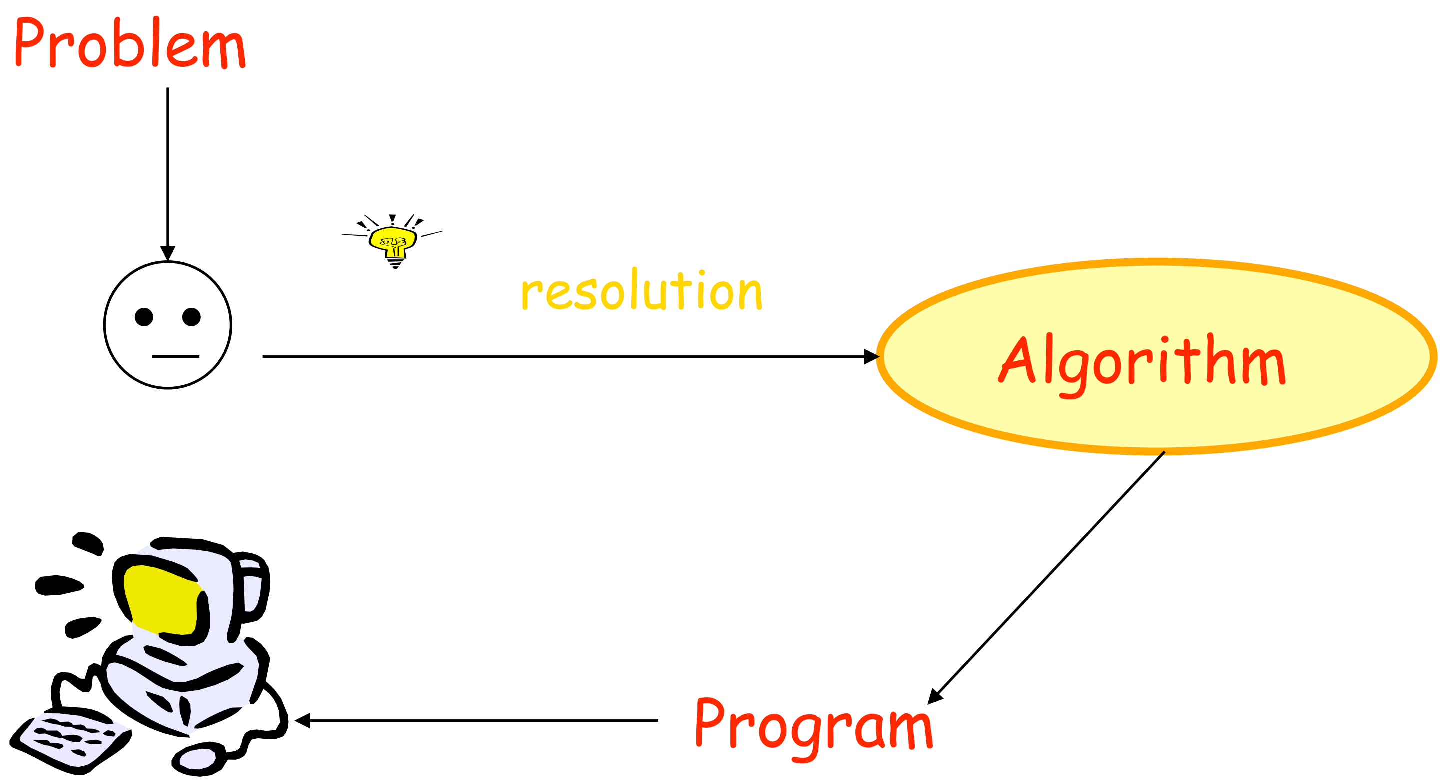
CSI 2101- Growth of Functions



CSI 2101- Complexity

Having an algorithm for a given problem that does not mean that the problem can be solved.

The procedure (algorithm) may be so inefficient that it would not be possible to solve the problem within a useful period of time.

So what is inefficient?

