Algorithm Design & Analysis (CS3383)[1]

Unit 0: Asymptotic Review

Rasoul Shahsavarifar

January 7, 2019

[1]Thanks to Dr. Ptricia Evans and Dr. David Bremner at UNB, and Dr. Erik Demaine at MIT for sharing the teaching stuffs

Outline

First Analysis, then Design !

Unlike the order in the course title,

- First, we learn some concepts which helps to analyze an algorithm.
- Then, we learn different algorithm design techniques.

First Analysis, then Design !

Unlike the order in the course title,

- First, we learn some concepts which helps to analyze an algorithm.
- Then, we learn different algorithm design techniques.

Why do we analyze Algorithms?

From all correct algorithms, we want to be able to pick those that are the best:

- Least time or fewest number of steps
- Least use of other resources (such as space)
- Possibly easiest to implement or use based on some other criteria.

First Analysis, then Design !

Unlike the order in the course title,

- First, we learn some concepts which helps to analyze an algorithm.
- Then, we learn different algorithm design techniques.

Why do we analyze Algorithms?

From all correct algorithms, we want to be able to pick those that are the best:

- Least time or fewest number of steps
- Least use of other resources (such as space)
- Possibly easiest to implement or use based on some other criteria.
- Since speed is fun, one of the usual criterion is time: of those that work, which one is the fastest?
- We want to be able to make our choice without implementing, since implementation can be expensive.

Unit prereqs

- O and Ω (CS2383)
- limits, derivatives (calculus)
- induction (CS1303)
- working with inequalities
- monotone functions

Asymptotic Analysis

Question: Does Asymptotic Analysis always works perfectly?

Asymptotic Analysis

Question: Does Asymptotic Analysis always works perfectly?

Given: Two algorithms with $10000n \log n$, and $5n \log n$ time complexities.
Task: Designing a software that deal with inputs with the size of at most 1000.

Asymptotic Analysis

Question: Does Asymptotic Analysis always works perfectly?

Given: Two algorithms with $10000n \log n$, and $5n \log n$ time complexities.
Task: Designing a software that deal with inputs with the size of at most 1000.

But,

In general, asymptotic analysis is the best available way to analyze an algorithm.

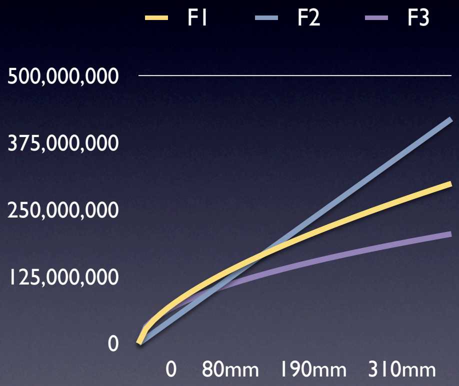
Contents

Asymptotic Notation

f1: $1{,}000 * n^{0.635}$

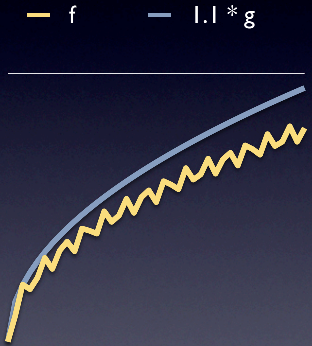
f2: n

f3: $10{,}000 * n^{0.5}$



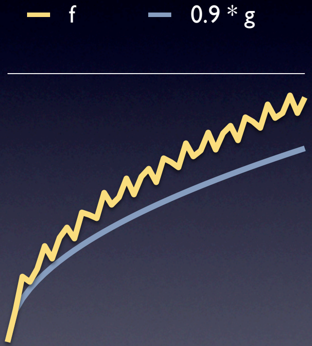
Asymptotic Notation

- f = O(g)

