CS 2420: Asymptotic Analysis

Dr. Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Outline

- In this topic, we will look at:
  - Justification for analysis
  - Quadratic and polynomial growth
  - Counting machine instructions
  - Big-Θ as the equivalent relation

Motivation

- 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
    - *Asymptotic analysis*
    - *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

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;
}
```

Maximum Value

One comment:

- In this class, we will look at both simple C++ arrays and the standard template library (STL) structures
- Instead of using the built-in array, we could use the STL **`vector`** class
- The **`vector`** class is closer to the C#/Java array

Maximum Value

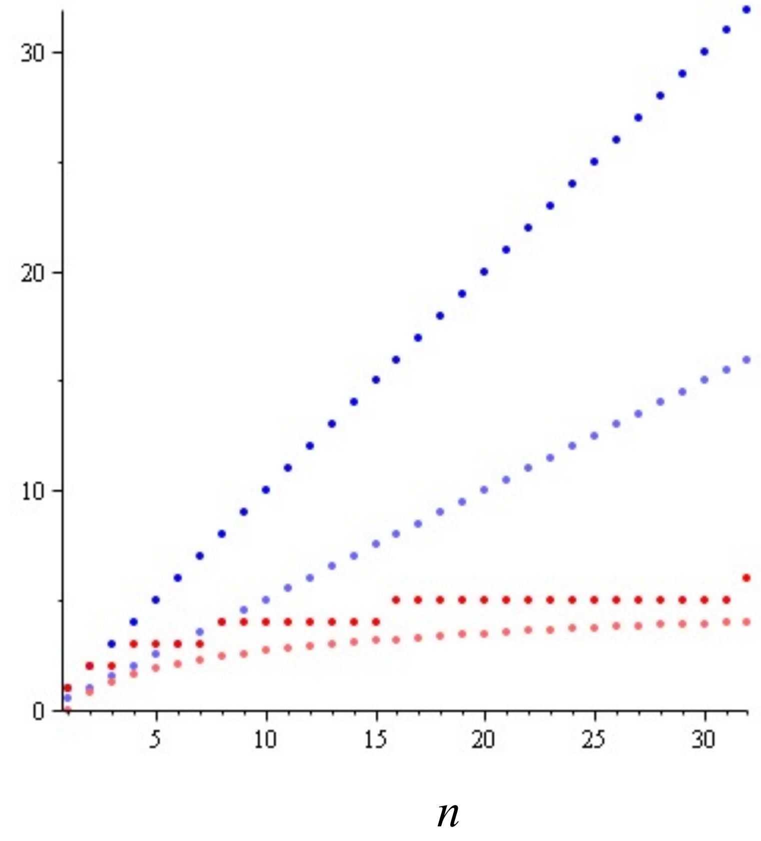
```
#include <vector>
int find_max( const std::vector<int>& array ) {
    if ( array.size() == 0 ) {
        throw underflow();
    }
    int max = array[0];
    for ( int i = 1; i < array.size(); ++i ) {
        if ( array[i] > max ) {
            max = array[i];
        }
    }
    return max;
}
```

Linear and binary search

There are other algorithms which 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



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

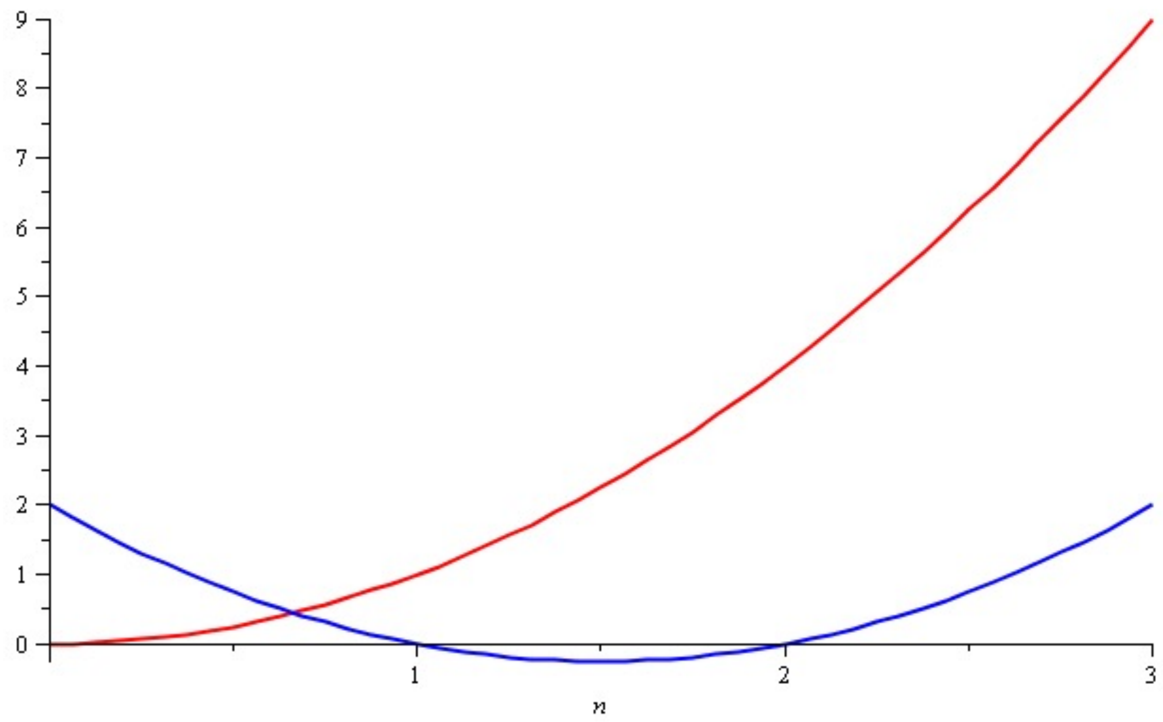
For this, we need asymptotic analysis

Quadratic Growth

Consider the two functions

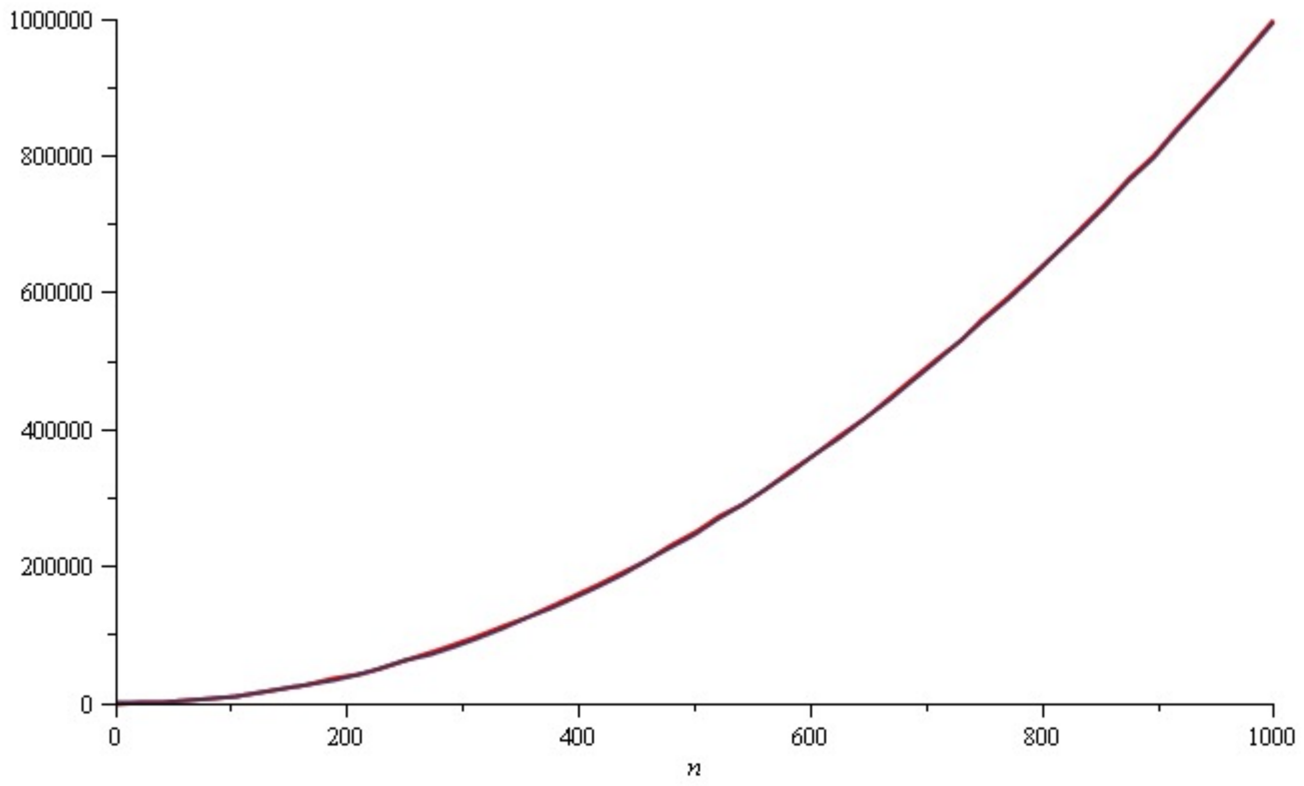
$f(n) = n^2$ and $g(n) = n^2 - 3n + 2$

Around $n = 0$, they look very different



Quadratic Growth