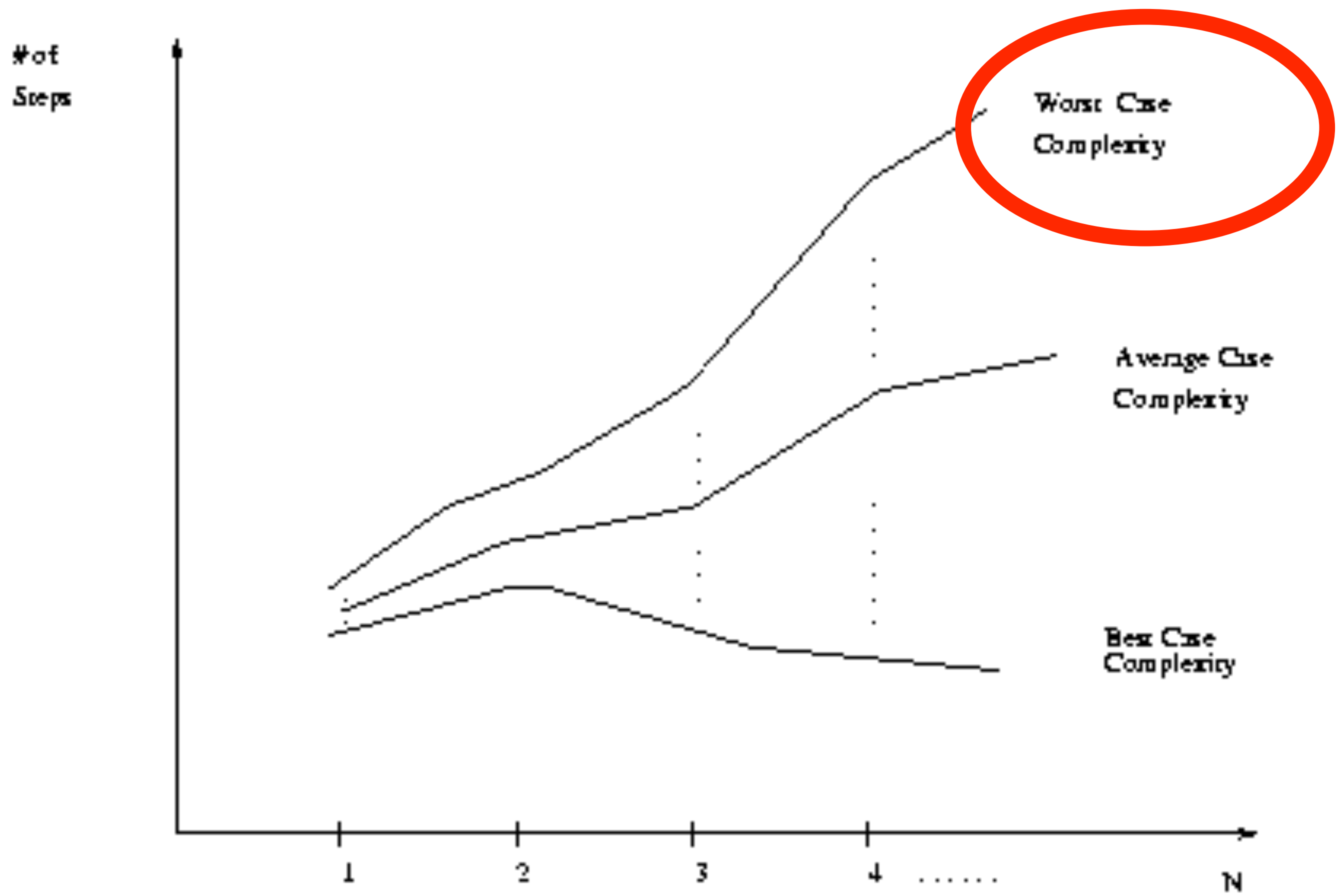
What is the "complexity" of an algorithm?

number of steps that it takes to transform the input data into the desired output.

Each simple operation (+,-,*,/,=,if, etc) and each memory access corresponds to a step. In general this depends of the problem.

The complexity of an algorithm is a function of the size of the input (or size of the instance). We'll denote the complexity of algorithm A by $C_A(n)$, where n is the size of the input.

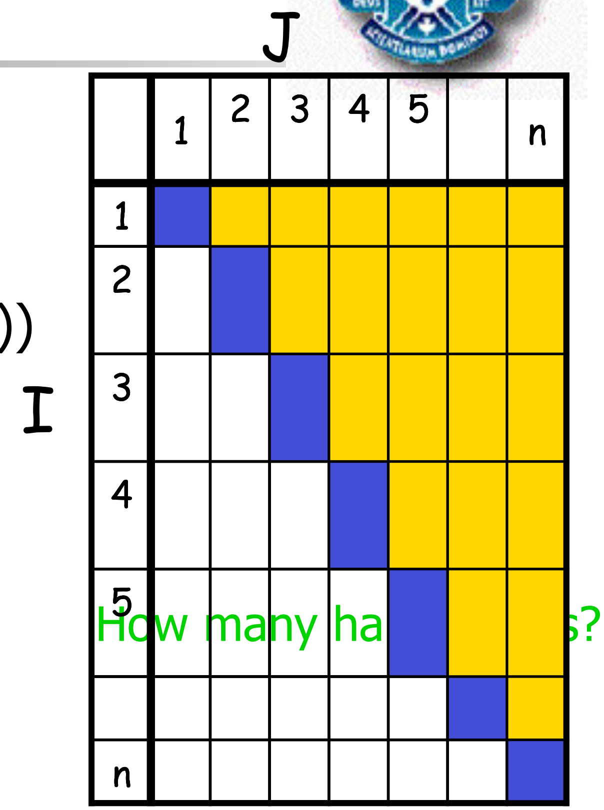
Different notions of complexity



In general this is the notion that we use to characterize the complexity of algorithms

Complexity

- Algorithm "Good Morning"
- For I = 1 to n
-     For J = I+1 to n
-         ShakeHands(student(I), student(J))



Running time of "Good Morning":

Time = (# of HS) x (time/HS) + overhead

Want an expression for T(n), running time of "Good Morning" on input of size n.

