MANCHESTER 1824
The University of Manchester

Complexity

The running time of algorithms

Joshua Knowles

School of Computer Science

The University of Manchester

COMP26120-Week 5 LT1.1, October 20 2014

Who Cares About Complexity?

Interview questions by top technical companies from (from xorswap.com):

Examples:

Which of the following tree data structures guarantee O(log n) lookup?

Google over 1 year ago 2 answers

- Red-black tree - AVL tree - Splay tree

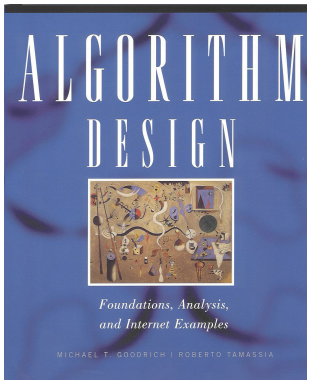
Design a data structure that allows O(1) insert/add/delete of an element, as well as O(1) lookup of a random element

Microsoft over 1 year ago

The required data structure is a hash set. At the high level, a hash set includes two internal structures: a hash table and a sequential array. Each pair in the hash table is (*element, index*), where *index* is the location of *element* in the sequential array. Insertion...

Algorithms and Imperative Programming

Rest of the course will focus on ***algorithms*** and data structures.



⟵ You need this book

Please read:
Pages 13–20 deal with Big-Oh Notation, which is our subject today.
Pages 20–24 help with some mathematics you might need.
Pages 24–27 help with Proof techniques.

Please also bring a ***calculator***

In this lecture...

$\Rightarrow$ Why running time is expressed as a function of input size

$\Rightarrow$ The 'big-Oh' notation (asymptotic notation)

$\Rightarrow$ Practice recognizing 'sizes' of expressions such as:

$$\log n \quad \sqrt{n} \quad n \quad n^2 \quad 2^n \quad n!$$

Analysis of algorithms

Algorithm Hello N times:
1: **input** $name, N$
2: $i \leftarrow 0$
3: **while** $i < N$ **do**
4: **output** 'hello' $name$, i++

There are two main things to analyse:

- Correctness
- ***Complexity***

Complexity predicts the ***resources*** the algorithm requires

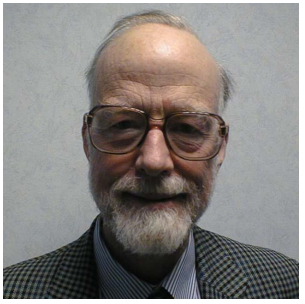
Time complexity relates to the running time;
Space complexity relates to the memory usage

The importance of running time

Running time is critical to the success of the algorithms used above. Why?

The importance of running time

Famous time complexity results



Tony Hoare

Peter Shor

Karankar's algorithm for linear algebra
$O(n^{3.5}L^2 \ln L \ln \ln L)$

Tony Hoare's Quicksort algorithm
$O(n \log n)$

Strassen's algorithm for matrix multiplication
$O(n^{2.807})$

Peter Shor's quantum factorization algorithm
$O((\log n)^3)$

On what does running time depend?

ACTIVITY

- Choose a letter A, B, C, or D.

For A,B,C you just need pen and paper
For D, you need a calculator

- Do the task of your letter
- Shout out your letter when you've finished.

Ready?

SHOUT OUT YOUR LETTER WHEN FINISHED !

A. Write down your full name

B. Write out the letters of your full name in alphabetical order

C. Write out every anagram of just your first name

D. ON your calculator:

Type a random 3 digit number
Repeat (until number == 8 OR 0):
    Divide number by 3 (throw away any decimal remainder)
    If the answer is even, multiply it by 4
    Else add 1

On what does running time depend?

Examples:

- Uploading / downloading a file

- Finding a word in a dictionary

- Code-breaking by brute force search

Other examples?

Evaluating an algorithm: how not to do it

Jenny: My algorithm can sort 10^6 numbers in 3 seconds.
Mo: My algorithm can sort 10^6 numbers in 5 seconds.

Jenny: I've just tested it on my new Pentium IV processor.
Mo: I remember my result from my PhD studies (1995).

Jenny: My input was a random permutation of $1..10^6$.
Mo: My input was the sorted output, so I only needed to verify that it is sorted.

