consider the two functions

$$f(n) = n^2 \text{ and } g(n) = n^2 - 3n + 2$$

Around $n = 0$, they look very different

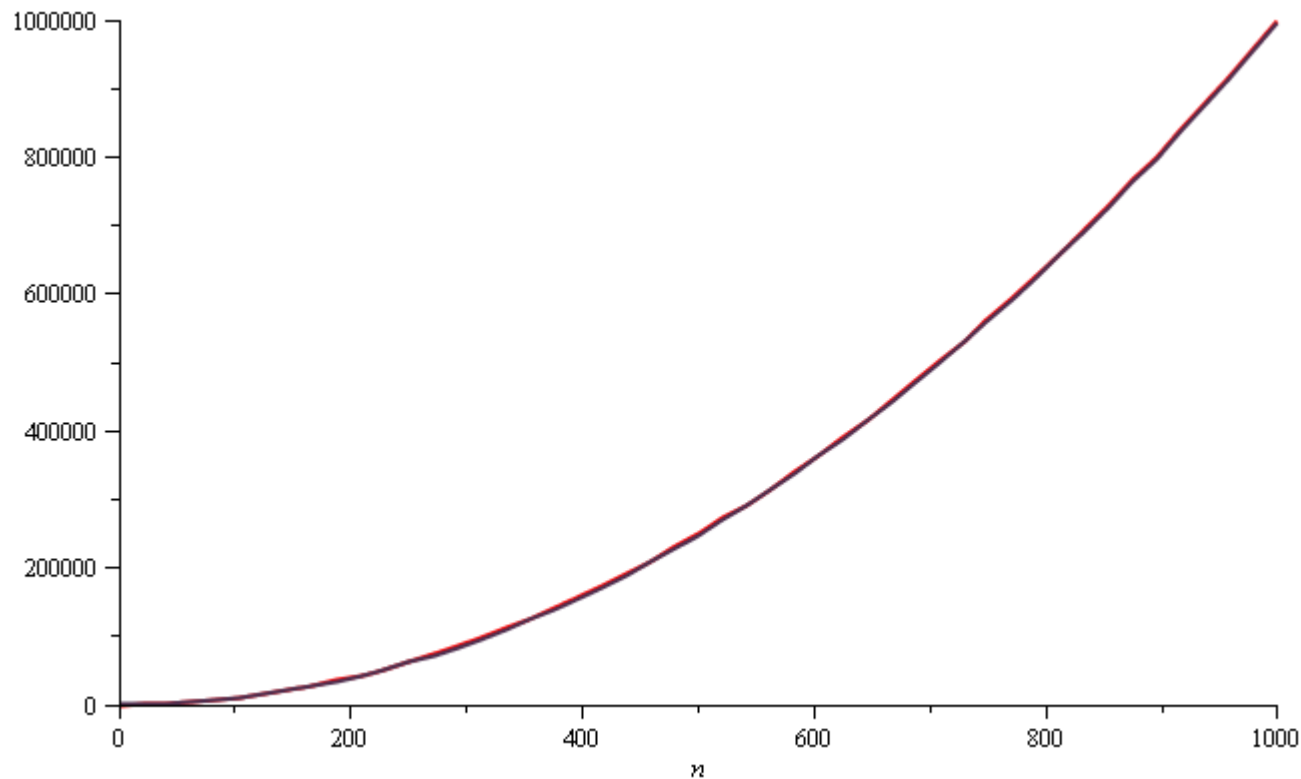




2.3.3

Quadratic Growth

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





2.3.3

Quadratic Growth

The absolute difference is large, for example,

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

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

but 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 $n \rightarrow \infty$