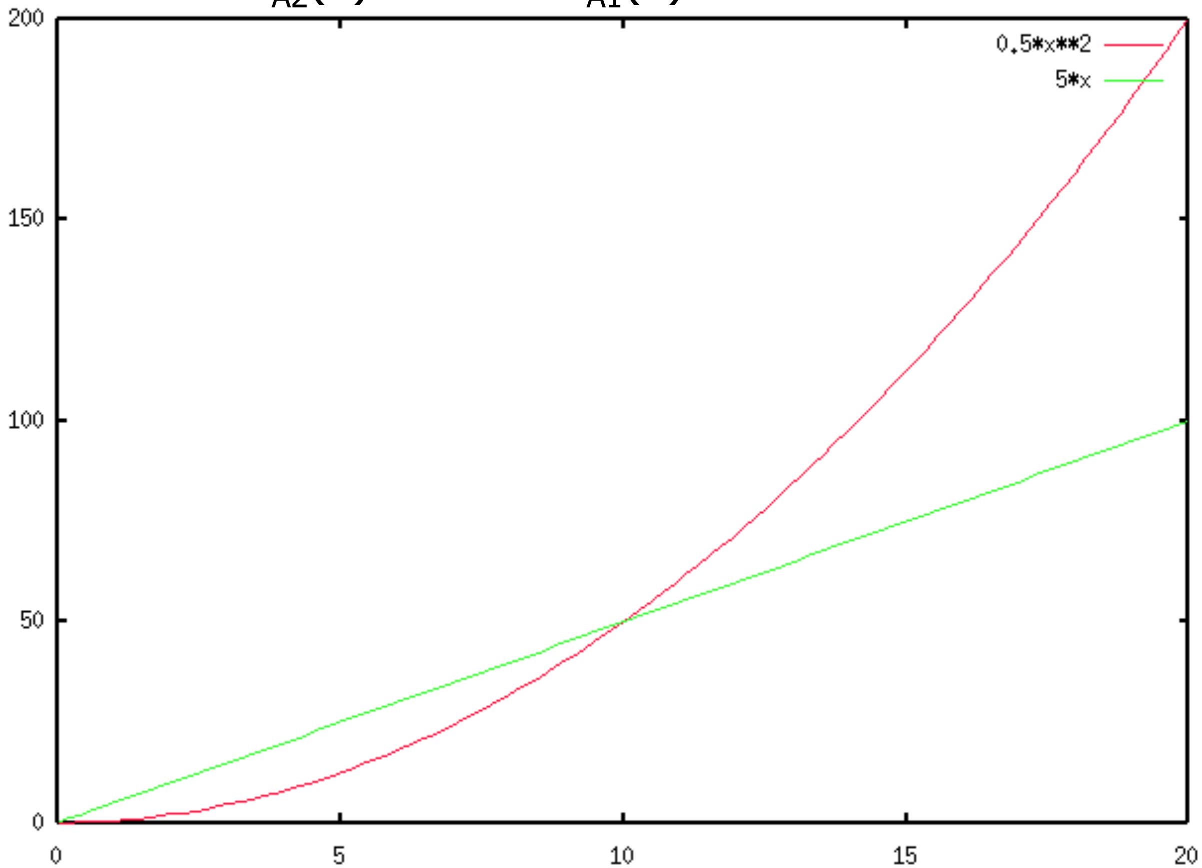
$T(n) = s(n^2 - n)/2 + t$

s is time for one HS, and t is time for getting organized

But do we always characterize the complexity of algorithms with such a detail? What is the most important aspect that we care about?