$\longrightarrow$ Problems can occur if comparisons are not like-for-like.

We need to abstract away these issues. Aymptotic analysis (big-Oh) helps do this.

Rate of Growth with Input Size

We said: running time often depends mainly upon the input size.

Larger inputs lead to larger running times (usually).

$\Rightarrow$ It is the ***rate of growth*** (with input size) that matters.

This is what we want to describe when analysing algorithms.

Functions

Running times can be expressed as a (growing) function of the input size n,

$$t = f(n) \quad .$$

The function $f(n)$ represents the number of ***primitive operations*** the algorithm performs on an input of size n.

e.g.

$t = f(n) = 2n$
$t = f(n) = n^2$
$t = f(n) = \log(n) + 10$
$t = f(n) = 6n^3 + 2^n + 24$

Functions

$f(n)$ could even be 1. E.g.:

Algorithm Top item:
1: **require** $list$
2: **return** $list$[1]

The running time could depend on two parameters, e.g. $n^2 + m^3$. That would be a function $f(n, m)$.

(We will consider functions of two variables in a later lecture.)

Comparing functions

Now let us compare two algorithms

A	B
running time $= f(n)$	running time $= g(n)$

We wish to know which algorithm is faster for large n.

Big O and Asymptotics

Asymptotic analysis concerns large values of n.

With asympotic analysis, we are not concerned if $f(n)$ is, say, 2 or 3 or even a 100 times larger than $g(n)$.

Why don't these constants matter much?

We only care whether $f(n)$ is of a different ***order*** than $g(n)$.

In other words, we only care if the relative size (or ratio) of $f(n)$ to $g(n)$ keeps growing as n grows (large).

Example 1

Compare $f(n) = 10n$ with $g(n) = 2n$.

The ***relative*** size of the two functions does **not** grow.

$$\frac{f(n)}{g(n)} = \frac{10n}{2n} = 5,$$

which is a constant.

So we say that $10n$ and $2n$ are ***of the same order.***

They grow at the same rate ***up to a constant factor***.

We can also write that $10n$ is $O(2n)$.
(read as “big-oh of 2n” or “just oh of 2n”)

It is normal to write that $10n$ is $O(n)$. Can you see why?

