Asymptotic Analysis

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*

Asymptotic Analysis

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

We will begin with one variable:

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

Examples with 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

Asymptotic Analysis

Given an algorithm:

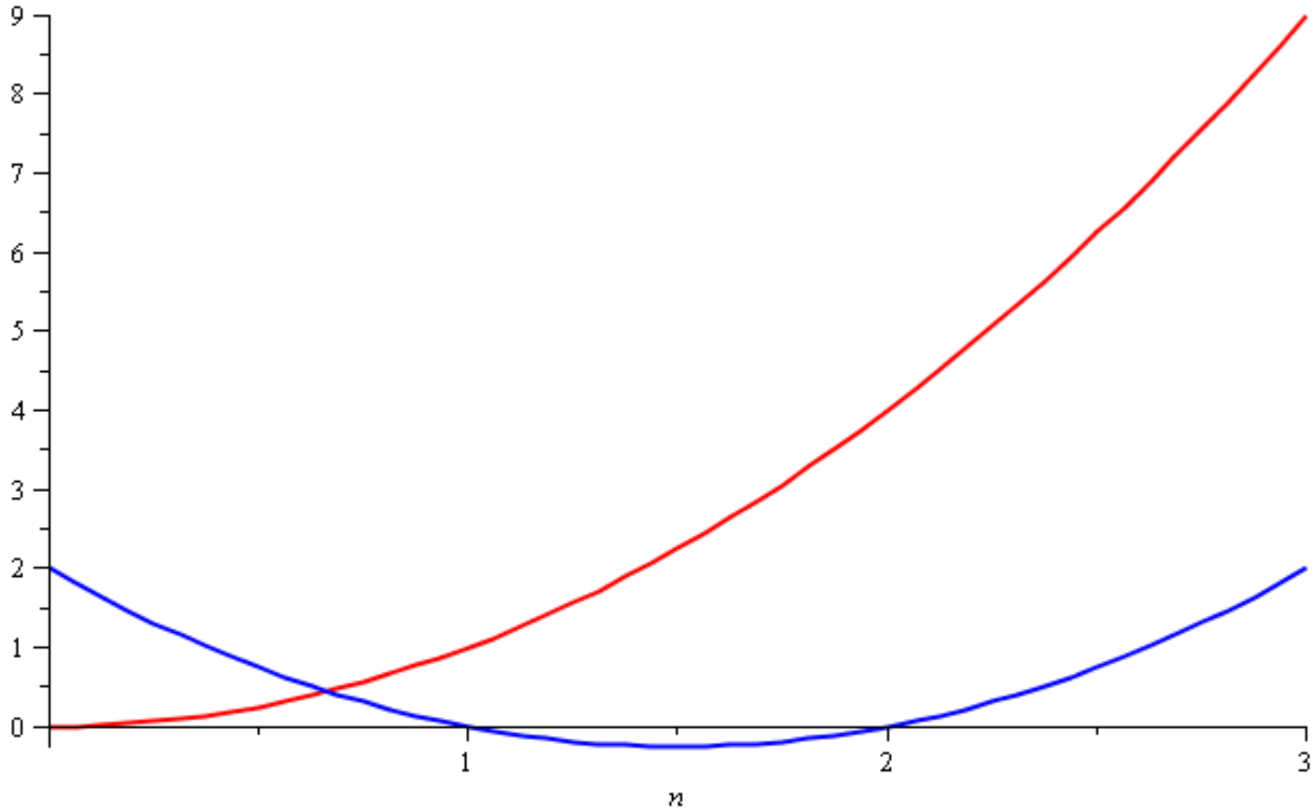
- We need to be able to describe these values mathematically
- We need a systematic means of using the description of the algorithm together with the properties of an associated data structure
- We need to do this in a machine-independent way