Growth of Functions

$C_{A2}(n) = 5\ n \geq C_{A1}(n) = 0.5\ n^2$ for $n \leq 10$



Two algorithms A1&A2

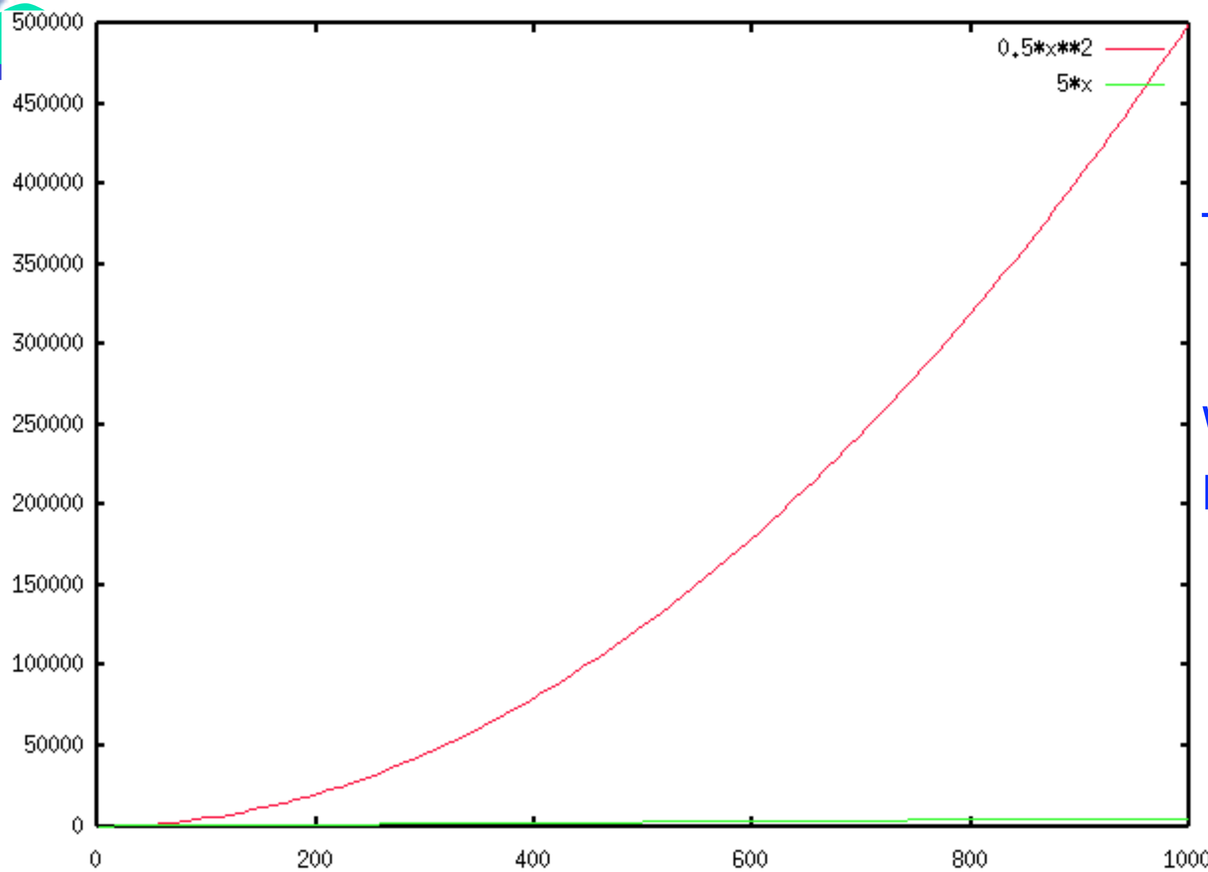
$C_{A1}(n) = 0.5\ n^2$

$C_{A2}(n) = 5\ n$

Which one is better?

Better Complexity?

Growth of Functions



Two algorithms A1&A2

$C_{A1}(n) = 0.5\ n^2$

$C_{A2}(n) = 5\ n$

Which one is better?

Better Complexity?

Main question: how the complexity behaves asymptotically – i.e., when the problem sizes tend to infinity!

Growth of Functions

In general we only worry about **growth rates** because:

- Our main objective is to analyze the cost performance of algorithms asymptotically. (reasonable in part because computers get faster and faster every year.)

- Another obstacle to having the exact cost of algorithms is that sometimes the algorithms are quite complicated to analyze.

- When analyzing an algorithm we are not that interested in the exact time the algorithm takes to run – often we only want to compare two algorithms for the same problem – the thing that makes one algorithm more desirable than another is **its growth rate relative to the other algorithm's growth rate.**

Growth of Functions

Algorithm analysis is concerned with:

- *Type* of function that describes run time (we ignore constant factors since different machines have different speed/cycle