Ignoring Constant Coefficients

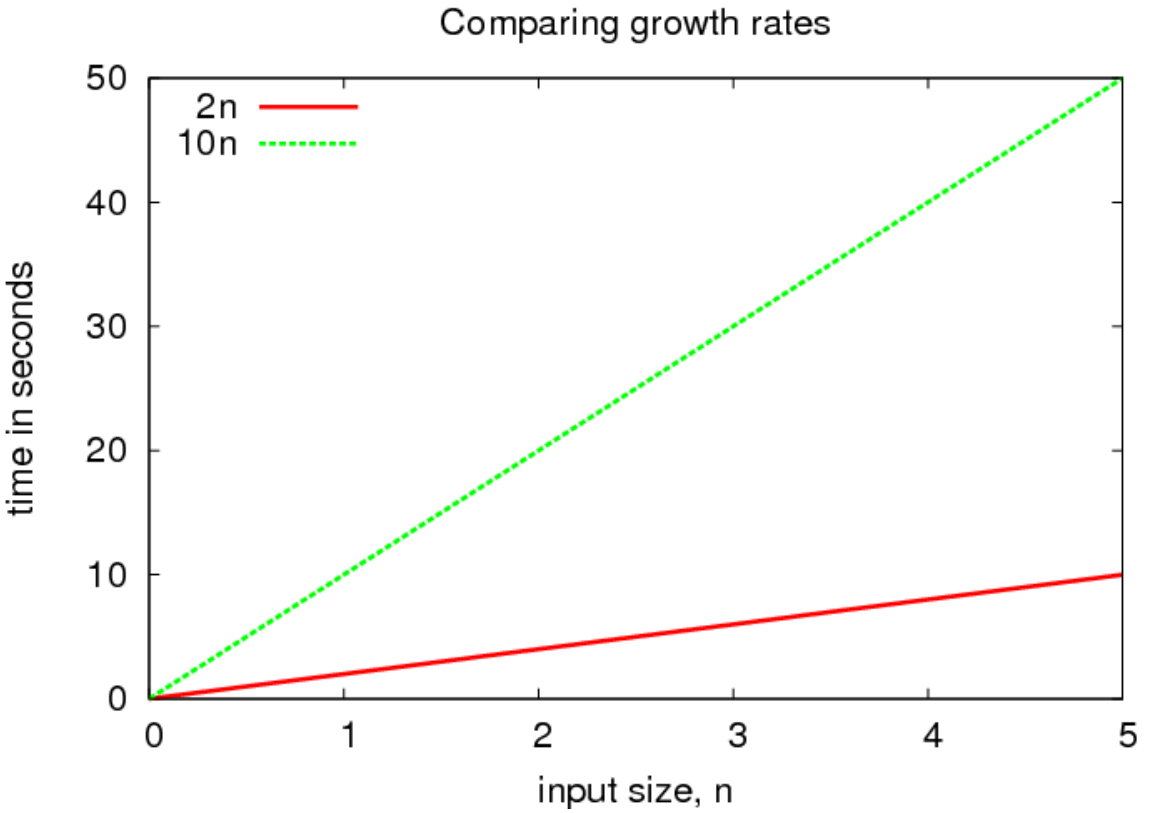
Informally,

$O(n)$ "Oh of n" means: (of the order of) some constant times n

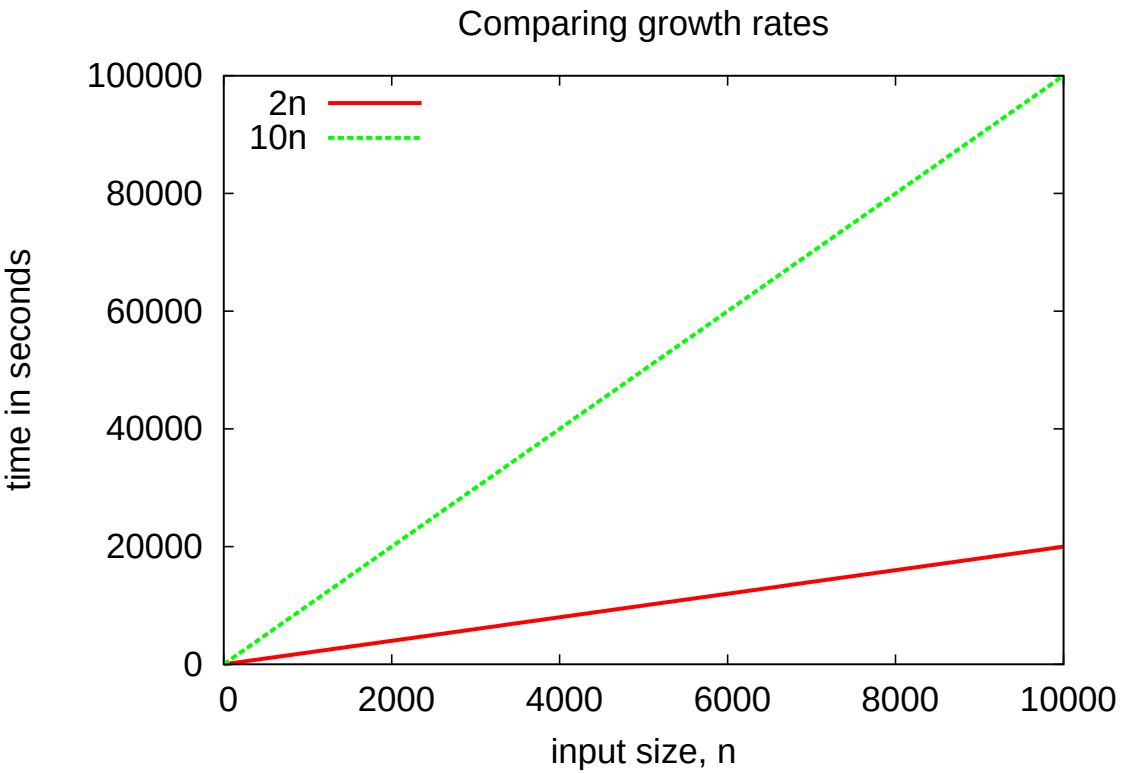
It could be $10n$ or $2n$, and we don't care.

This may seem odd, but it allows us to hide or not talk about things that are affected by implementation issues such as how fast our computer is, what language we are using, or whether some concurrent task is being run.