Quadratic Growth

Consider the two functions

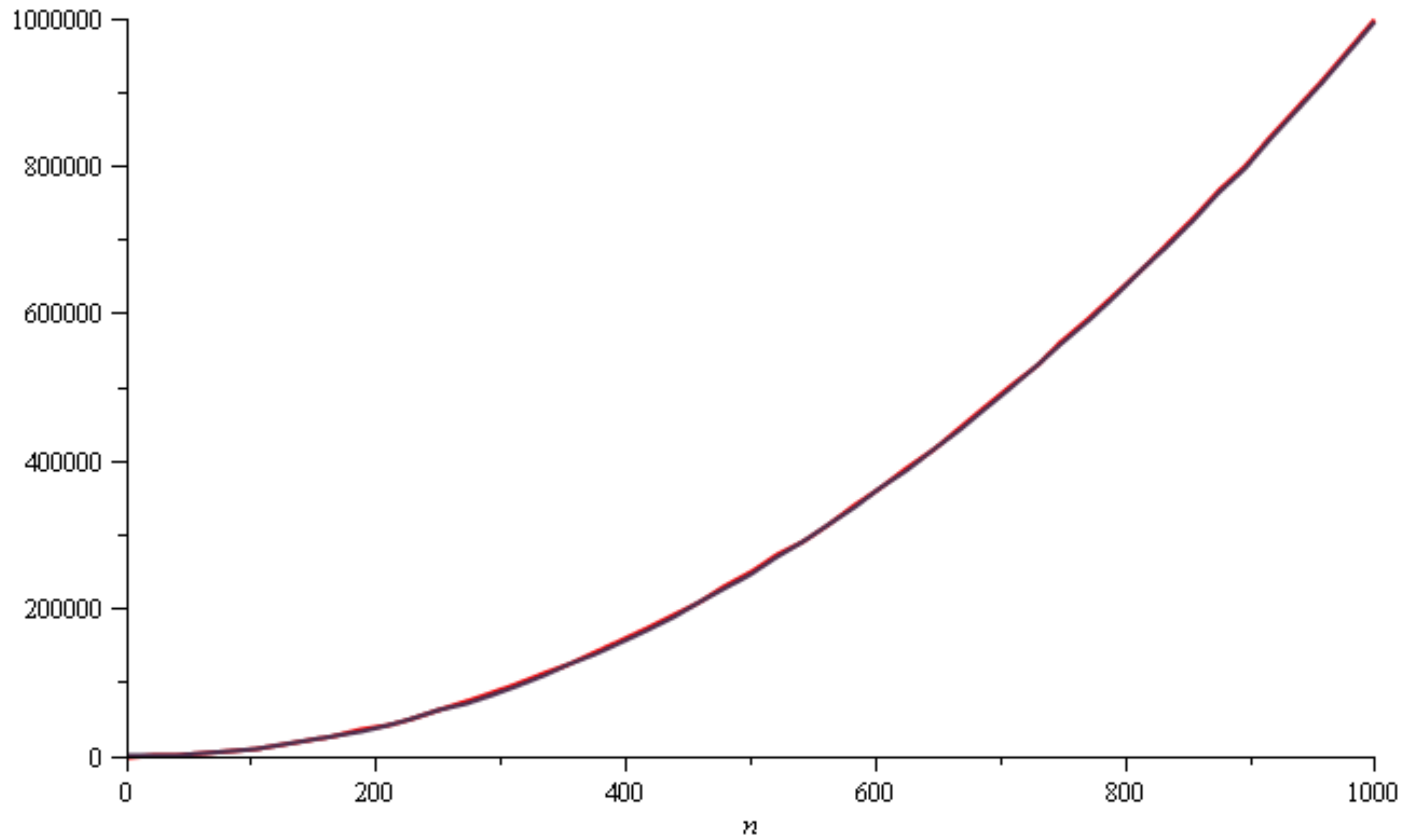
$f(n) = n^2$ and $g(n) = n^2 - 3n + 2$

Around $n = 0$, they look very different



Quadratic Growth

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



Quadratic Growth

The absolute difference is large, for example,

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

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

but the relative difference is very small

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

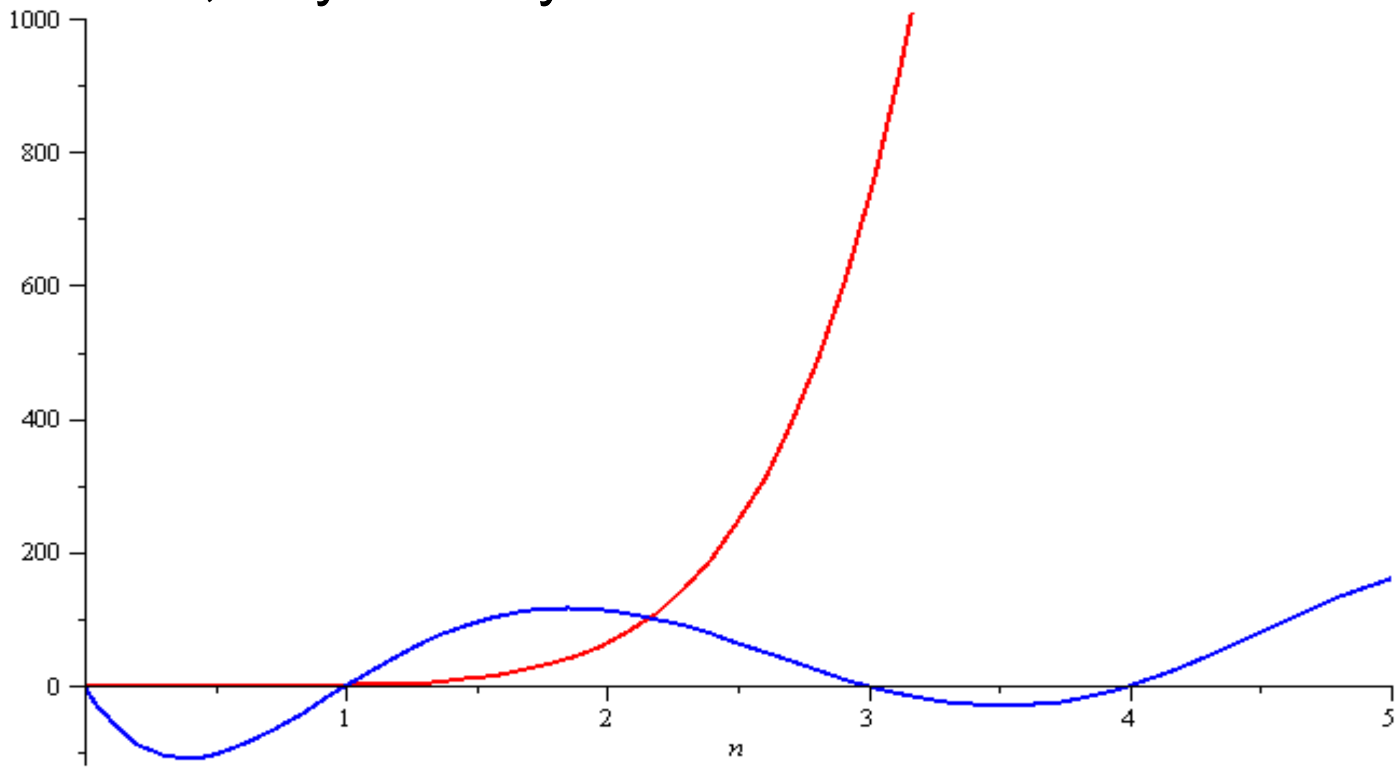
and this difference goes to zero as $n \rightarrow \infty$