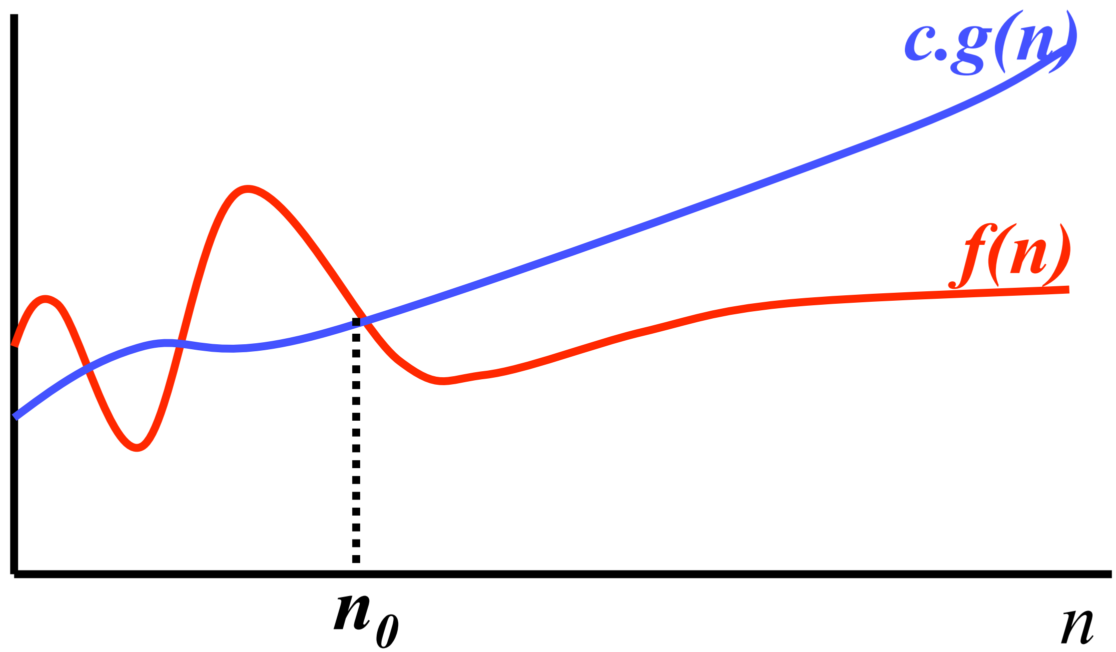
- Large values of n

Growth of Functions

Size / Complexity	10	20	30	40	50	60
n	.00001s	.00002s	.00003s	.00004s	.00005s	.00006s
n^2	.0001s	.0004s	.0009s	.0016s	.0025s	.0036s
n^3	.001s	.008s	.027s	.064s	.125s	.216s
n^5	.1s	3.2s	24.3s	1.7 mn	5.2 mn	13 mn
2^n	.0001s	1.0s	17.9 mn	12.7 days	**35.7 century**	**366 century**
3^n	.059s	58 mn	**6.5 years**	**3855 century**	**2×10^8 century**	**1.3×10^{13} century**

Assuming 10^6 operations per second

Growth of Functions



f(n) is O(g(n)) We say "f(n) is big O of g(n)"

There exist two constants ***c*** and $\boldsymbol{n_0}$ such that
$0 \le f(n) \le c.g(n)$ for $n \ge n_0$

Growth of Functions

How to prove that $5x + 100$ is $O(x/2)$

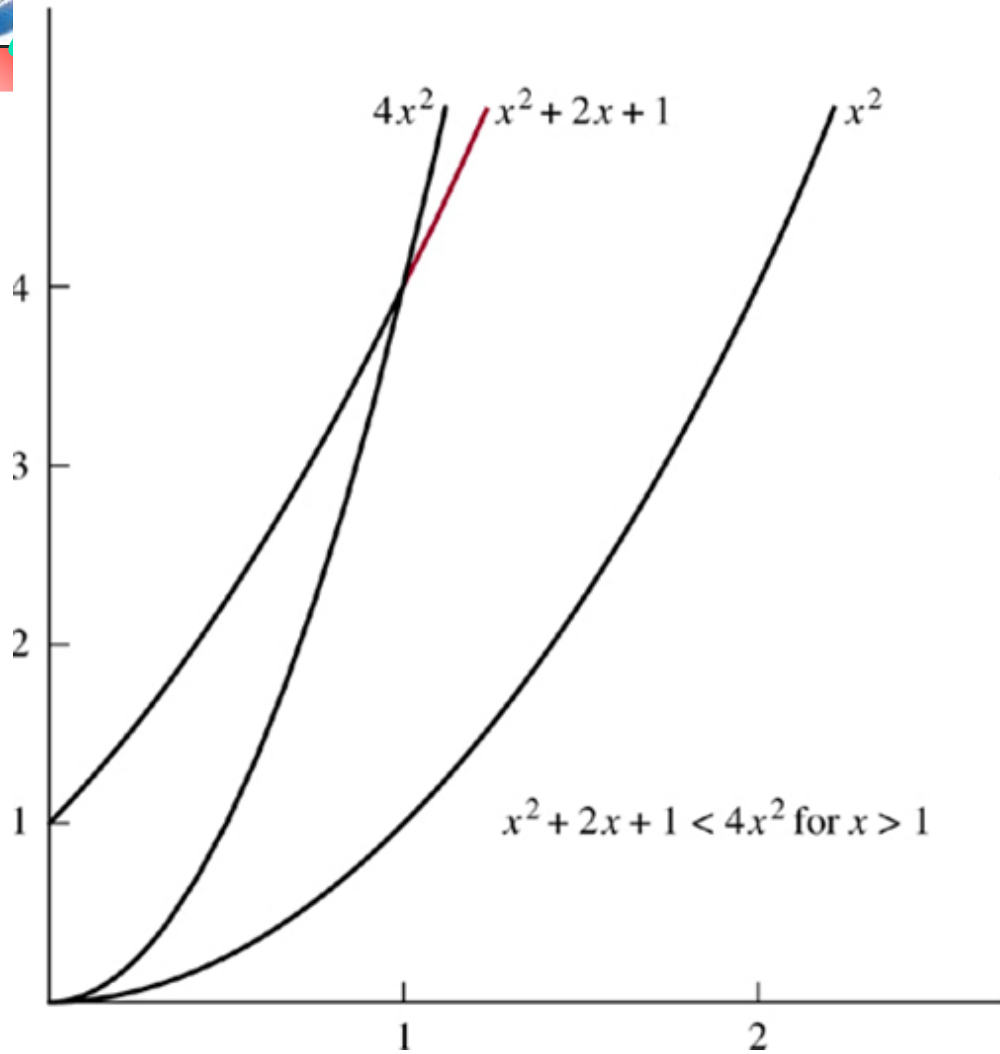
Need $\forall$ x> ___, $5x + 100 \leq$ ___ * x/2

Try $c=11$ and $n_0= 200$

$\forall x> 200, 5x + 100 \leq 11 * x/2$

(If $x> 200$ then $x/2 > 100$. Thus $11 * x/2 = 5x + x/2 >5x +100$.)