Example 1


Comparing growth rates
2n
10n
time in seconds
input size, n
0
10
20
30
40
50
0
1
2
3
4
5

Example 1 - Zoom out



Example 2

Compare $f(n) = n^2$ and $g(n) = n$.

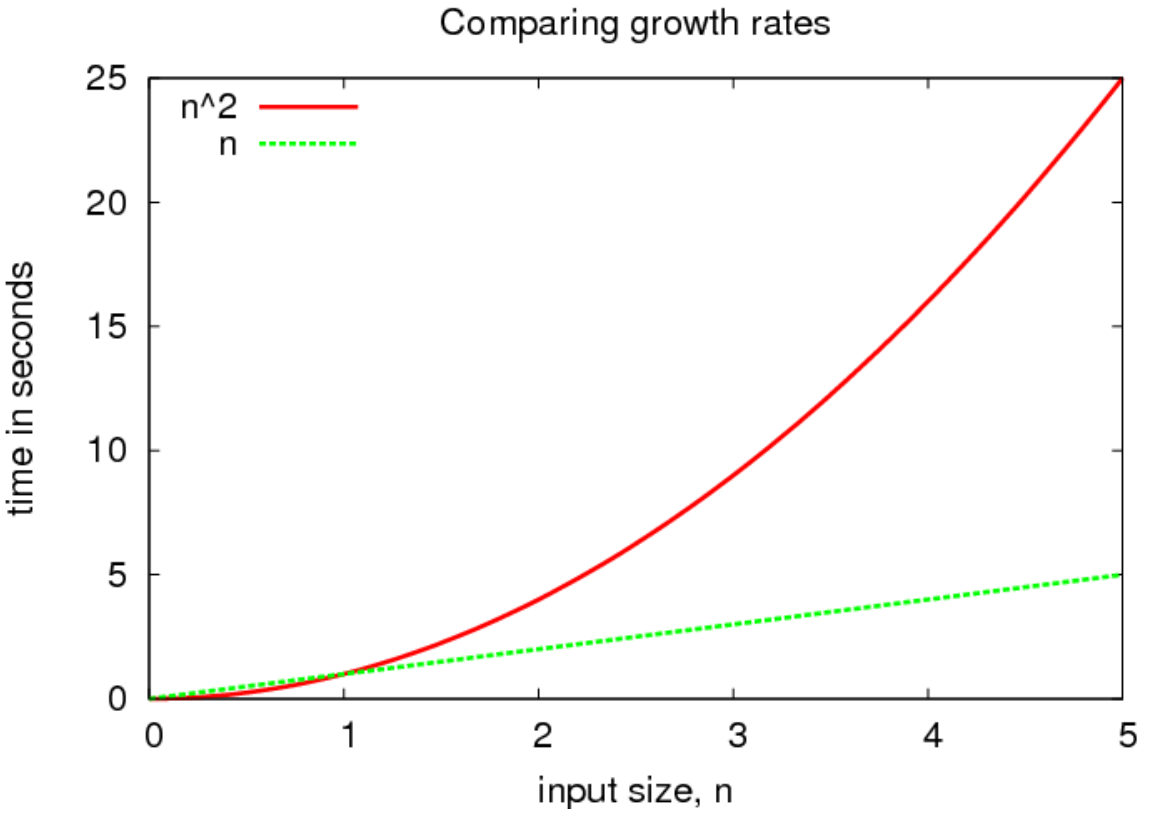
The relative size of n^2 to n grows without bound (forever) as n grows.