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

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

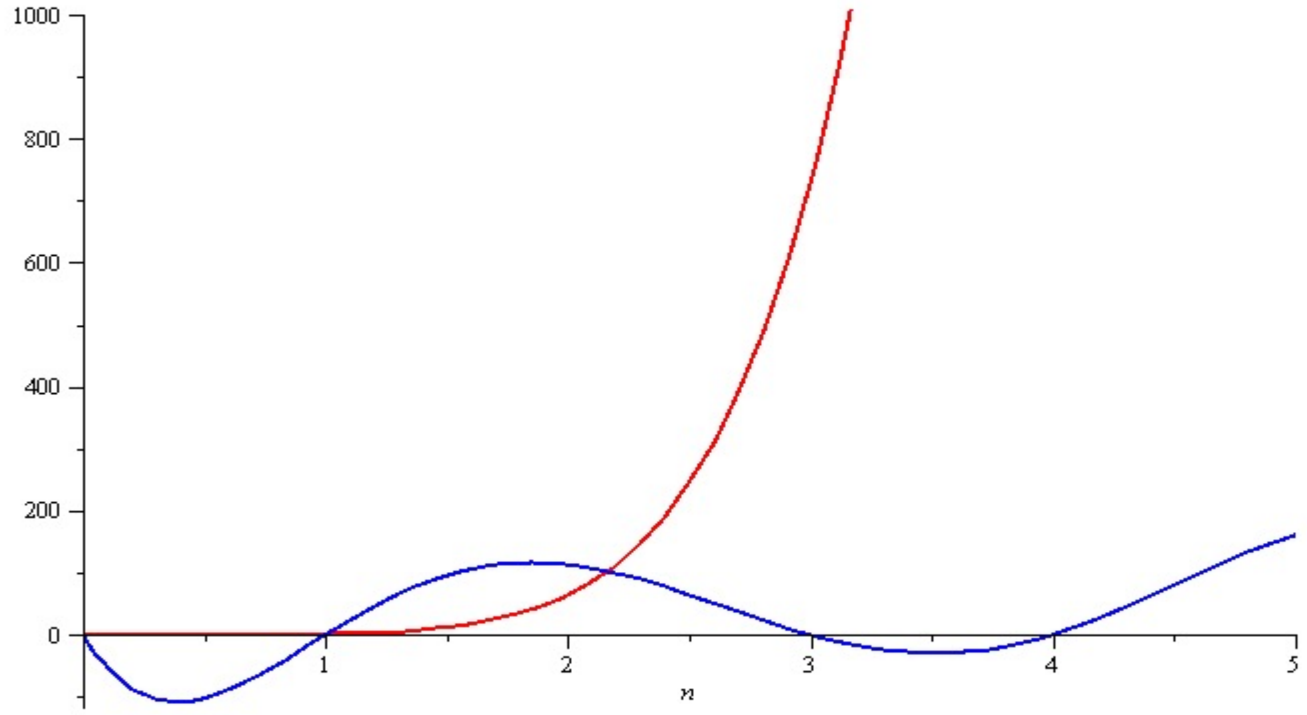
and this difference goes to zero as $n \to \infty$

Polynomial Growth

To demonstrate with another example,

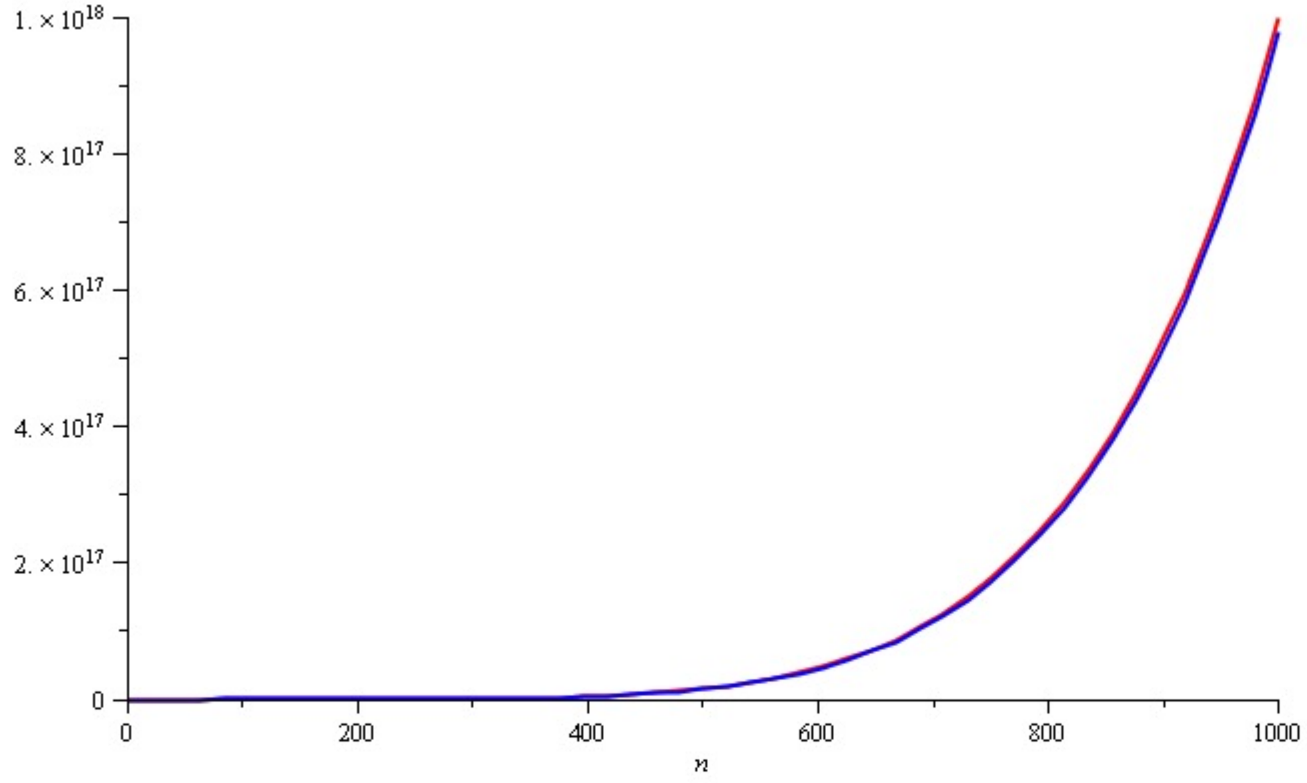
$f(n) = n^6$ and $g(n) = n^6 - 23n^5 + 193n^4 - 729n^3 + 1206n^2 - 648n$

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 example, n^6 in the second example

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

- Find a power of a number
  - Input: a, b ($1 < a, b < 2147483647$)
  - Output: $x = a^b$
    - $a=3, b=4, x=3^4=81$
    - $a=2, b=5, x=2^5=32$
  - Assume you can only do multiplication one at a time
- Naïve method
  - $2^{16} = 2*2*2*2*2*2*\ldots*2$ total 15 calculations
- Can we do better?