Growth of Functions

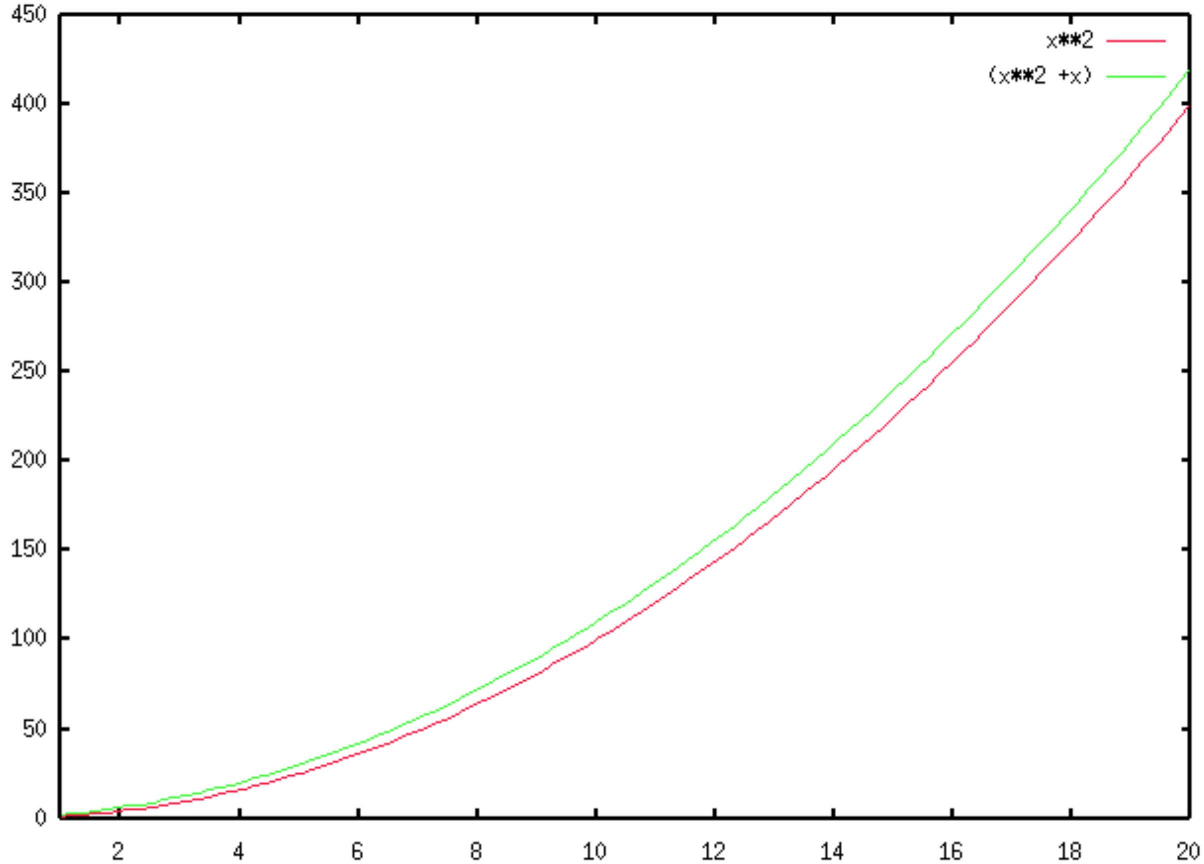


The part of the graph of $f(x) = x^2 + 2x + 1$ that satisfies $f(x) < 4x^2$ is shown in color.