$$\frac{n^2}{n} \text{ is not a constant}$$

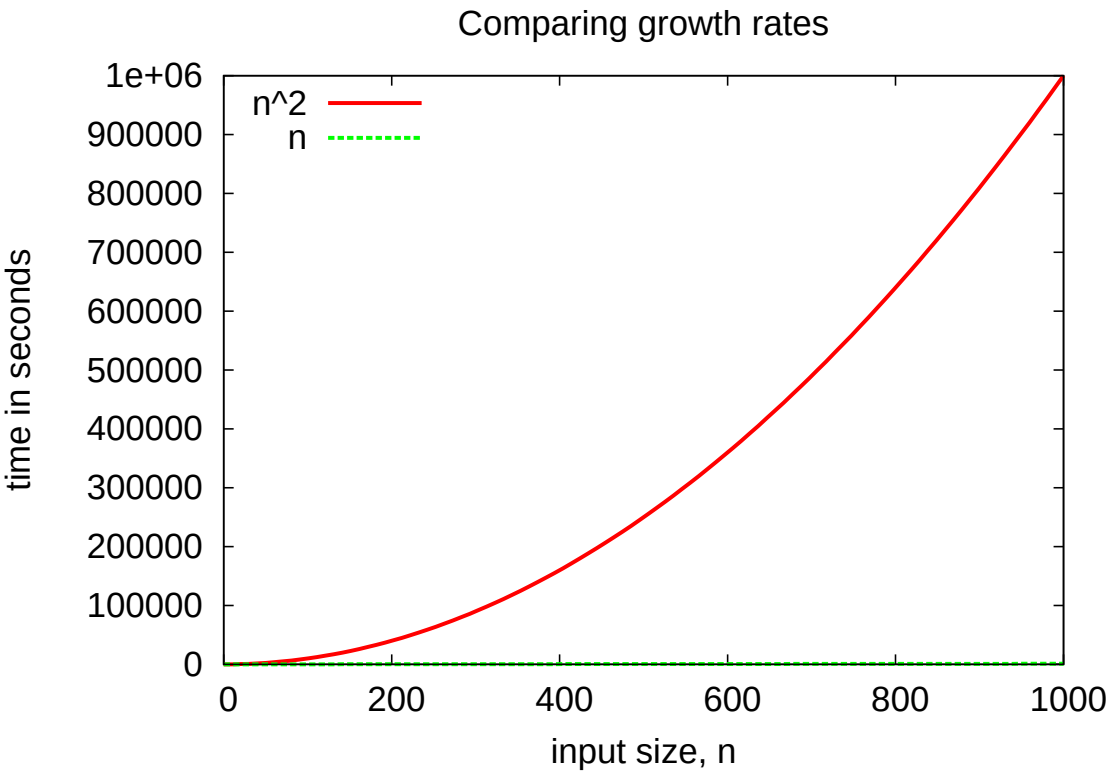
So we can say that n^2 is of a larger order than n.

Another way to write that is n^2 is not $O(n)$.

Example 2


Comparing growth rates
n^2
n
time in seconds
input size, n
0
5
10
15
20
25
0
1
2
3
4
5

Example 2 - Zoom Out



Example 3

Compare $f(n) = n^2 + n$ and $g(n) = n^2$.

$f(n)$ **looks** bigger again.

But

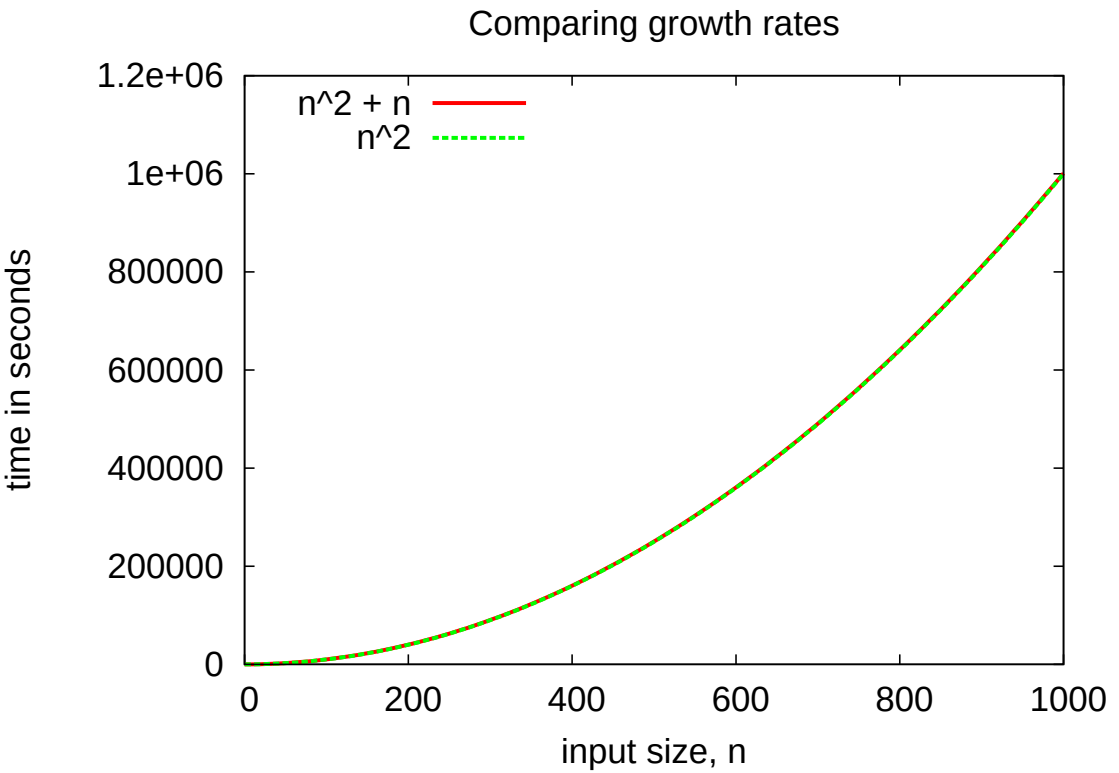
$$\frac{f(n)}{g(n)} = \frac{n^2 + n}{n^2} \approx 1$$

