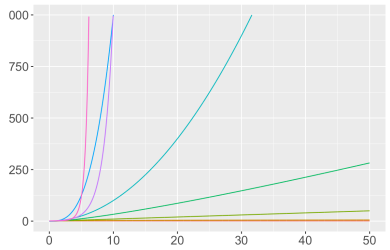


Big-O and Big-Θ
Lecture 3

COSC 242 – Algorithms and Data Structures

Today’s outline

1. Rates of growth
2. Big-Θ and Big-O
3. Proof for Insertion Sort

Today’s outline

1. Rates of growth
2. Big-Θ and Big-O
3. Proof for Insertion Sort

Landmarks

We saw that the amount of work done by Insertion Sort, in the worst case, is roughly indicated by:

$$f(n) = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

We'd like to tie this in to some special **landmark functions:**

$f(n) = 1$ $\quad$ $f(n) = \log n$ $\quad$ $f(n) = n$

$f(n) = n \log n$ $\quad$ $f(n) = n^2$ $\quad$ $f(n) = n^3$

$\cdots$

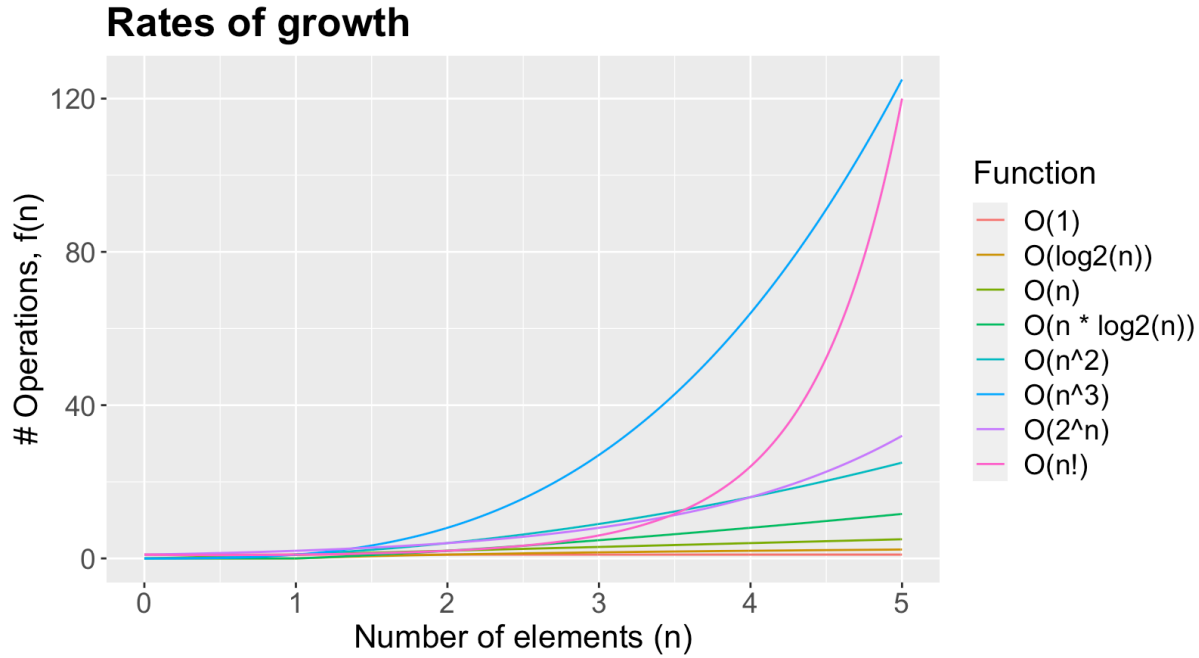
$f(n) = 2^n$ $\quad$ $f(n) = n!$

Given an algorithm, we estimate how much work it would have to do in the worst case (call this estimate the time-complexity function of the algorithm) and then we identify which landmark it should be grouped with.

Rates of growth

The landmark functions have different rates of growth.

By the *rate of growth* of f we mean how fast its output $f(n)$ increases in size as the input n gets bigger. Intuitively, a function with a slow rate of growth **scales up more efficiently** than a function with a high rate of growth.



Are plots enough?

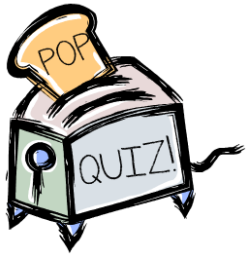
Lets take a look using the following two growth functions:

$$f(x) = 10^{-8} * x^2$$

and

$$f(x) = {}^{100}/_{(1 + e^{(10-x)})}$$

Pop quiz 1



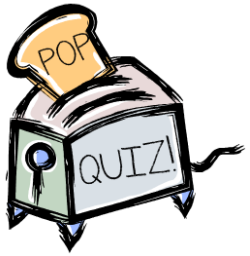
Which function do you think will grow faster?

1. $f(x) = 10^{-8} * x^2$

OR

2. $f(x) = {}^{100}/_{(1+e^{(10-x)})}$

Pop quiz 1



Which function do you think will grow faster?