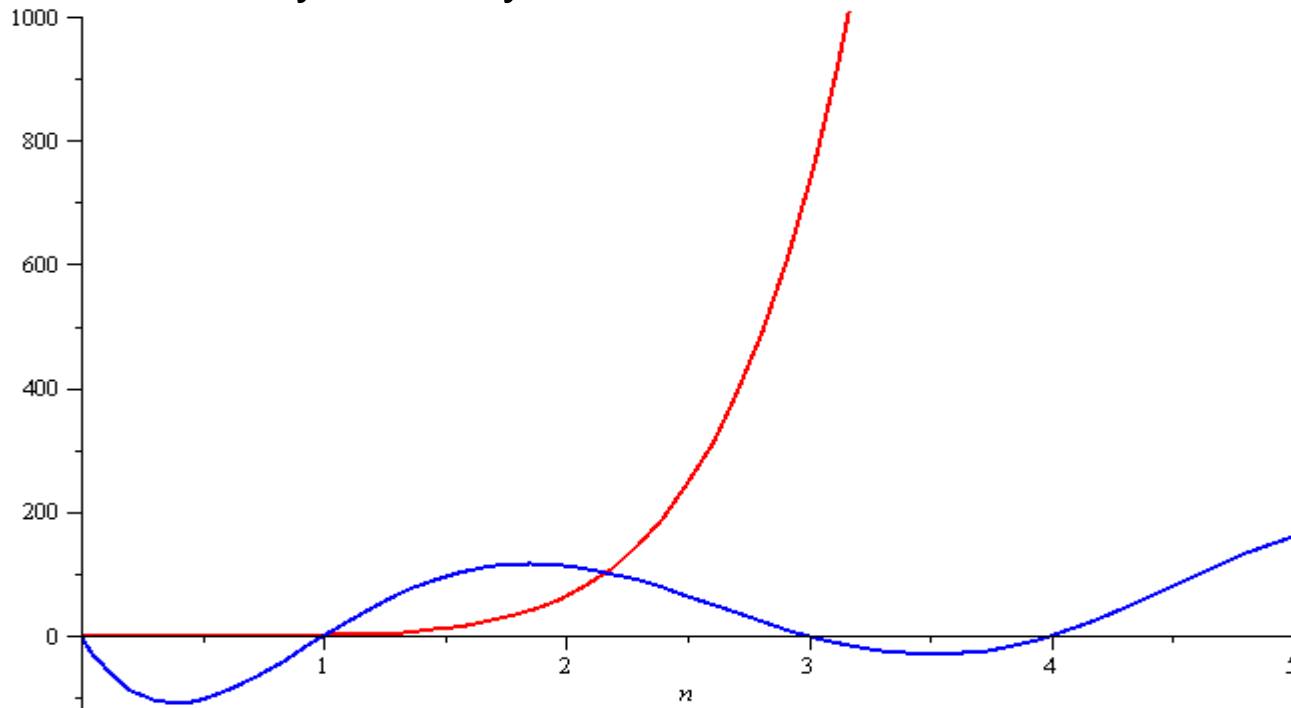
2.3.3

Polynomial Growth

To demonstrate with another example,

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

Around $n = 0$, they are very different

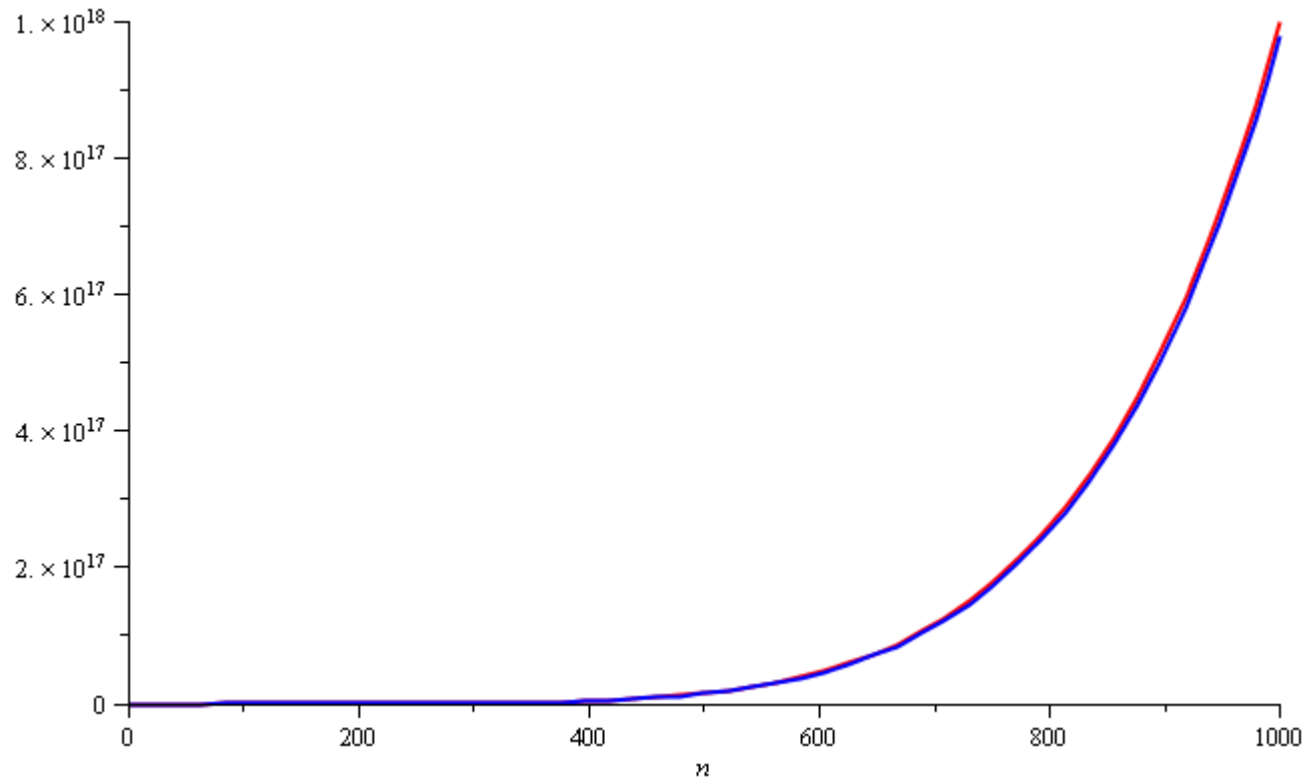




2.3.3

Polynomial Growth

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





2.3.3

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



2.3.4.1

Counting Instructions

Suppose we had two algorithms which sorted a list of size n and the run time (in μs) is given by