if n is large enough.

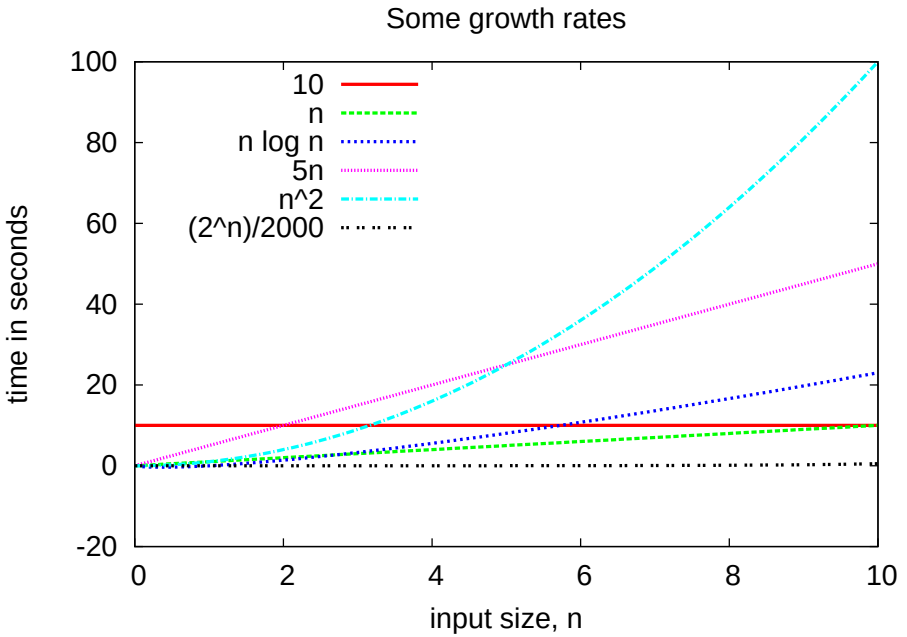
The ratio of the two functions ***approaches*** 1.

So $n^2 + n$ is $O(n^2)$. They are of the same order.