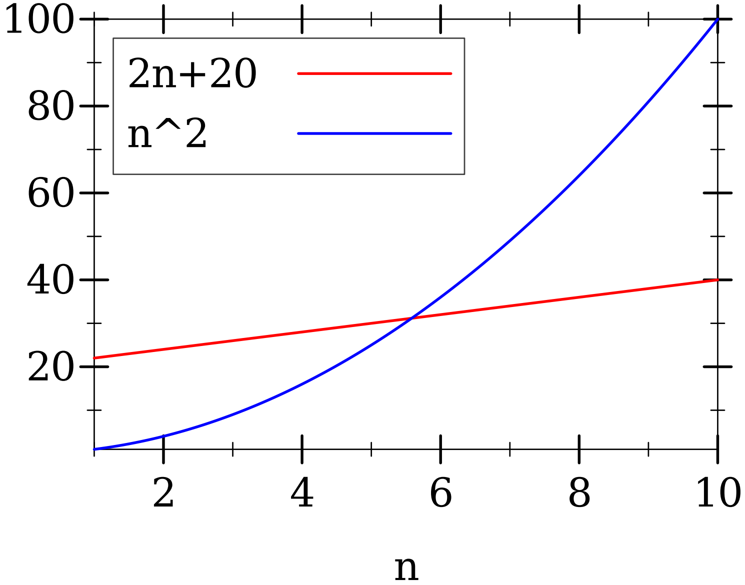


Asymptotic Notation

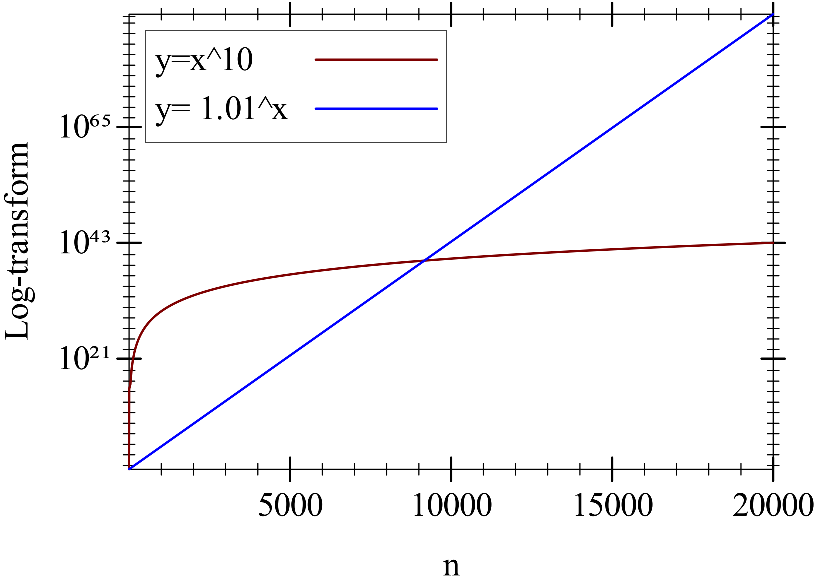
- $f = \Omega(g)$



Linear versus Quadratic



Exponential versus Polynomial

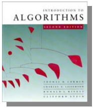


Contents

Asymptotic notation

O-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.



Asymptotic notation

O-notation (upper bounds):

> We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

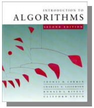
Asymptotic notation

O-notation (upper bounds):

> We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

functions, not values



Asymptotic notation

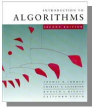
O-notation (upper bounds):

> We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

functions, not values

funny, "one-way" equality



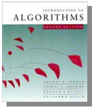
Set definition of O-notation

$O(g(n)) = \{ f(n) :$ there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$ $\}$

Set definition of O-notation

$O(g(n)) = \{ f(n) :$ there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$ $\}$

EXAMPLE: $2n^2 \in O(n^3)$

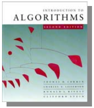


Set definition of O-notation

$O(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0 \}$

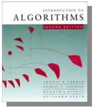
EXAMPLE: $2n^2 \in O(n^3)$

(*Logicians:* $\lambda n.2n^2 \in O(\lambda n.n^3)$, but it's convenient to be sloppy, as long as we understand what's *really* going on.)



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.



Macro substitution

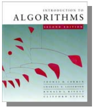
Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $f(n) = n^3 + O(n^2)$

means

$f(n) = n^3 + h(n)$

for some $h(n) \in O(n^2)$.



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.

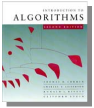
EXAMPLE: $n^2 + O(n) = O(n^2)$

means

for any $f(n) \in O(n)$:

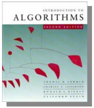
$$n^2 + f(n) = h(n)$$

for some $h(n) \in O(n^2)$.



Ω-notation (lower bounds)

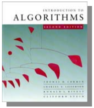
O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.



Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0 \}$

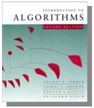


Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$ $\}$

Example: $\sqrt{n} = \Omega(\lg n)$ $\quad (c = 1, n_0 = 16)$



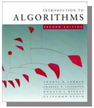
Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \;\cap\; \Omega(g(n))$$

EXAMPLE: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

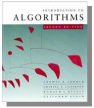


o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$o(g(n)) = \{ f(n)$: for any constant $c > 0$,
there is a constant $n_0 > 0$
such that $0 \leq f(n) < cg(n)$
for all $n \geq n_0$ $\}$

EXAMPLE: $2n^2 = o(n^3)$ $\quad (n_0 = 2/c)$



o-notation and ω-notation

O-notation and Ω-notation are like $\leq$ and $\geq$.
o-notation and ω-notation are like $<$ and $>$.

$\omega(g(n)) = \{ f(n)$: for any constant $c > 0$,
there is a constant $n_0 > 0$
such that $0 \leq cg(n) < f(n)$
for all $n \geq n_0$ $\}$

EXAMPLE: $\sqrt{n} = \omega(\lg n)$ $\quad (n_0 = 1 + 1/c)$

Contents

Working with asymptotic notation (board)

- brute force algebra
- bounding with constants
- little o

Case Analysis

Cool example! (board)

Consider different inputs ($2D$ objects) for a triangulation algorithm.

- Worst Case(s)
- Best Case(s)
- Average Case: over all inputs, according to the input distribution

Case Analysis

Cool example! (board)

Consider different inputs ($2D$ objects) for a triangulation algorithm.

- Worst Case(s)
- Best Case(s)
- Average Case: over all inputs, according to the input distribution

Randomized Algorithms

- Expectation: over all internal random choices, according to the distribution of random choices

Analysis of bubble sort(board)

Input : integer array $A[1..n]$
Output: sorted array A

```
continue ← True
while continue do
    continue ← False
    for i = 1 to n − 1 do
        if A[i] > A[i + 1] then
            temp ← A[i]
            A[i] ← A[i + 1]
            A[i + 1] ← temp
            continue ← True
```