$$b_{\text{worst}}(n) = 4.7n^2 - 0.5n + 5$$

Bubble sort

$$b_{\text{best}}(n) = 3.8n^2 + 0.5n + 5$$

$$s(n) = 4n^2 + 14n + 12$$

Selection sort

The smaller the value, the fewer instructions are run

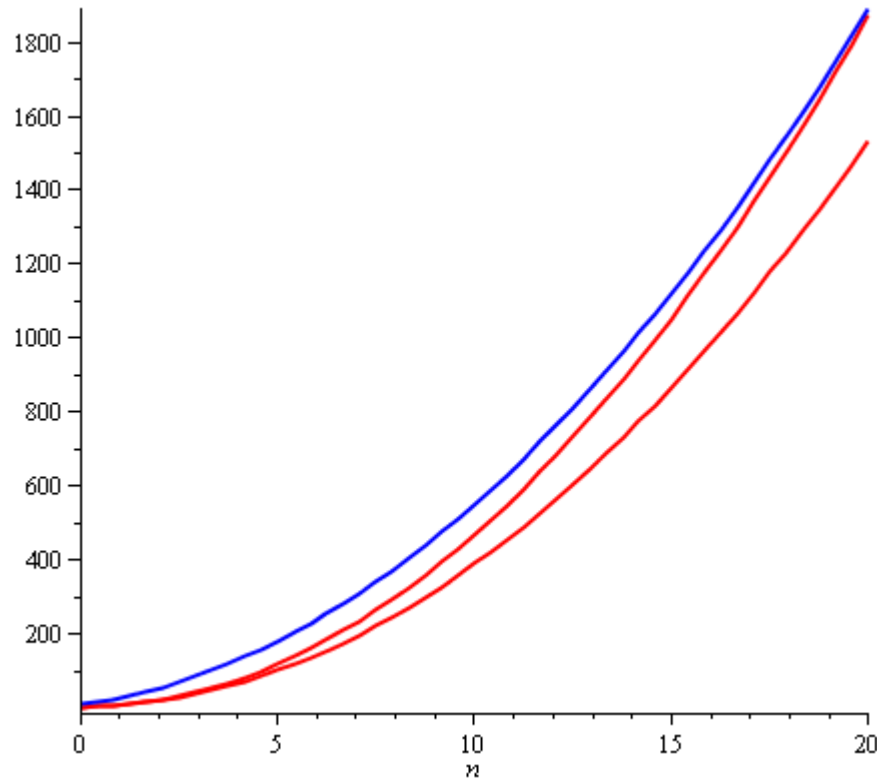
- For $n \leq 21$, $b_{\text{worst}}(n) < s(n)$
- For $n \geq 22$, $b_{\text{worst}}(n) > s(n)$



2.3.4.1

Counting Instructions

With small values of n , the algorithm described by $s(n)$ requires more instructions than even the worst-case for bubble sort