Examples

- Naïve method
  - $2^{16} = 2*2*2*2*2*2*\ldots*2$ total 15 calculations
- A better way as follows:

$2^{16} = 2^8 * 2^8$

$2^8 = 2^4 * 2^4$

$2^4 = 2^2 * 2^2$

$2^2 = 2 * 2$

Examples

- Naïve method
  - # calculations: linear to b

- Divide and Conquer
  - # calculations: $\log_2(b)$

- Let's say n = 2147483648
  - Naïve method takes **2147483647** calculations (~10-30s)
  - Divide and Conquer takes only **31** calculations (~1us)
    - 10000000x faster!
  - Indeed, this is a Goo____ interview question

Sorting Example

- Let’s look at two famous sorting algorithms
  - Bubble sort
  - Selection sort

Counting Instructions

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

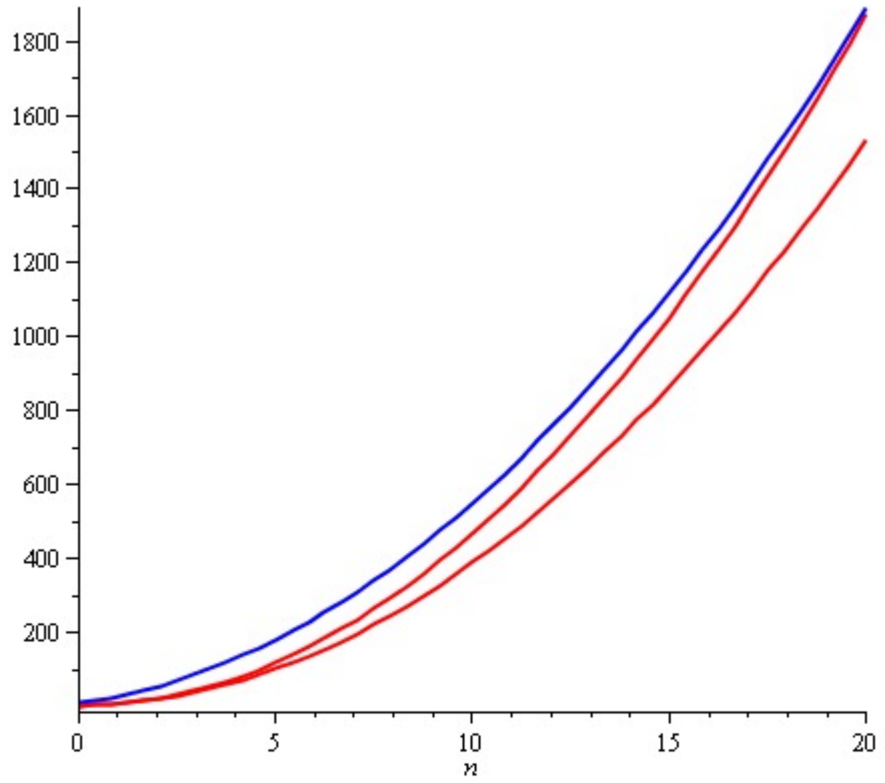
$b_{\text{worst}}(n)$	$= 4.7n^2 - 0.5n + 5$	Bubble sort
$b_{\text{best}}(n)$	$= 3.8n^2 + 0.5n + 5$	
$s(n)$	$= 4n^2 + 14n + 12$	Selection sort

The smaller the value, the fewer instructions are run

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

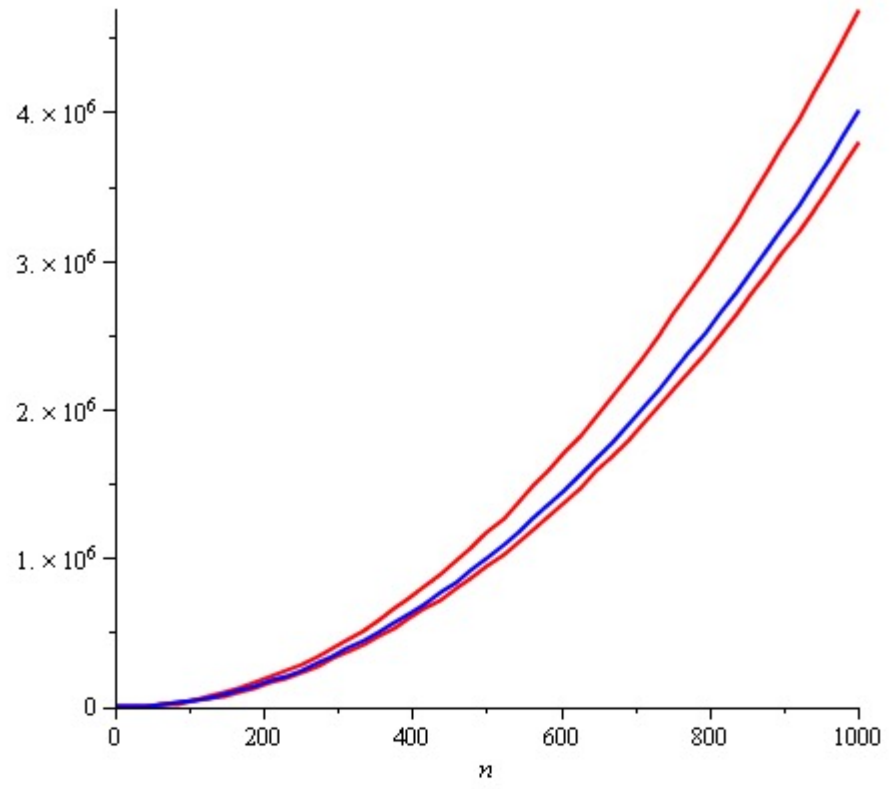
Counting Instructions

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

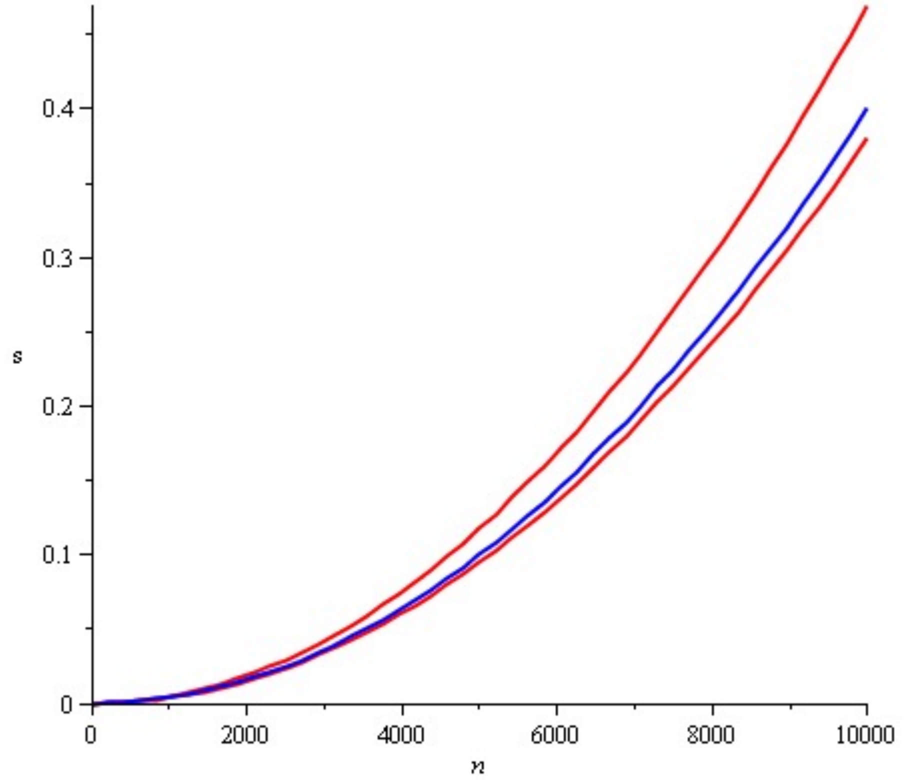
Is this a serious difference between these two algorithms?

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

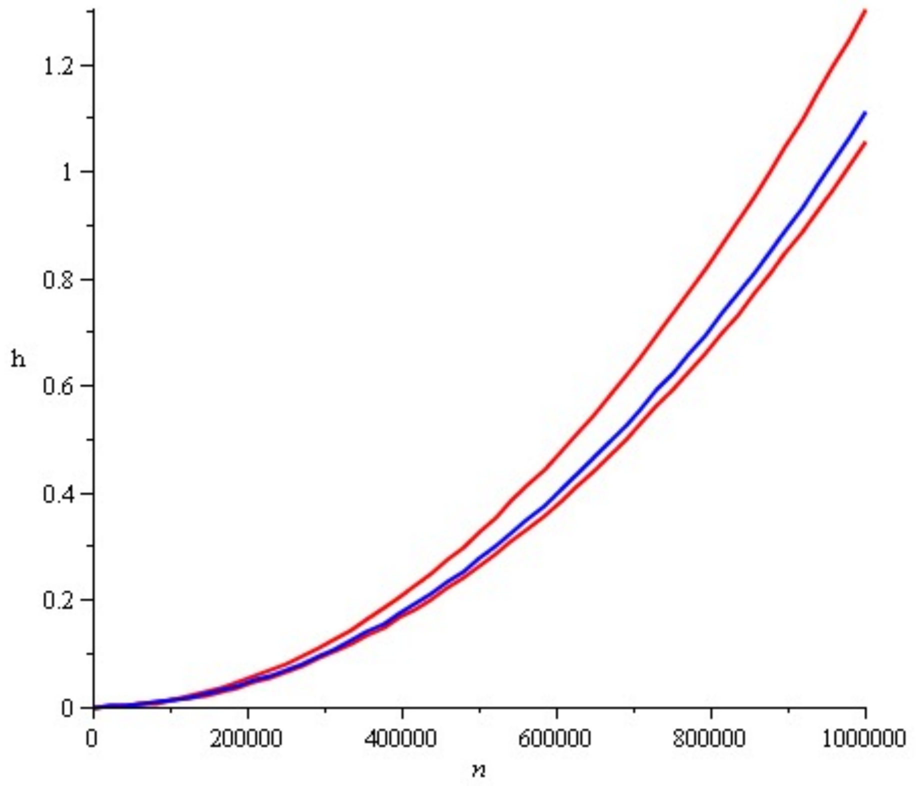