$x^2 + 2x + 1$ is $O(x^2)$

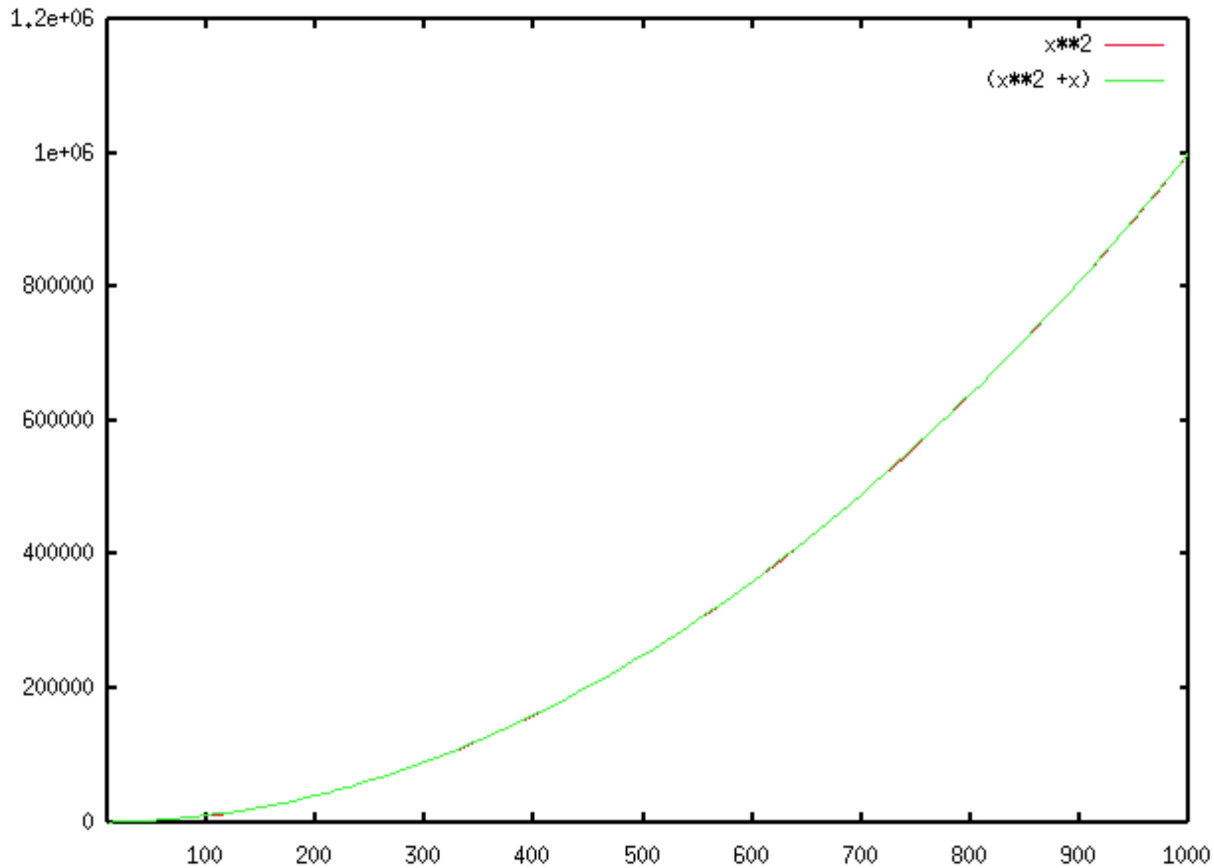
$C = 4$
$k = 1$
also
$C = 3$
$k = 2$

Growth of Functions



x^2 vs. $(x^2 + x)$
$(x <= 20)$

Growth of Functions



x^2 vs. $(x^2 + x)$

$(x^2 + x)$ is $O(n^2)$

Growth of Functions

Very useful: $f(n)=a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ then $f(n)$ is $O(n^k)$

$$
\begin{aligned}
f(n) &\leq (|a_k| + |a_{k-1} / n| + \ldots + |a_0 / n^k|)\ n^k \\
&\leq (|a_k| + |a_{k-1}| + \ldots + |a_0|)\ n^k \qquad \text{for every } n \geq 1.
\end{aligned}
$$

Guidelines:

- In general, only the largest term in a sum matters.

  $a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x^1 + a_n x^0$ is $O(x^n)$

- n dominates lg n.

  $n^5 \lg n = O(n^6)$

Growth of Functions

List of common functions in increasing O() order:

1 Constant time

n Linear time

(n lg n)

n^2 Quadratic time

n^3

…

2^n Exponential time

n!

Growth of Functions

If we have f(n) is O(g(n)) then we may say too that g(n) is Ω(f(n)) (g(n) is omega of f(n)).

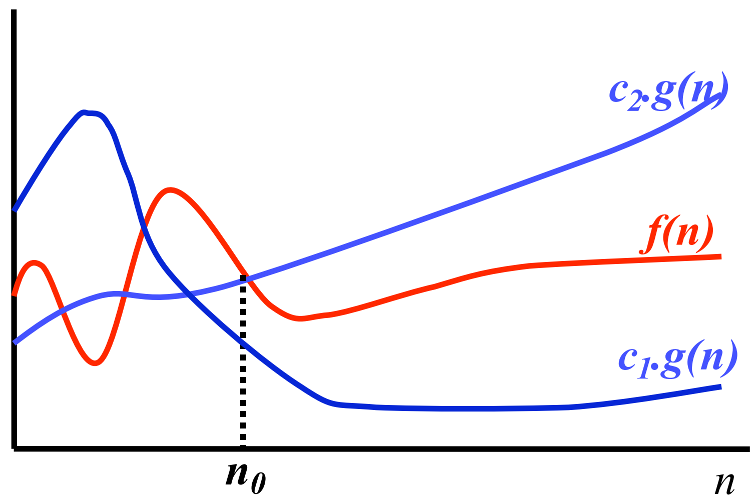
g(n) is Ω(f(n)) if and only if there exist constants *c* and n_0 such that: $\boxed{g(n) \geq c\, f(n)}$ for all $n \geq n0$

A function g(n) is Θ(f(n)) (g(n) is Theta of f(n)) if g(n) is O(f(n)) and g(n) is Ω(f(n)) .

The f(n) and g(n) functions have the same growth rate.

When we write f is O(g), it is like f ≤ g
When we write f is Ω(g), it is like f ≥ g
When we write f is Θ(g), it is like f = g.