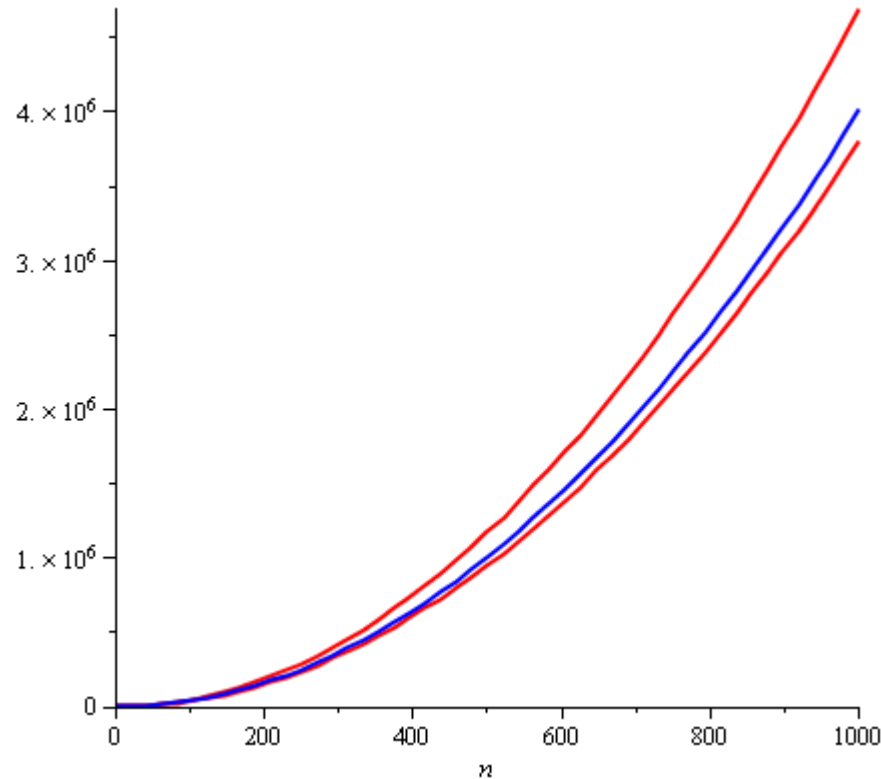




2.3.4.1

Counting Instructions

Near $n = 1000$, $b_{\text{worst}}(n) \approx 1.175 s(n)$ and $b_{\text{best}}(n) \approx 0.95 s(n)$





2.3.4.1

Counting Instructions

Is this a serious difference between these two algorithms?

Because we can count the number instructions, we can also estimate how much time is required to run one of these algorithms on a computer

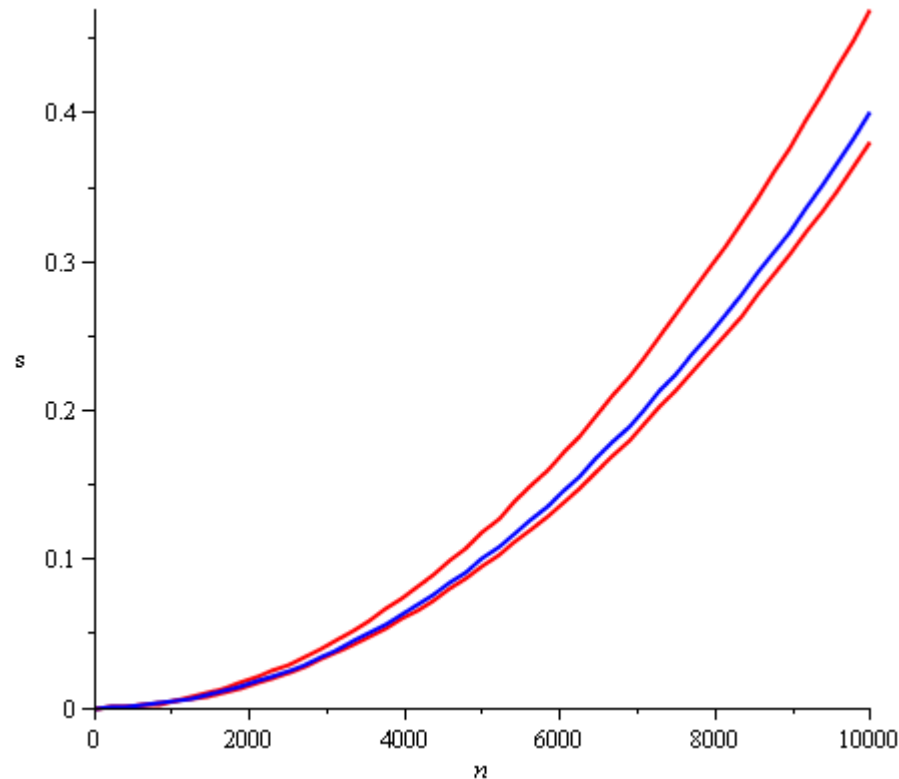


2.3.4.1

Counting Instructions

Suppose we have a 1 GHz computer

- The time (s) required to sort a list of up to $n = 10\,000$ objects is under half a second