Polynomial Growth

To demonstrate with another example,

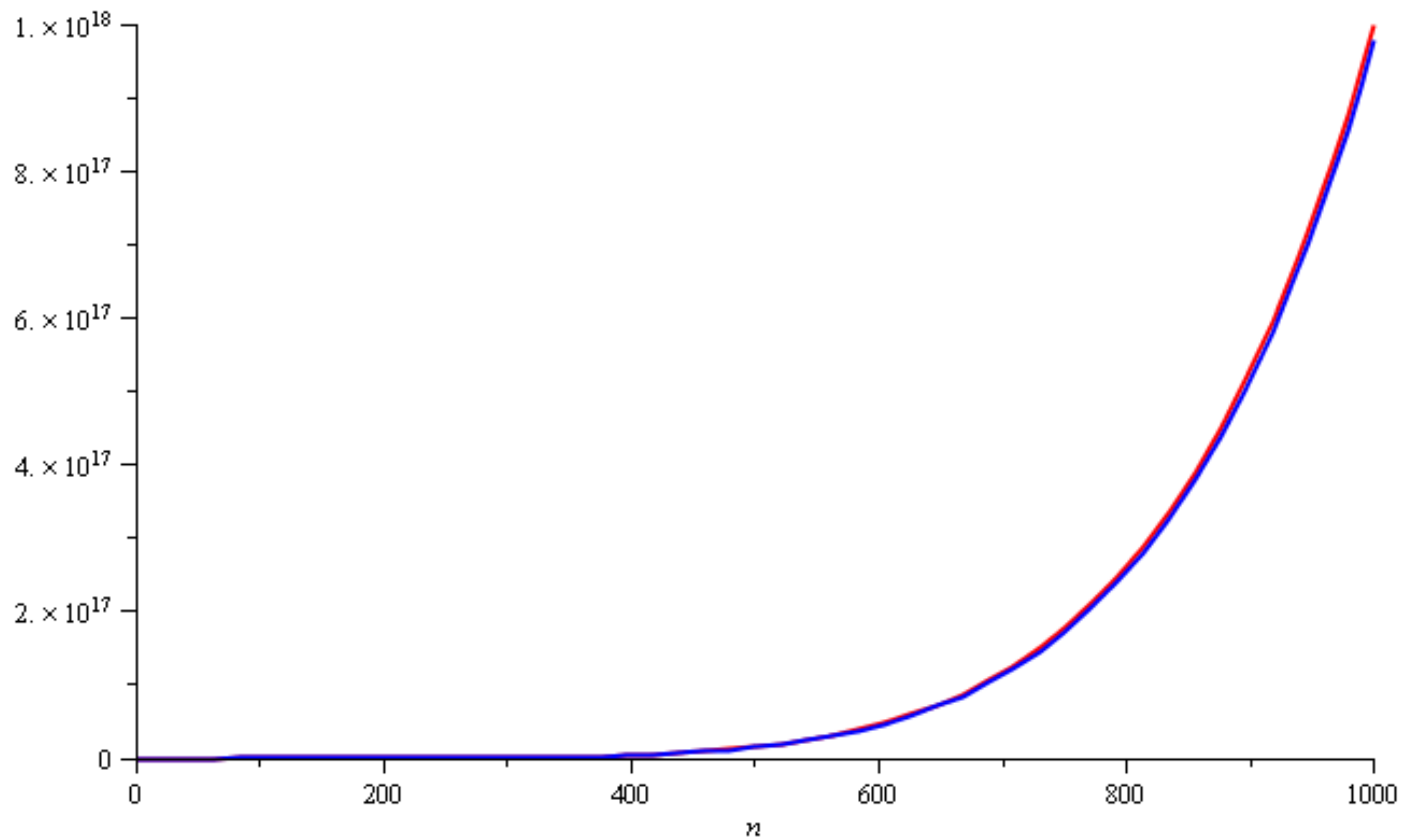
$f(n) = n^6$ and $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$

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 two examples:

– A comparison of selection sort and bubble sort

– A comparison of insertion sort and quicksort

Counting Instructions

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

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

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

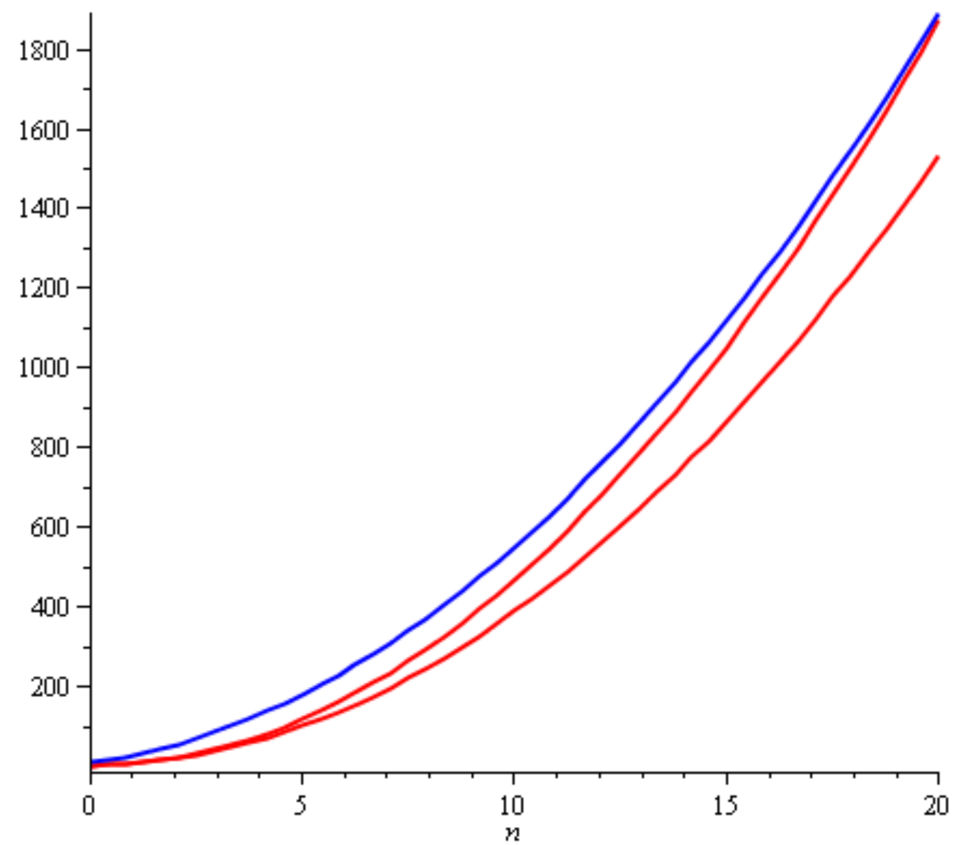
$$s(n) = 4n^2 + 14n + 12 \qquad \text{Selection sort}$$