Example 3 - Zoom Out

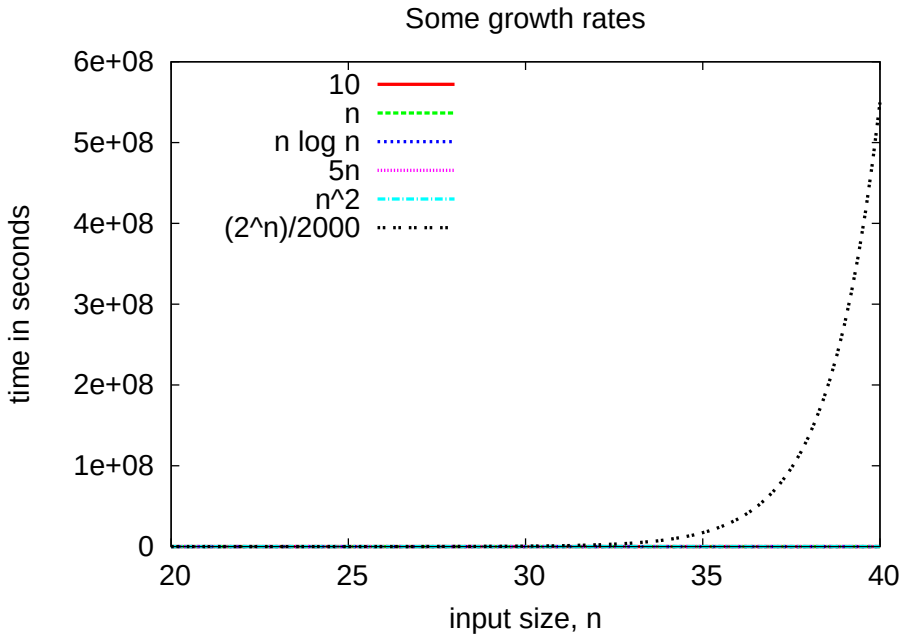


Growth rates compared



Which is the fastest algorithm?

Growth rates compared - Zoom Out



Which is the fastest algorithm?

The Importance of Asymptotics

What is the largest problem algorithm A can solve in fixed time?

Running time of A (μs)	Maximum Problem Size (n)		
	1 second	1 minute	1 hour
$400n$	2,500	150,000	9,000,000
$20n\lceil \log n \rceil$	4,096	166,666	7,826,087
$2n^2$	707	5477	42,426
n^4	31	88	244
2^n	19	25	31

Notice: In the example above, with exponential time complexity (2^n) *multiplying* resources by 3600 only *adds* 12 to the maximum size of input.

Descriptive Terms for Growth Rates

$$\log n \qquad \sqrt{n} \qquad n \qquad n^2 \qquad 2^n \qquad n!$$

If the largest (or leading) term is

- $\log n$ it is a ***logarithmic*** growth rate
- n^k for any integer $k > 0$ it is a ***polynomial*** g.r.
- $\sqrt{n}$ is $n^{1/2}$ so this grows less quickly than polynomial

- n is n^1 is a polynomial g.r. too
- n is also known as ***linear*** (a special case of a polynomial)
- n^2 is ***quadratic*** and n^3 is ***cubic***. These are both just special polynomials too.

- 2^n is ***exponential***
- $n!$ is ***factorial***

The growth rates above the line are ***tractable***. That means we can usually handle them even for large n.

The growth rates below the line are ***intractable***. For large n they can easily run for longer than the age of the Universe (on any computer).

A Simple Rule to Simplify Many Big-Oh Expressions

Identify the fastest growing term, and write it in its simplest form. Drop all the other terms.

e.g.:

$$O(5n^3 + \log n) \text{ is } O(n^3)$$

(See your textbook, p15 for a fuller list: 8 simplification rules.)

Summary

Big Oh notation allows us to describe algorithm running time (or space) as a function of n

It simplifies the exact running time

It allows us to compare functions: to establish if they grow at the same or different rates (up to a constant factor)

Big Oh notation is also called: Landau notation, BachmannLandau notation, and asymptotic notation

Closing Remarks

For more practice:
Try the worksheet; ***Try gnuplot***

Rest of the course:
Analyse the complexity of specific algorithms using Big Oh

Lab:
Do an algorithm to find centiles and analyse its time complexity

Next Lecture:
Much more practice on analysis of algorithms

Who Cares About Complexity?

Running time analysis is related to an exciting theoretical field:

...computational complexity theory— the field that studies the resources (such as time, space, and randomness) needed to solve computational problems—leads to new perspectives on the nature of mathematical knowledge, the strong AI debate, computationalism, the problem of logical omniscience, Hume's problem of induction, Goodman's grue riddle, the foundations of quantum mechanics, economic rationality, closed timelike curves, and several other topics

Scott Aaronson (2011)

Who Cares About Complexity?

On whether the PvsNP problem is worth a Clay Millennium (million dollar) prize:

> Dude. One way to measure P vs. NP's importance is this. If NP problems were feasible, then mathematical creativity could be automated. The ability to check a proof would entail the ability to find one. Every Apple II, every Commodore, would have the reasoning power of Archimedes or Gauss.

from http://www.scottaaronson.com/democritus/lec6.html