Suppose we have a 1 GHz computer

- 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

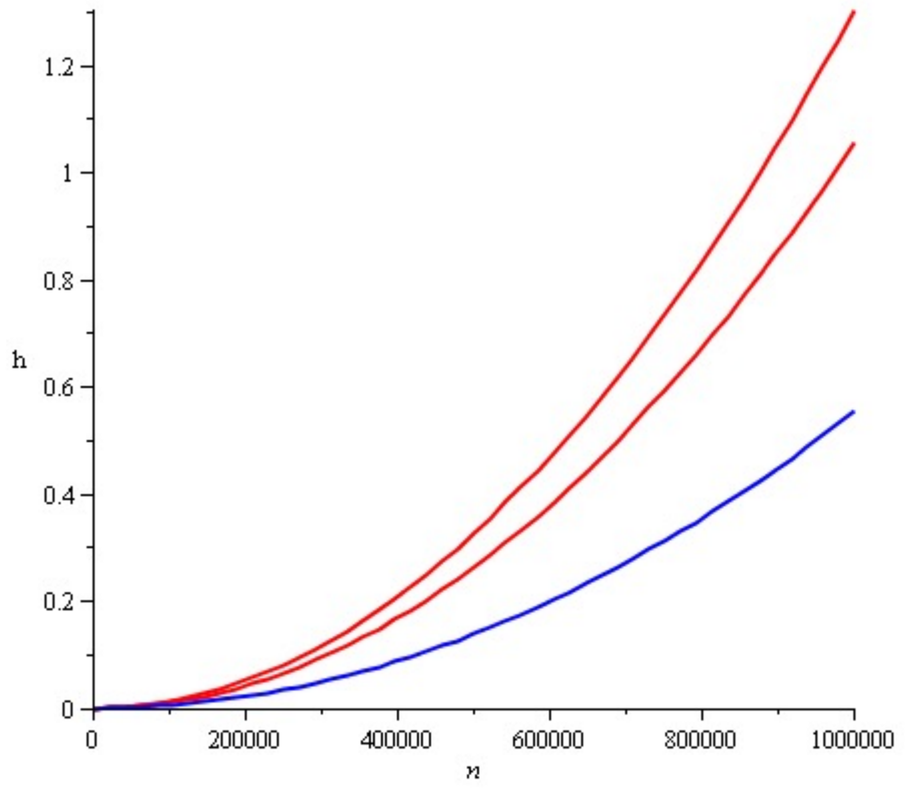


Bubble sort could, under some conditions, be 200 s faster

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)

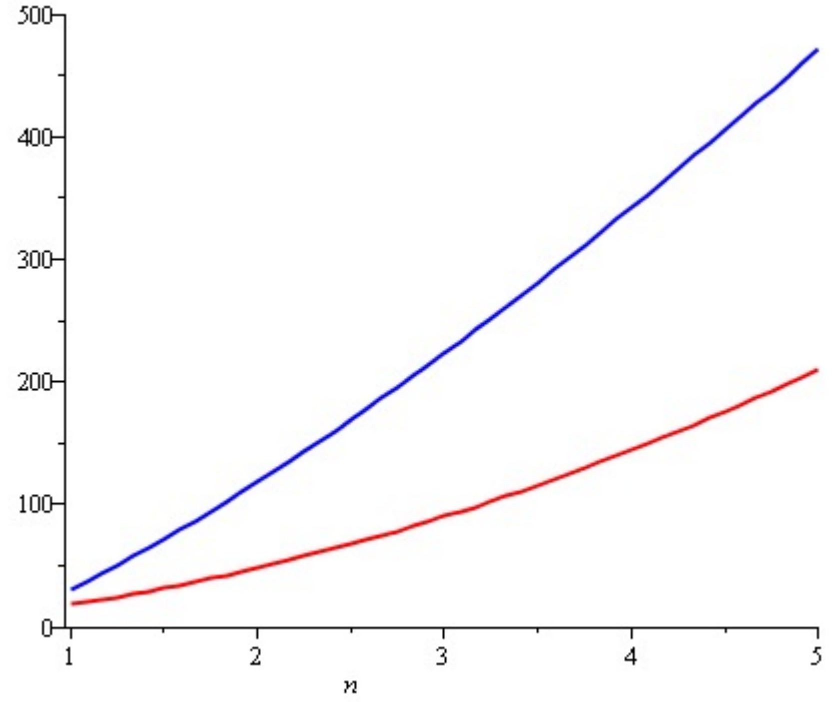
Question:

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

Counting Instructions

As another example:

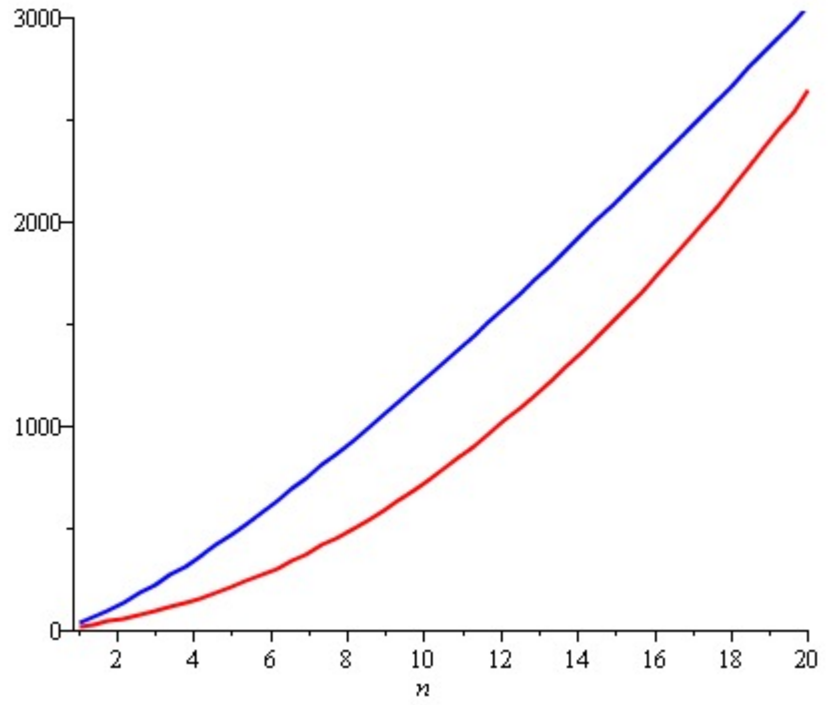
- Compare the number of instructions required for insertion sort and for quicksort
- Both functions are concave up, although one more than the other



Counting Instructions

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