Growth of Functions



f(n) is Θ (g(n))

There exist 3 constants c_1, c_2 and n_0 such that

$$c_1 g(n) \le f(n) \le c_2 g(n) \quad \text{for } n \ge n_0$$

Growth of Functions

Use the limit for comparing the order of growth of two functions.

$$\lim_{n \to \infty} g(n)/f(n) = \begin{cases} 0 & \text{then } g(n) \text{ is } O(f(n)) \\ & \text{[but } g(n) \text{ is not } \Theta(f(n))\text{]} \\ c > 0 & \text{then } g(n) \text{ is } \Theta(f(n)) \\ \infty & \text{then } g(n) \text{ is } \boldsymbol{\Omega}(f(n)). \\ & \text{[but } g(n) \text{ is not } \Theta(f(n))\text{]} \end{cases}$$

Estimating Functions

Estimate the sum of the first n positive integers

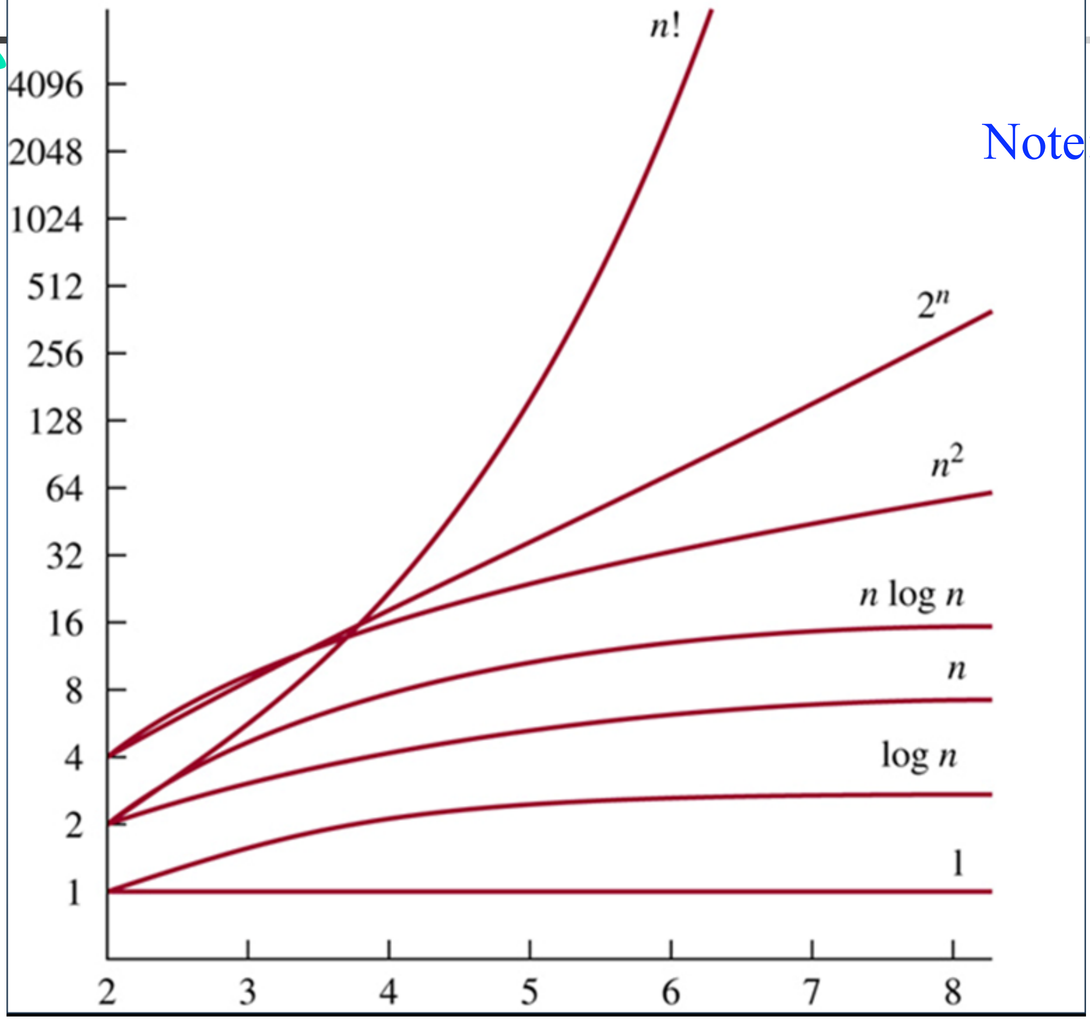
$1 + 2 + \ldots + n = n(n+1)/2 = n^2/2 + n/2$ is $\Theta(n^2)$

$(1 + 2 + \ldots + n)$ is $\Theta(n^2)$

What about $f(n) = n!$ and $\log n!$

$n! = 1*2* \ldots *n \leq n*n* \ldots *n = n^n$. Thus $n!$ is $O(n^n)$.

$\log n! \leq \log n^n = n \log n$. Thus $\log n!$ is $O(n \log n)$.



Note: log scale on y axis.