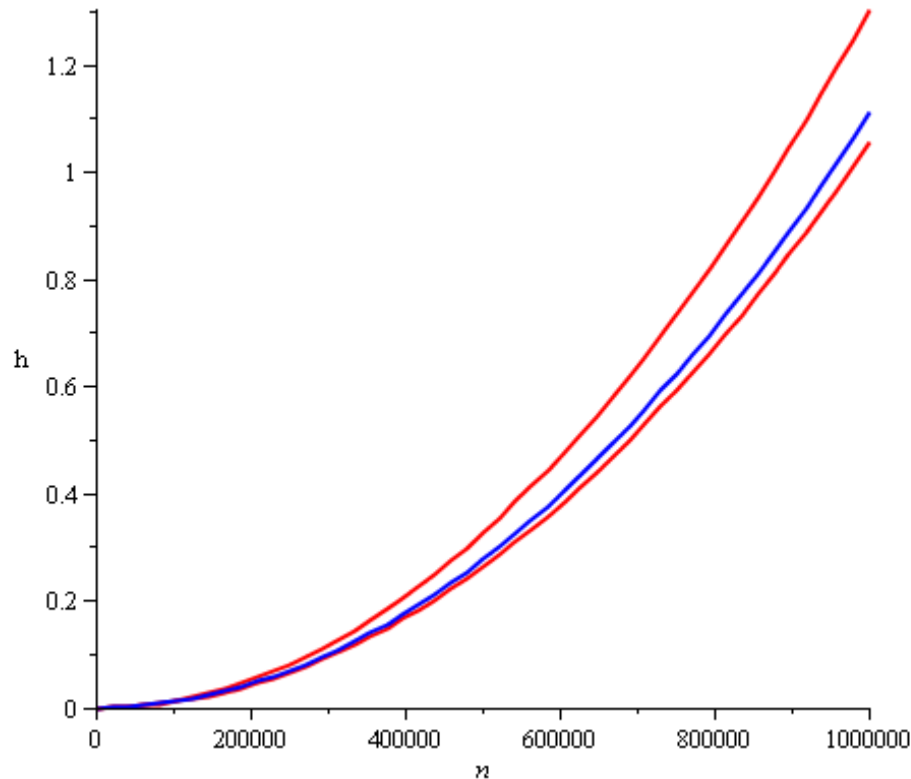




2.3.4.1

Counting Instructions

To sort a list with one million elements, it will take about 1 h



Bubble sort could, under some conditions, be 200 s faster

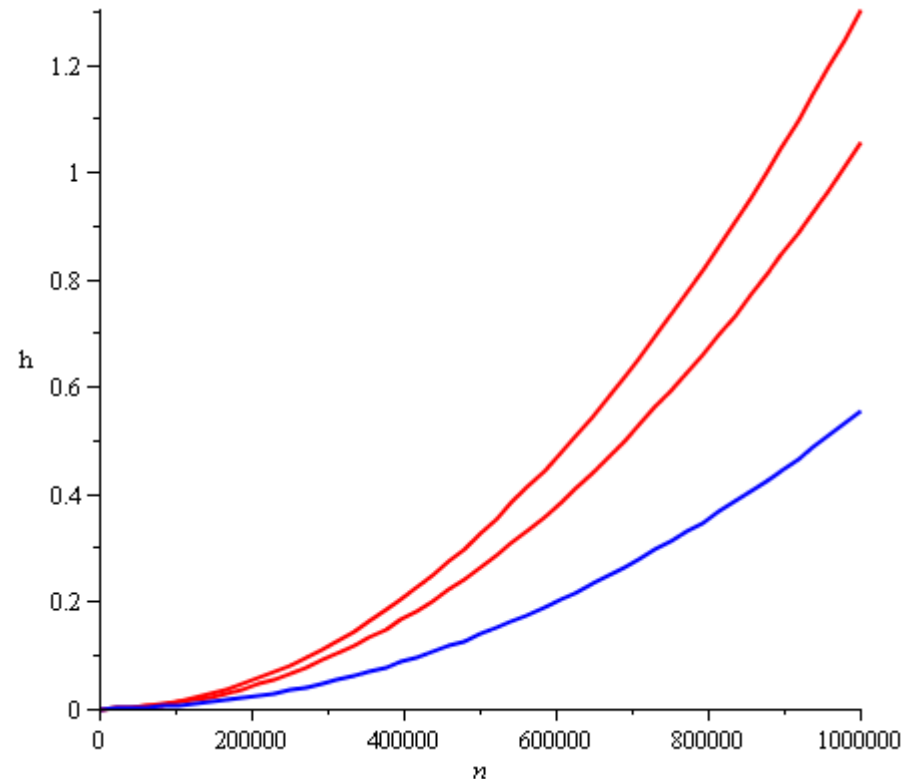


2.3.4.1

Counting Instructions

How about running selection sort on a faster computer?

- For large values of n , selection sort on a faster computer will always be faster than bubble sort





2.3.4.1

Counting Instructions

Justification?

- If $f(n) = a_k n^k + \dots$ and $g(n) = b_k n^k + \dots$,
for large enough n , it will always be true that

$$f(n) < M g(n)$$

where we choose

$$M = a_k / b_k + 1$$

In this case, we only need a computer which is M times faster (or slower)

Question:

- Is a linear search comparable to a binary search?
- Can we just run a linear search on a slower computer?