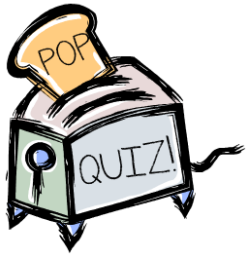
1. $f(x) = 10^{-8} * x^2$

OR

2. $f(x) = {}^{100}/_{(1+e^{(10-x)})}$

Lets find out…

Pop quiz 2



Are plots enough?

Answers

Yes

No

Today’s outline

1. Rates of growth
2. Big-Θ and Big-O
3. Proof for Insertion Sort

$\boldsymbol{\Theta}$-notation (Big-theta)

In Lecture 1, we found that the amount of work done by Insertion Sort, in the worst case, is roughly indicated by:

$$f(n) = \frac{n^2 - n}{2}$$

More specifically, it was: $c_1 \, {}^{n^2}/_2 + c_2 \frac{n}{2} + c_3$

We'll rewrite this as: $an^2 + bn + c$, for some constants a, b, and c.

Order of growth

We'll now make a another simplifying abstraction: are we interested in the **rate of growth**, or the **order of growth**?

Considering our function, $an^2 + bn + c$, the lower order-terms of bn and c are relatively insignificant for large n.

We therefore consider only the leading term, an^2

Insertion sort then has a worst-case running time of: $\Theta(n^2)$. This is pronounced "theta n-squared".

Asymptotic efficiency

Once we consider input sizes (n) large enough to make only the order of growth of the running time relevant, we are studying the **asymptotic efficiency** of algorithms.

Usually, an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.

Efficiency?

Why should we care which is more efficient? While computing resources are growing, they are not infinite.

Efficiency can mean the survival of a company.

For example, a sort algorithm taking $O(n^2)$ on a database of millions of records may take several days to run, whereas one of measure $O(n \cdot \log n)$ may take only a few minutes.

Asymptotic notation

Because asymptotic notation applies to **functions**, what we wrote as $\Theta(n^2)$, was the function $an^2 + bn + c$.

Typically, we will use asymptotic notation to characterize the running time of an algorithm.

Domain of asymptotic notation

The asymptotic running time of an algorithm is defined in terms of **functions**, whose domains are the set of **natural numbers** $\mathbb{N} = \{1,2,3, \dots\}$.

By natural numbers, we mean the set of counting numbers - whole positive integers – that are between 0 and ∞.

We're dealing with *whole positive numbers*, not floating point.

Think about it, we can't have "1/3rd of a CPU instruction cycle".

Defining $\boldsymbol{\Theta}$-notation (Big-theta)

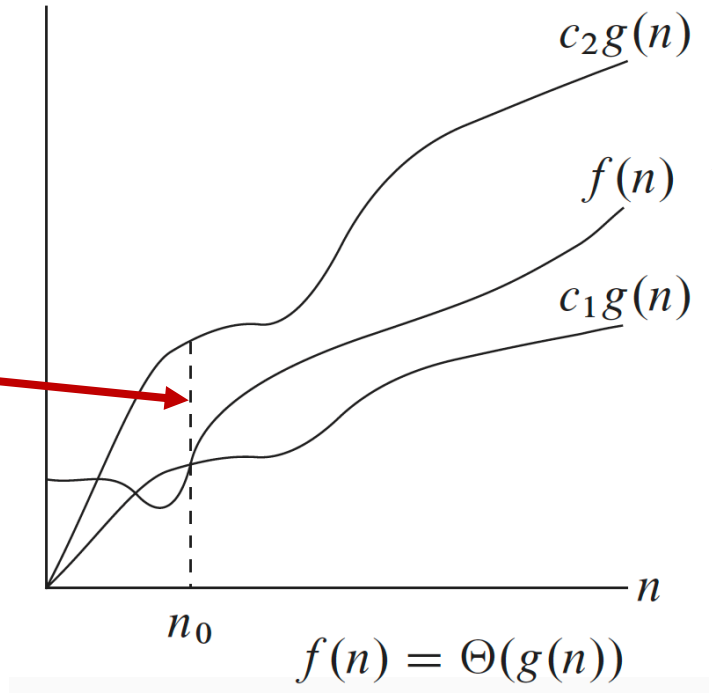
For a given function, $g(n)$, we denote by $\Theta\big(g(n)\big)$ the **set of functions***:

$\Theta\big(g(n)\big)$ = {f(n): there exists positive constants, c_1, c_2, and n_0, such that: $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_n \cdot g(n)$ for all $n \geq n_0$}

* By set of functions, we mean *many functions,* not just one function

Θ-notation

What this means is that as n gets large enough, the running time is **at least $c_1g(n)$**, and **at most $c_2g(n)$:**



But only when $n > n_0$

Here's our function *f(n)*, sandwiched between $c_1g(n)$ and $c_2g(n)$

A set of functions

As $\Theta\big(g(n)\big)$ is a **set**, the relation could also be written using set notation: $\mathrm{f}(n) \in \Theta\big(g(n)\big)$.