The smaller the value, the fewer instructions are run

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

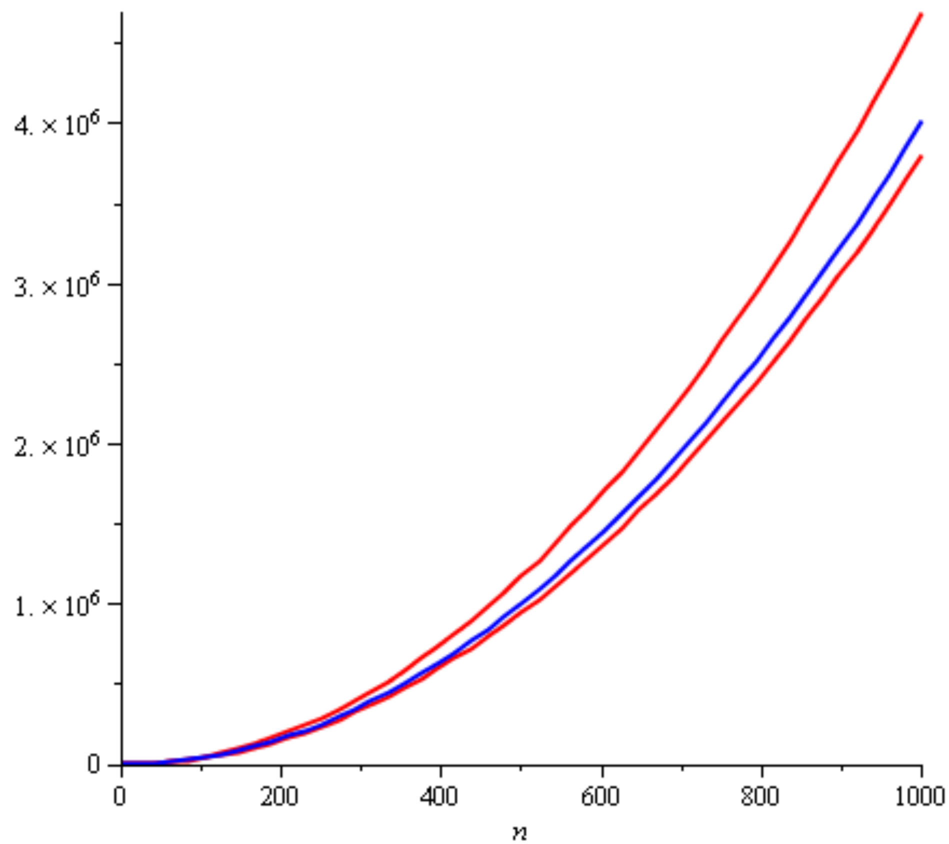
Counting Instructions

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



Counting Instructions

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



Counting Instructions

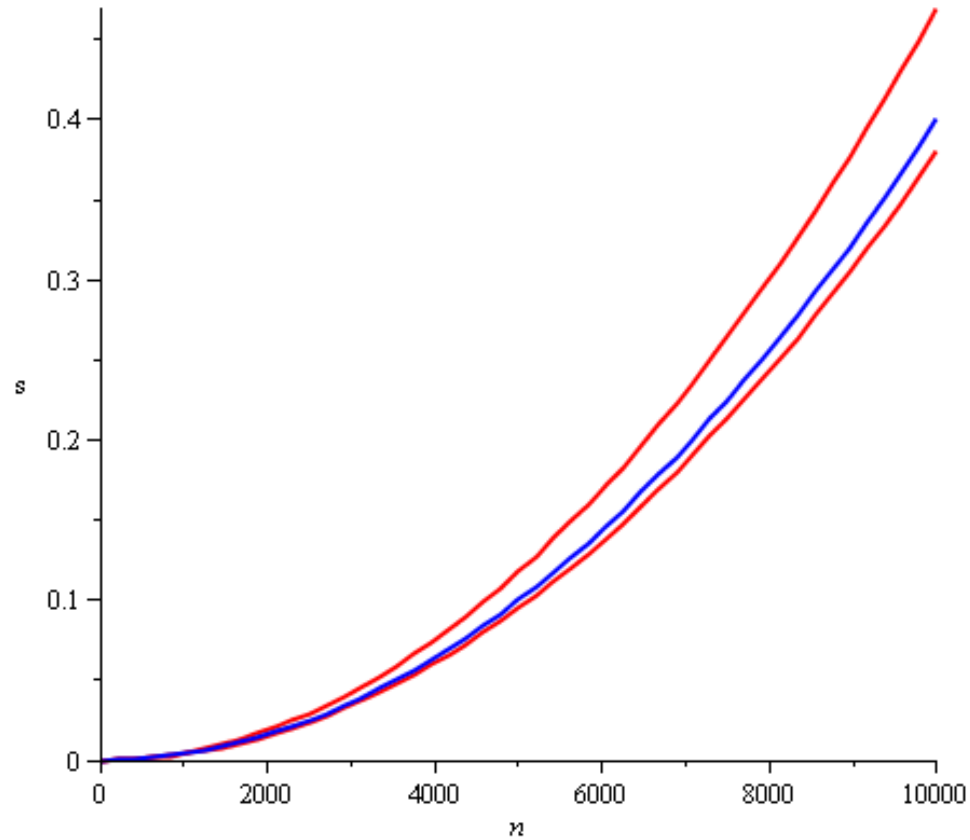
Is this a serious difference between these two algorithms?

