Computational complexity

Adam Richards

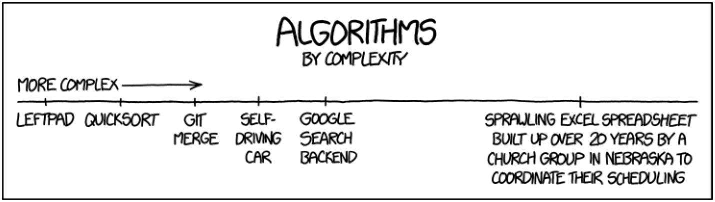
Galvanize, Inc

2017

Objectives

- Exposure to computational complexity concepts
- Exposure to design patterns
- Provide resources for further study

How do we describe the difficulty of problems and algorithms designed to solve such problems?



Calculating the covariance matrix is somehow **harder** than calculating the mean for a matrix and not all algorithms will use the same amount of memory or have the same speed when it comes to tackling this task.

With Big-oh we are trying to...

1. specify differences in problems and algorithms independent of current computer hardware
2. Tells us about how an algorithm scales

To be more specific we use the concept of the order of a function and the asymptotic notations big oh, big omega, and big theta.

Asymptotic bounds

Asymptotic upper bound $O(g(x)) = \{f(x)$: there exist positive constants c and x_0 such that $0 \le f(x) \le cg(x)$ for all $x \ge x_0\}$

Asymptotic lower bound $\boldsymbol{\Omega}(g(x)) = \{f(x)$: there exist positive constants c and x_0 such that $0 \le cg(x) \le f(x)$ for all $x \ge x_0\}$

Asymptotically tight bound $\boldsymbol{\Theta}(g(x)) = \{f(x)$: there exist positive constants c_1, c_2, and x_0 such that $0 \le c_1 g(x) \le f(x) \le c_2 g(x)$ for all $x \ge x_0\}$

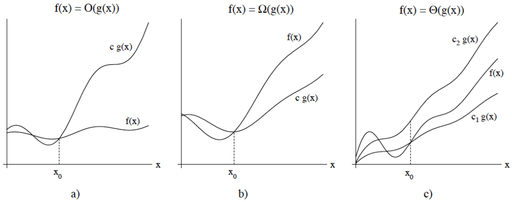


Figure A.6: Three types of asymptotic bounds: a) $f(x) = O(g(x))$. b) $f(x) = \boldsymbol{\Omega}(g(x))$. c) $f(x) = \boldsymbol{\Theta}(g(x))$.

Duda, Hart, Stork: Pattern Classification (pp 633)

Big-oh

We say that

$f(x)$ is of order big oh of $g(x)$

and that is written as

$$f(x) = O(g(x)) \tag{1}$$

We assume that all functions are positive so we do not have to talk about absolute values

this means that for a sufficiently large x an upper bound on $f(x)$ grows no worse than $g(x)$

an example

For a given Big oh we have constants c_0 and x_0 such that $f(x) \leq c_0 g(x)$ for all $x > x_0$. If $f(x) = a + bx + cx^2$ then $f(x) = O(x^2)$ because for sufficiently large x, the constant, linear and quadratic terms can be overcome by proper choice of c_0 and x_0.

- The generalization to functions of two or more variables is straightforward
- The (big oh) order of a function is not unique
- A particular $f(x)$ can be as $O(x^2)$, $O(x^3)$, $O(x^4)$, $O(x^2 \ln x)$ etc.

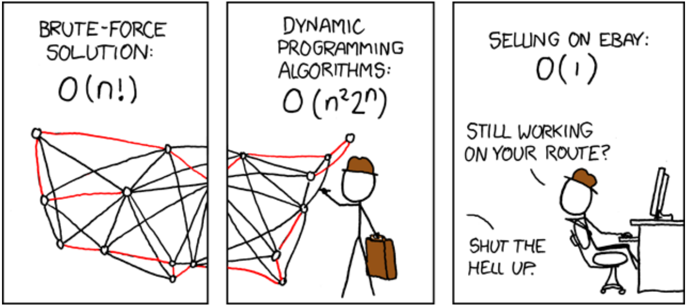
Sorting

- Sorting is a classic example for discussing of Big Oh
- Algorithms: Bubble sort, insertion sort, merge sort, heap sort, and quick sort
- Bubble and insertion sort are $O(n^2)$ (fairly slow)
- Merge, heap and quick are $O(n * logn)$.

That means, given 10000 items, the $O(n^2)$ slow sorts will take about 1,000,000 steps

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

https://xkcd.com/1185/



https://xkcd.com/399/

Where is this used?

Some things are known

The complexity ofsome problems — such as computing the mean of a discrete set– is known, and thus once we have found an algorithm having equal complexity, the only possible improvement could be on lowering the constants of proportionality.

Some things re unknown

The complexity of other problems such as inverting a matrix is unknown so we must rely on comparing the algorithms that solve these problems

Not always intuitive

For example, it is absolutely correct to say that binary search runs in $O(n)$ time. That's because the running time grows no faster than a constant times n. In fact, it grows slower.

Suppose you have 10 dollars in your pocket. You go up to your friend and say, 'I have an amount of money in my pocket, and I guarantee that it's no more than one million dollars.' Your statement is absolutely true, though not terribly precise. One million dollars is an upper bound on 10 dollars

https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation

an example

```
my_list = range(10)
for x in my_list:
    run_function(x)
    for y in my_list:
        run_function(y)
        for z in my_list:
            run_function(z)
```

Often you can estimate the big oh of your algorithm by counting the number of for loops.

i.e. Here we have $O(n^3)$

Performance considerations

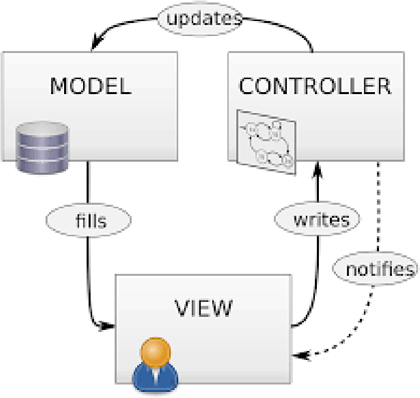
Usually we discuss Big oh wrt time-complexity (CPU), but there are other considerations

- memory usage
- disk usage
- network usage
- latency

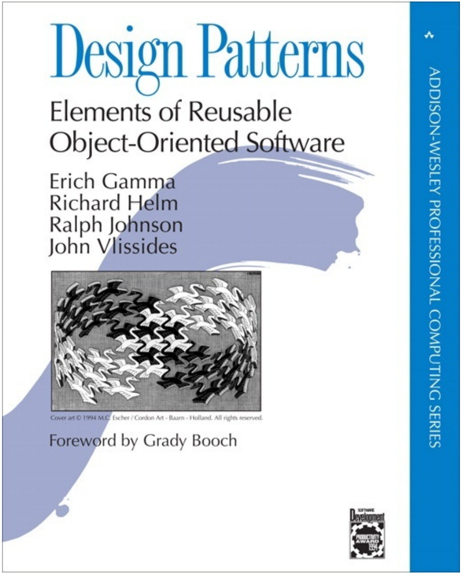
Performance and complexity are not the same

$$\frac{\text{sample}_n}{\text{sample runtime}} = \frac{\text{all}_n}{X} \tag{2}$$

Model view controller



Wiki MVC


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Resources

- Khan Academy
- MIT big oh lecture
- A gentle intro to computational complexity
- Computational complexity theory (Stanford)
- Beginner's guide to design patterns
- http://www.oodesign.com/

References I