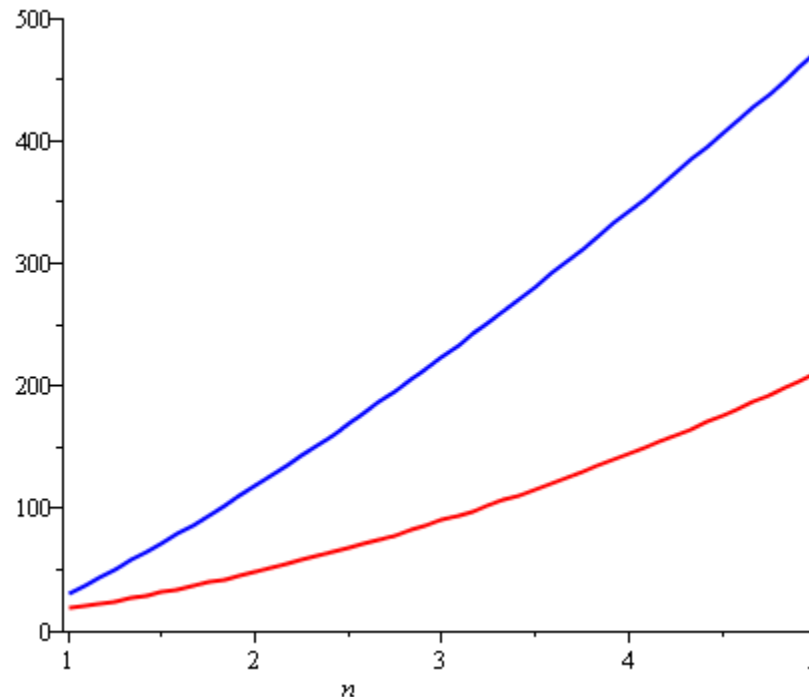


2.3.4.2

Counting Instructions

As another example:

- Compare the number of instructions required for insertion sort and for quicksort
- Both functions are concave up, although one more than the other



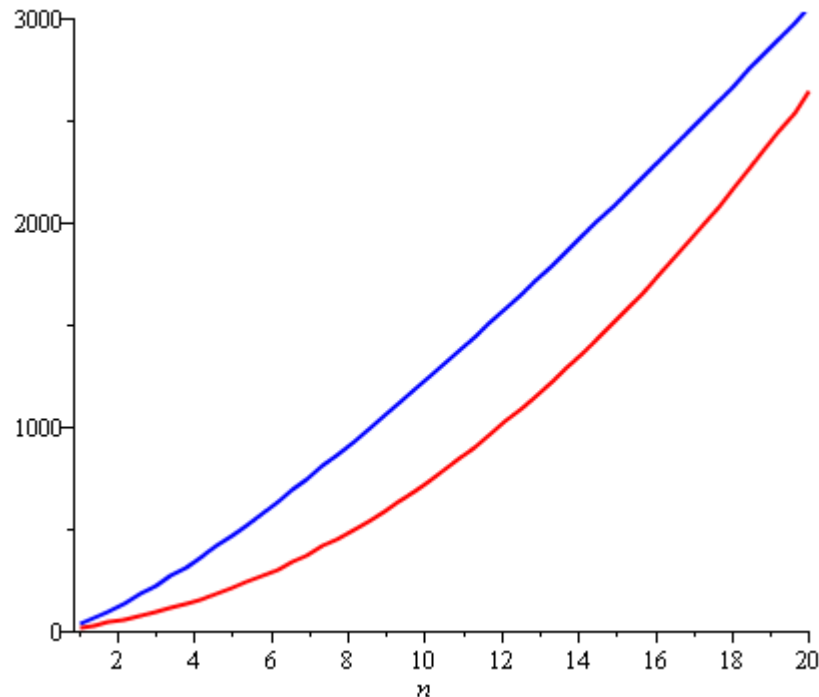


2.3.4.2

Counting Instructions

Insertion sort, however, is growing at a rate of n^2 while quicksort grows at a rate of $n \lg(n)$

- Never-the-less, the graphic suggests it is more useful to use insertion sort when sorting small lists—quicksort has a large overhead



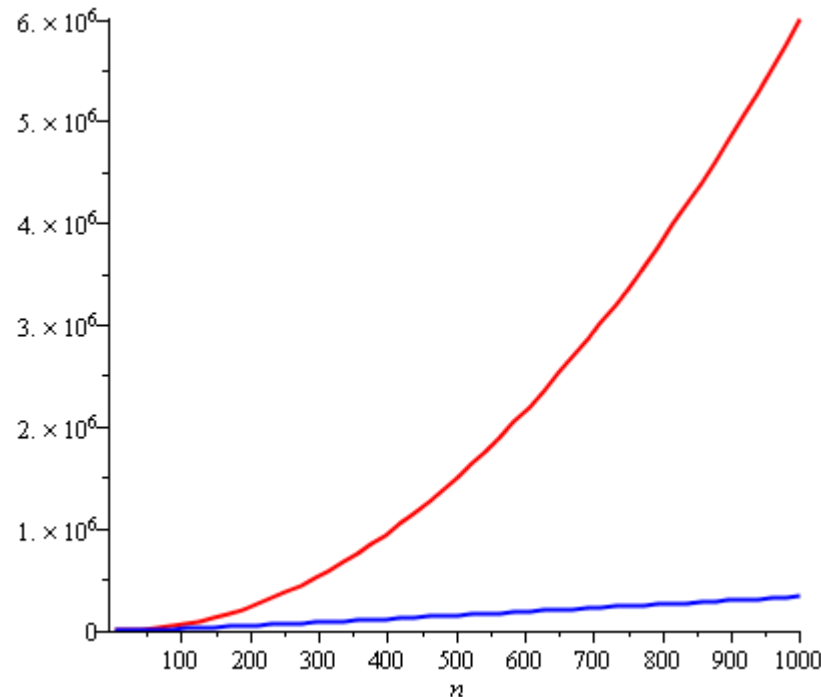


2.3.4.2

Counting Instructions

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

- The quicksort algorithm becomes significantly more efficient
- Question: can we just buy a faster computer?