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

Counting Instructions

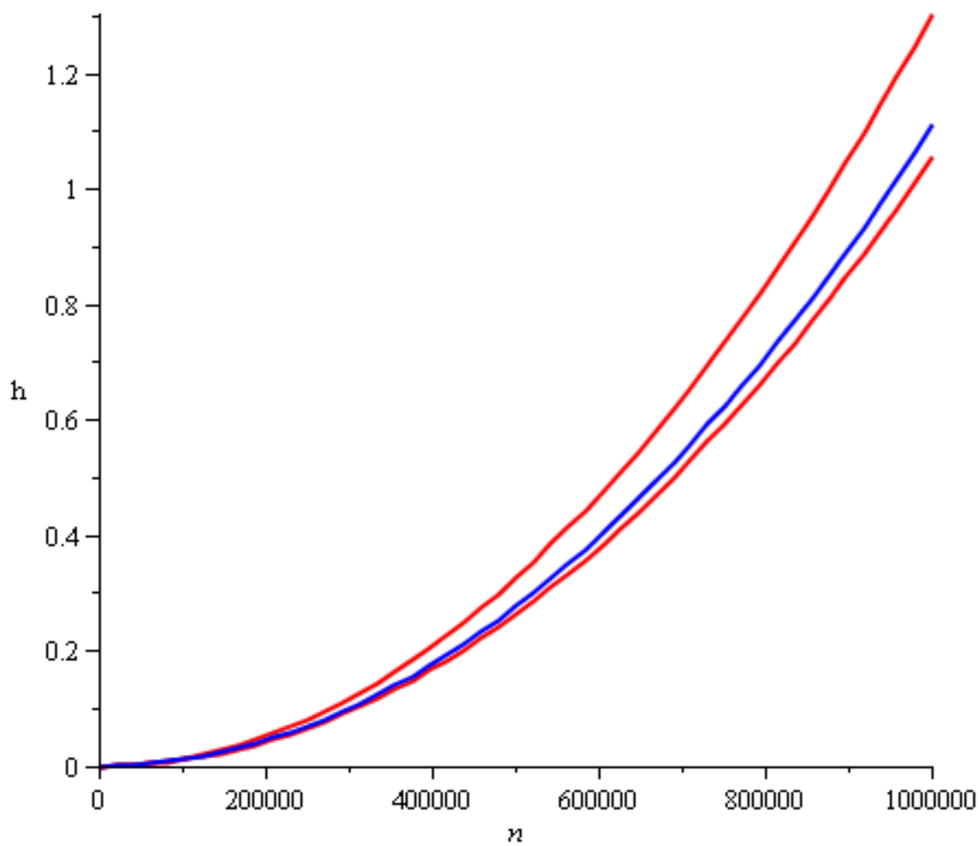
Suppose we have a 1 GHz computer

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



Counting Instructions

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

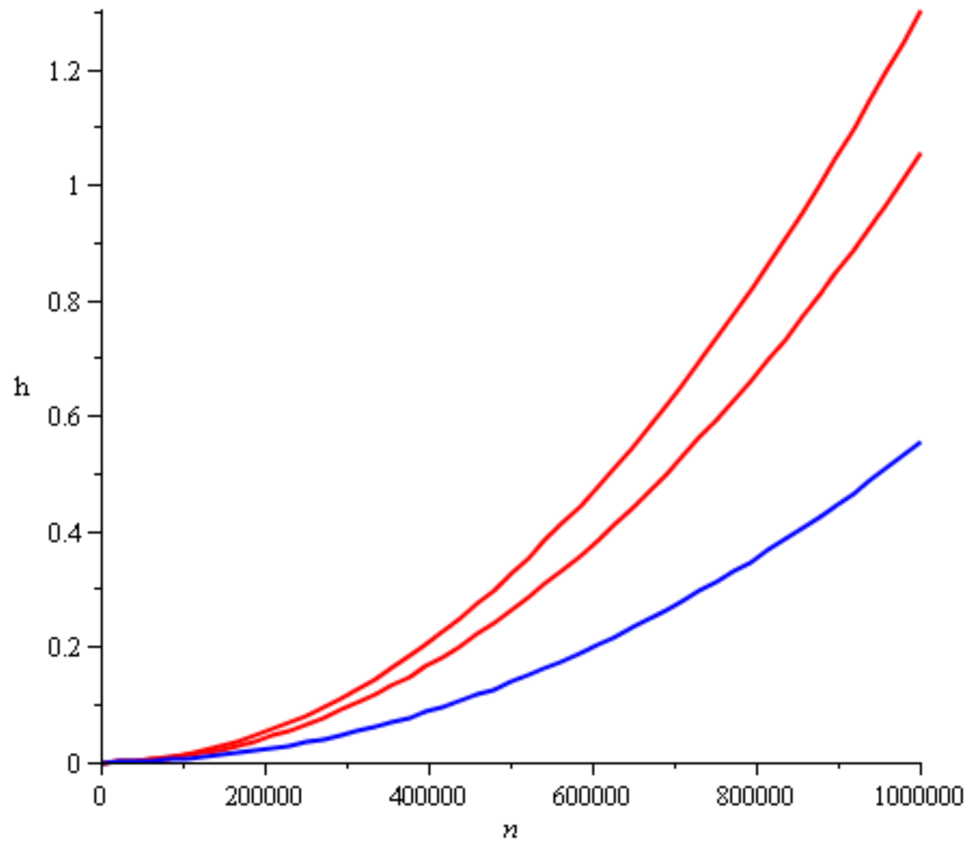


Bubble sort could, under some conditions, be 200 s faster

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



Counting Instructions

Justification?

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

$$\mathrm{f}(n) < M\mathrm{g}(n)$$

  where we choose

$$M = a_k/b_k + 1$$

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

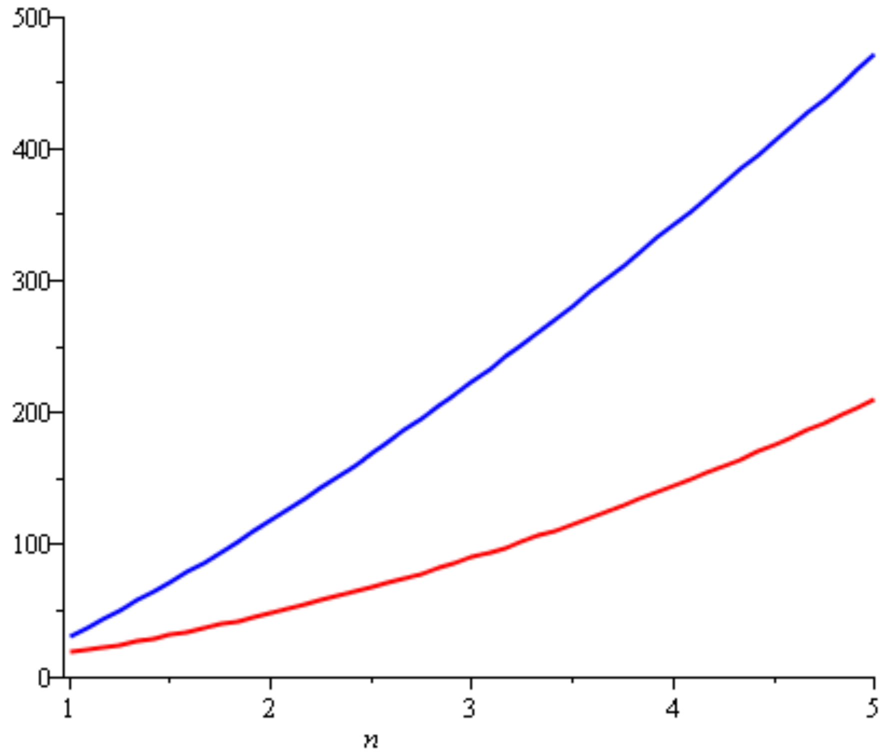
Question:

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

Counting Instructions

As another example:

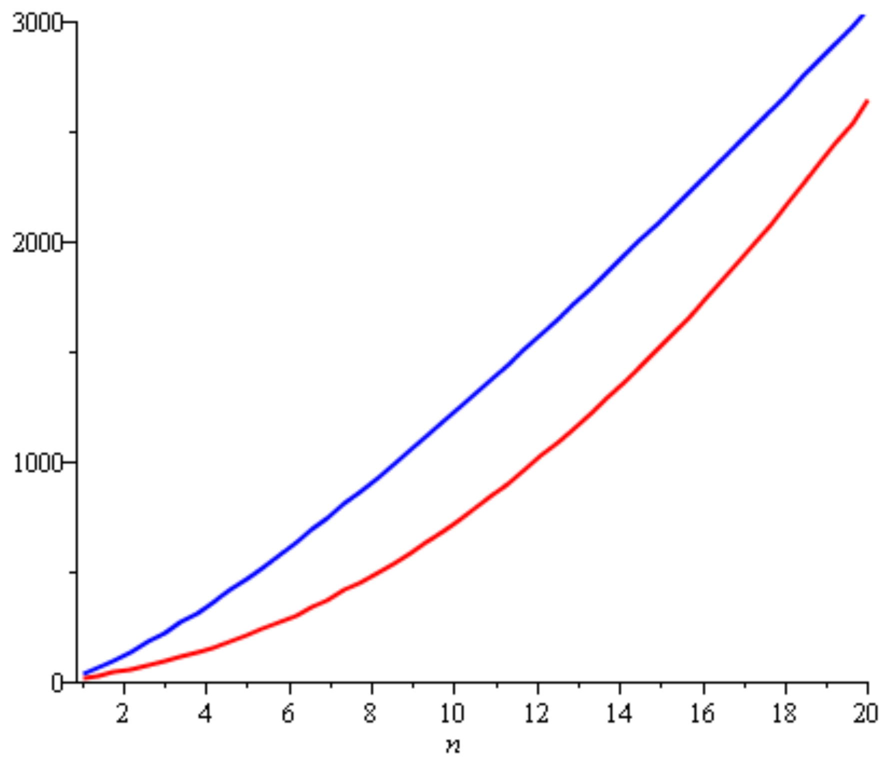
- Compare the number of instructions required for insertion sort and for quicksort
- Both functions are concave up, although one more than the other



Counting Instructions

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

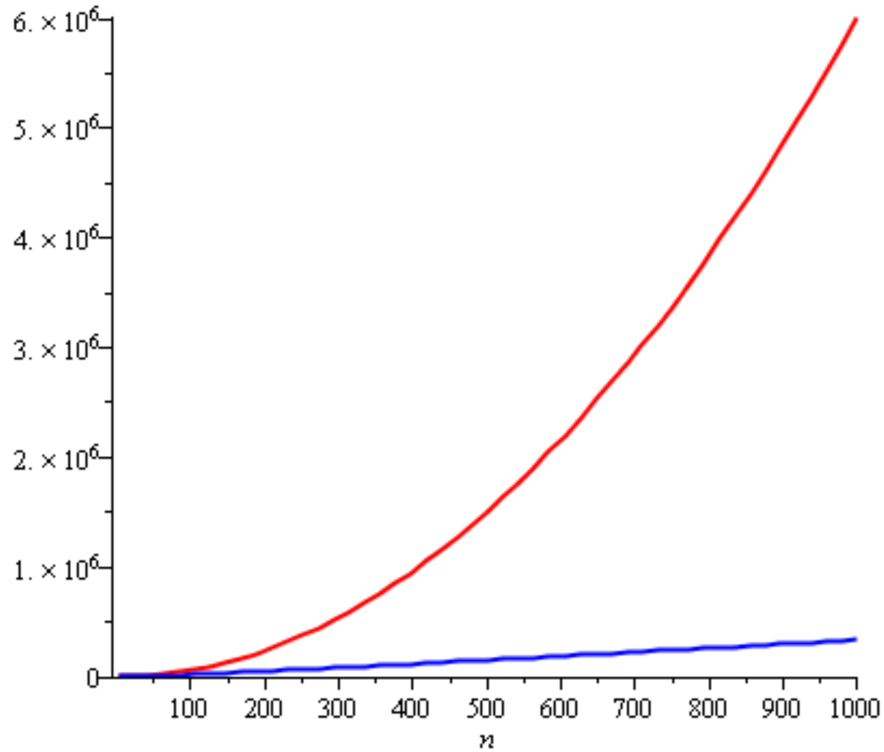
– Never-the-less, the graphic suggests it is more useful to use insertion sort when sorting small lists—quicksort has a large overhead



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?