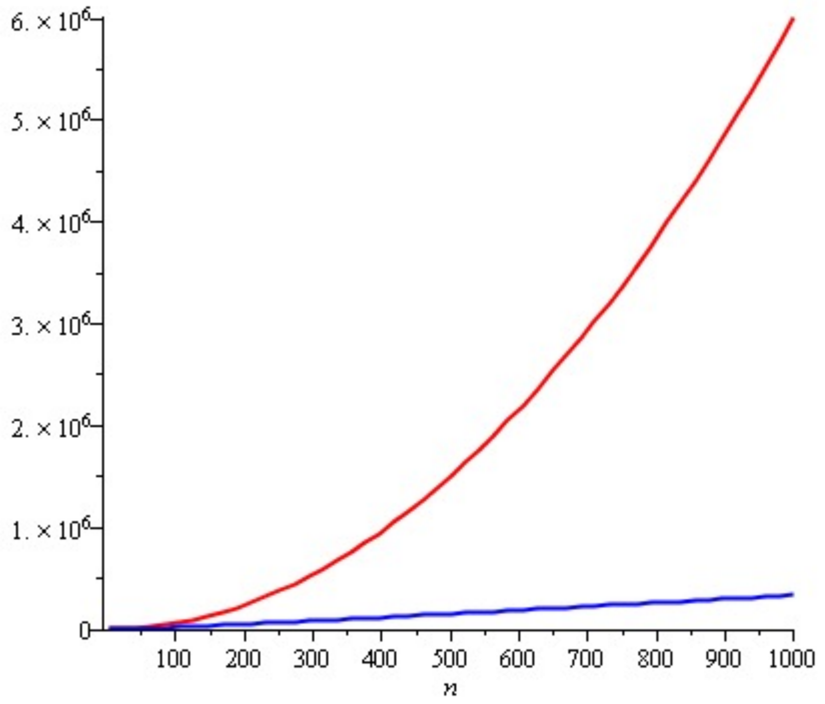
- Never-the-less, the graphic suggests it is more useful to use insertion sort when sorting small lists—quicksort has a large overhead



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?



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

Recall that with the equivalence class of all 19-year olds, we only had to pick one such student?

Similarly, 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 $a^n = \mathbf{o}(b^n)$

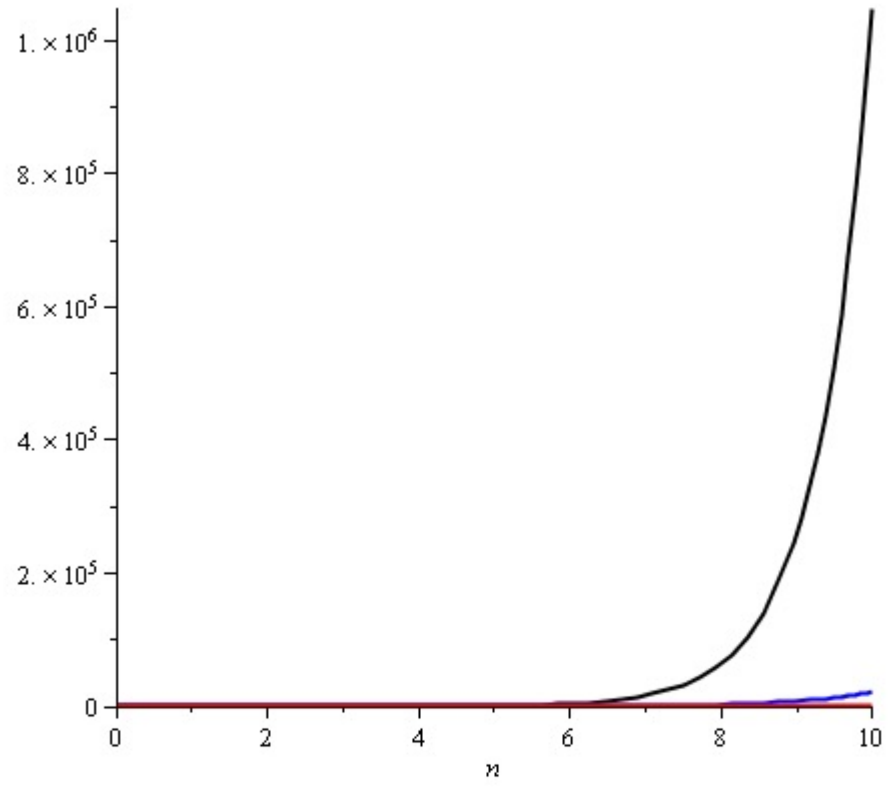
However, we will see that it is almost universally undesirable to have an exponentially growing function!