2.3.5

Weak ordering

Consider the following definitions:

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

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

- We will state that $f < g$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

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



2.3.5

Weak ordering

Let $f(n)$ and $g(n)$ describe either the run-time of two algorithms

- If $f(n) \sim g(n)$, then it is always possible to improve the performance of one function over the other by purchasing a faster computer
- If $f(n) < g(n)$, then you can never purchase a computer fast enough so that the second function always runs in less time than the first

Note that for small values of n , it may be reasonable to use an algorithm that is asymptotically more expensive, but we will consider these on a one-by-one basis



2.3.5

Weak ordering

In general, there are functions such that

- If $f(n) \sim g(n)$, then it is always possible to improve the performance of one function over the other by purchasing a faster computer
- If $f(n) < g(n)$, then you can never purchase a computer fast enough so that the second function always runs in less time than the first

Note that for small values of n , it may be reasonable to use an algorithm that is asymptotically more expensive, but we will consider these on a one-by-one basis



2.3.5

Landau Symbols

Recall Landau symbols from 1st year:

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

$$f(n) < c g(n)$$

whenever $n > N$

- The function $f(n)$ has a rate of growth no greater than that of $g(n)$



2.3.5

Landau Symbols

Before we begin, however, we will make some assumptions:

- 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, $f(n) > c$ for some value $c > 0$
 - That is, any problem requires at least one instruction and byte
 - They are increasing (monotonic increasing)



2.3.5

Landau Symbols

Another Landau symbol is Θ

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

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

whenever $n > N$

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



2.3.5

Landau Symbols

These definitions are often unnecessarily tedious

Note, however, that if $f(n)$ and $g(n)$ are polynomials of the same degree with positive leading coefficients:

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



2.3.5

Landau Symbols

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

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

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

That is,

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

$$g(n)(c - \varepsilon) < f(n) < g(n)(c + \varepsilon)$$



2.3.5

Landau Symbols

However, the statement
says that $f(n) = \Theta(g(n))$

$$g(n)(c - \varepsilon) < f(n) < g(n)(c + \varepsilon)$$

Note that this only goes one way:

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



2.3.6

Landau Symbols

We have a similar definition for **O**:

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

There are other possibilities we would like to describe:

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

- The function $f(n)$ has a rate of growth less than that of $g(n)$

We would also like to describe the opposite cases:

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



2.3.7

Landau Symbols

We will at times use five possible descriptions

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

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

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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

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

$$0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$$

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

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



2.3.7

Landau Symbols

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

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

and

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



2.3.7




Landau Symbols

Graphically, we can summarize these as follows:

We say $f(n) =$

$\mathbf{O}(g(n))$	$\mathbf{\Omega}(g(n))$
$\mathbf{o}(g(n))$	$\mathbf{\Theta}(g(n))$
$\mathbf{\omega}(g(n))$	

if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} =$

		
0	$0 < c < \infty$	∞



2.3.8

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
$2^n, e^n, 4^n, \dots$	exponential



2.3.8

Logarithms and Exponentials

Recall that all logarithms are scalar multiples of each other

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

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

- If $1 < a < b$, $\lim_{n \rightarrow \infty} \frac{a^n}{b^n} = \lim_{n \rightarrow \infty} \left(\frac{a}{b}\right)^n = 0$
- Therefore $a^n = o(b^n)$