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

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 <u>never</u> 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

Asymptotic Notation

A function $\mathrm{f}(n) = \mathbf{O}(\mathrm{g}(n))$ if there exists positive constants n_0 and c such that

$$0 \leq \mathrm{f}(n) \leq c\ \mathrm{g}(n)$$

whenever $n \geq n_0$

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

Asymptotic Notation

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

Asymptotic Notation

A function $\mathrm{f}(n) = \Theta(\mathrm{g}(n))$ if there exist positive constants $n0$, c_1, and c_2 such that

$$0 \leq c_1\,\mathrm{g}(n) \leq \mathrm{f}(n) \leq c_2\,\mathrm{g}(n)$$

whenever $n \geq n0$

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

Asymptotic Notation

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

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

Asymptotic Notation

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 $\mathrm{g}(n)$

Asymptotic Notation

We will at times use five possible descriptions

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

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

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

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

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

Asymptotic Notation

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

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

and

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

Asymptotic Notation

Graphically, we can summarize these as follows:

We say $\mathrm{f}(n) =$

$$\mathbf{O}(\mathrm{g}(n)) \qquad \Omega(\mathrm{g}(n))$$

$$\mathbf{o}(\mathrm{g}(n)) \qquad \Theta(\mathrm{g}(n)) \qquad \omega(\mathrm{g}(n))$$

if $\displaystyle \lim_{n \to \infty} \frac{\mathrm{f}(n)}{\mathrm{g}(n)} =$

$$0 \qquad 0 < c < \infty \qquad \infty$$

Asymptotic Notation

Some other observations we can make are:

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

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

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

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

We will select just one element to represent the entire class of these functions: n^2

- We could chose any function, but this is the simplest

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

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

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

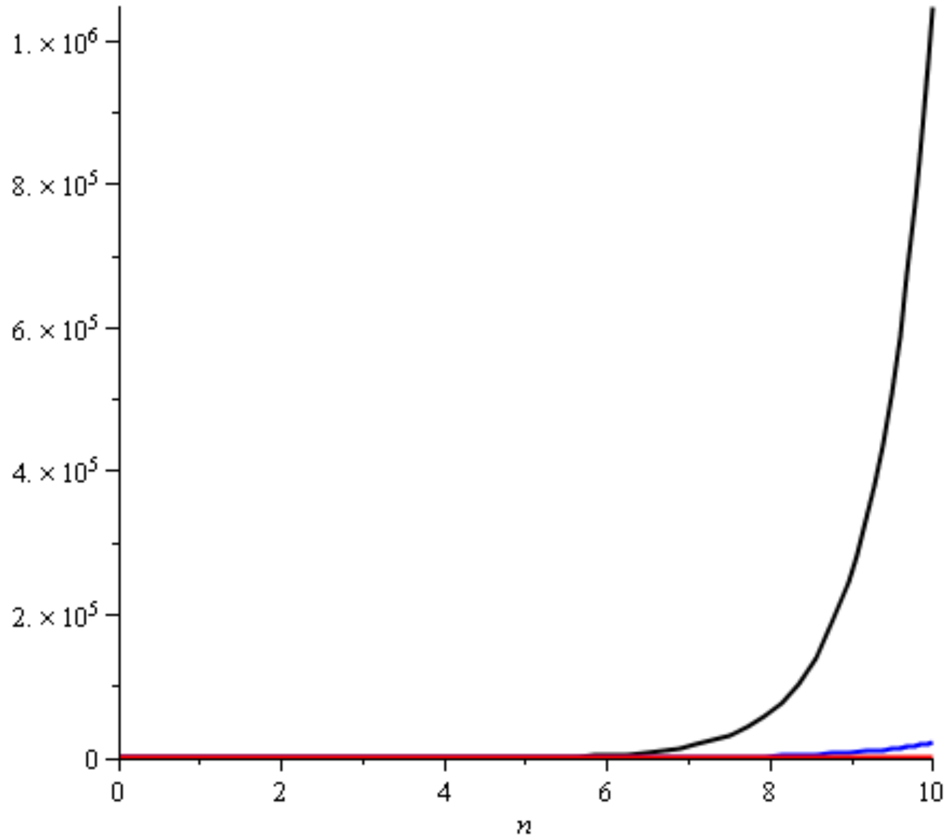
However, we will see that it is almost universally undesirable to have an exponentially growing function!

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:

$$2^{10} = 1024$$
$$e^{10} \approx 22\,026$$
$$4^{10} = 1\,048\,576$$



Algorithms Analysis

We will use Asymptotic Notation 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: $\Theta(n^2 2^n)$

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