Logarithms and Exponentials

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

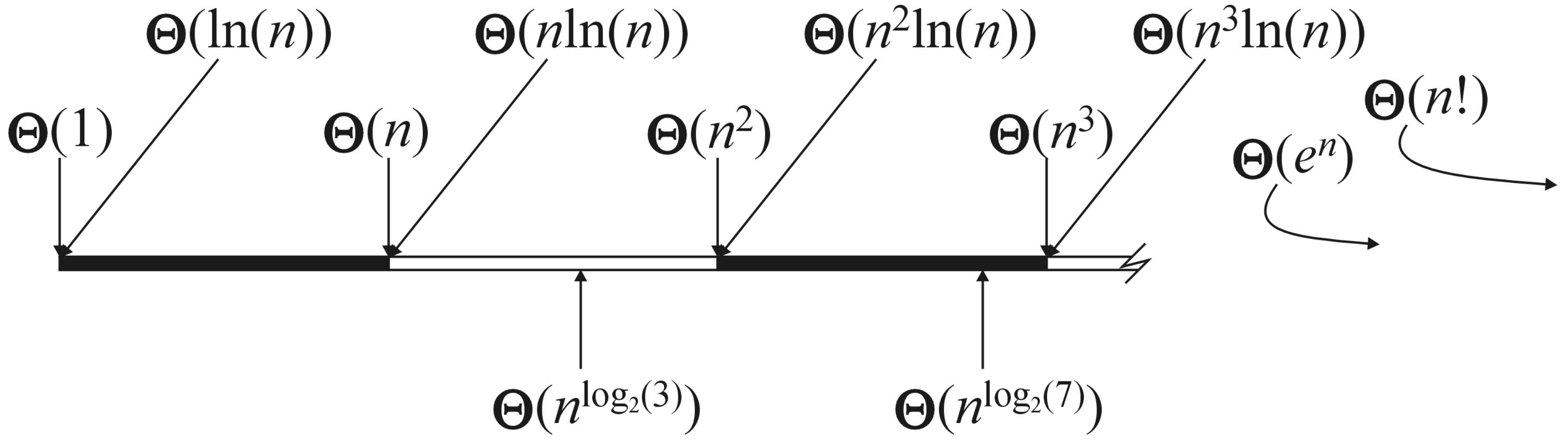
Note:

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



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

- 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 $\mathrm{O}(n^d)$ for some fixed $d \geq 0$

- We will consider such algorithms to be *efficient*

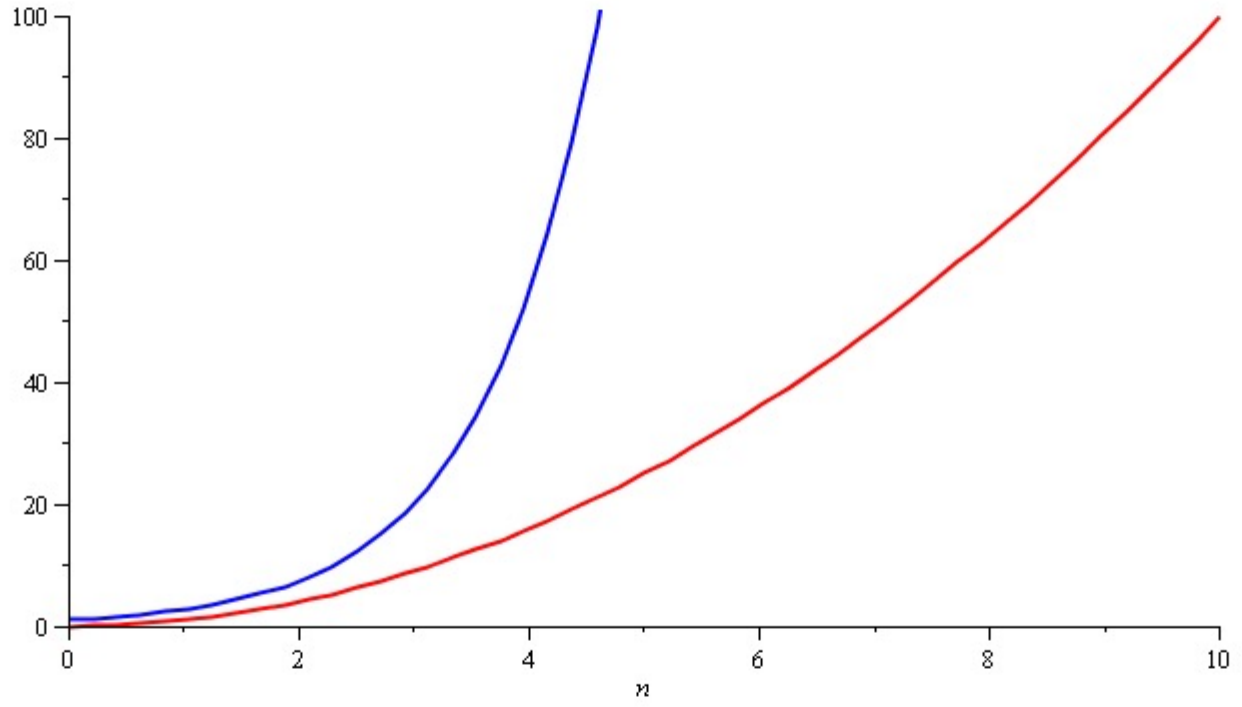
Problems that have no known polynomial-time algorithms are said to be *intractable*

- 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



Summary

In this class, we have:

- Introducing some new notations: o O Θ Ω ω
- Discussed how to use these
- Looked at the equivalence relations

LAB 1

- Q1: What is the relative error between $n^2 + 2n + 5$ and an approximation n^2 when $n = 1000$ and when $n = 1000000$? The relative error is the difference between the actual value and the approximation over the actual value.
- Q2: Find the most appropriate representative element that describes each of the following rates of growth. For example, the most appropriate representative of $3n^2 + 4n \ln(n) + 5n + 2$ is n^2
  - $5n^2 + 4n + 3n^{\lg(6)} + 4 + \ln(n)$
  - $6n + 7 \ln(n) + 8n \ln(n) + 9$
  - $4n^3 + 7n + 514n^4 + 35n^2 + 2n^6 + 5624$
  - $10n + 11 \ln(n) + 1 + 2n^2 + 4n^2$

Email your solution to tsung-wei.huang@utah.edu by 23:59 PM 9/22