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



2.3.8

Logarithms and Exponentials

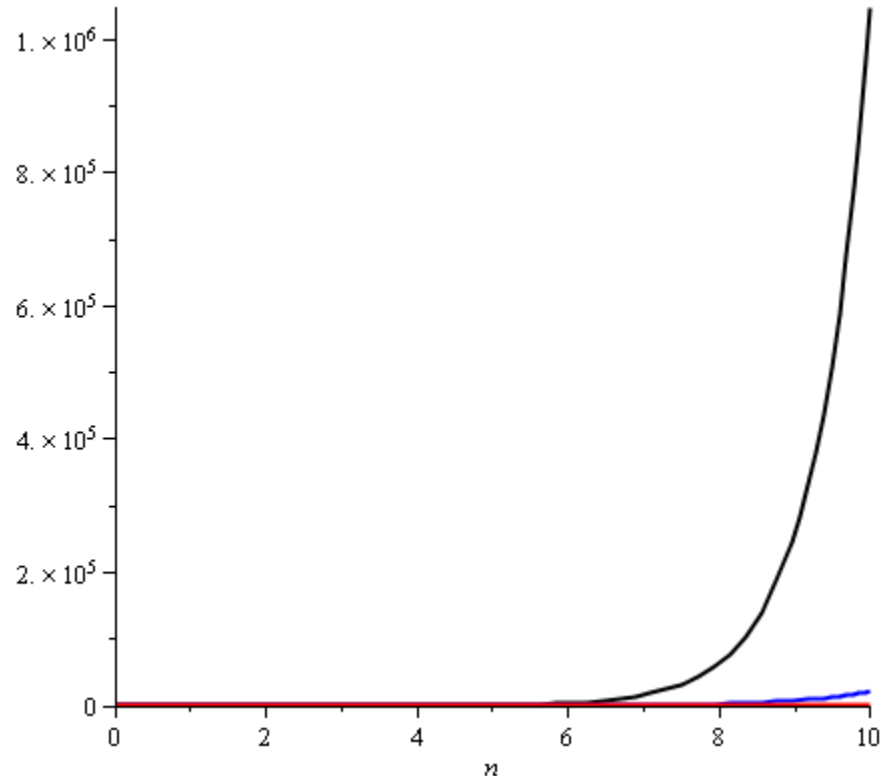
Plotting 2^n , e^n , and 4^n on the range $[1, 10]$ already shows how significantly different the functions grow

Note:

$$2^{10} = 1024$$

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

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





2.3.10

Algorithms Analysis

We will use Landau symbols to describe the complexity of algorithms

- E.g., adding a list of n doubles will be said to be a $\Theta(n)$ algorithm

An algorithm is said to have *polynomial time complexity* if its run-time may be described by $O(n^d)$ 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: $\Theta(n^2 2^n)$

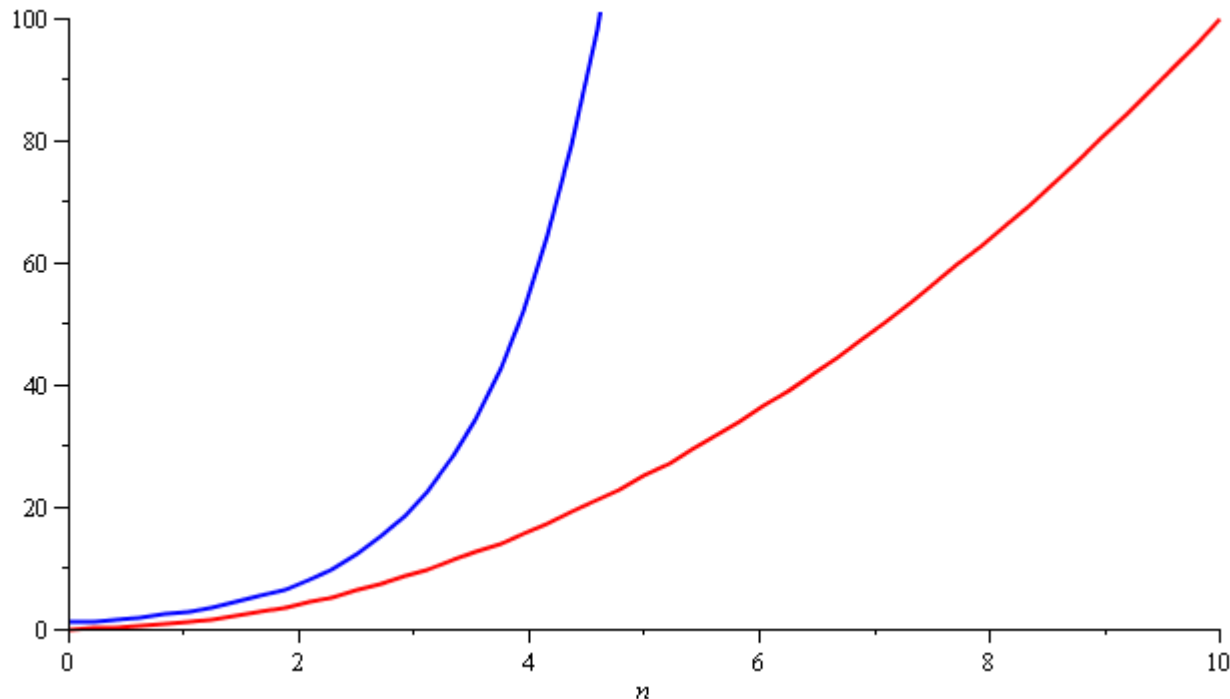


2.3.10

Algorithm Analysis

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

- Warning: don't call a **quadratic** curve “**exponential**”, either...please





Summary

In this class, we have:

- Reviewed Landau symbols, introducing some new ones: o O Θ Ω ω
- Discussed how to use these
- Looked at the equivalence relations



References

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

These slides are provided for the ECE 250 *Algorithms and Data Structures* course. The material in it reflects Douglas W. Harder's best judgment in light of the information available to him at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. Douglas W. Harder accepts no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.