The book, and therefore we, will write "$\mathrm{f}(n) = \Theta\big(g(n)\big)$" to express this meaning.

Note: The use of "=" here does not mean equality. It's confusing. How can one thing (LHS) equal many things (RHS)? It doesn't, and this is *abusing equality*, as the book notes. But it's useful to do so, so just be aware of it ☺

Bounding

We now say that g(n) is an **asymptotically tight bound** for f(n).

It also requires that $\mathrm{f}(n) \in \Theta\big(g(n)\big)$ be asymptotically nonnegative.

This means that f(n) be nonnegative whenever n is sufficiently large.

Discussion

Given that we're talking about efficiency, do we care that much about lower bounds?

Big-O notation

The Θ-notation asymptotically bounds a function from above and below.

When we have only an **asymptotic upper bound**, we use O-notation.

We call this “big-oh” notation.

Defining O-notation (Big-oh)

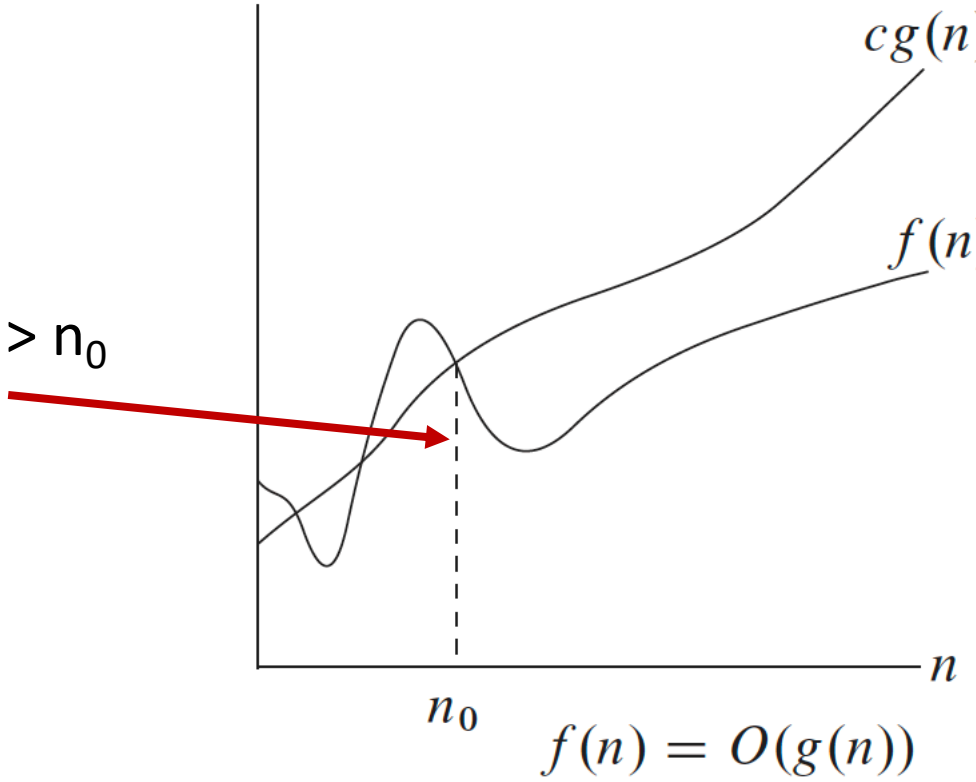
For a given function, $g(n)$, we denote by $O\big(g(n)\big)$ the **set of functions**:

$O\big(g(n)\big) =$ {f(n): there exists positive constants, c and n_0, such that: $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$}

Note that c and n_0 are positive, so $c \geq 1$ and $n_0 \geq 1$. Our definition does not allow c = 0 nor does it allow n0 = 0.

O-notation

What this means is that as n gets large enough, the running time is **at most cg(n):**



But only when n > n_0

Here's our function *f(n)*, which is **upper-bounded** by *cg(n)*.

O's relation to $\boldsymbol{\Theta}$

Given that f(n) = $\Theta\big(g(n)\big)$, this implies that f(n) = O(g(n)), since $\boldsymbol{\Theta}$-notation is *stronger* than O-notation.

We can formally write this relation using set notation: $\Theta\big(g(n)\big) \subseteq O\big(g(n)\big)$.

That is, $\Theta\big(g(n)\big)$ is a subset of $O\big(g(n)\big)$

Therefore, any function f(n) that's within $\Theta\big(g(n)\big)$ must also be in $O\big(g(n)\big)$.

Example

Lets now show that $6n = O(n^2)$

Note, it is not necessary to find the smallest c and n_0, any values will do.

Big-O proof for insertion sort

Consider insertion sort again, which has time complexity: $f(n) = n(n-1)/2$. We now show that $n(n-1)/2 = O(n^2)$

Proof: Is it possible to find c and n_0 such that:

$$n(n-1)/2 \leq c \cdot n^2 \text{ for all } n \geq n_0$$

Step 1: Choose c, n_0: Let c = 1 and n_0 = 1

Step 2: Show that $n(n-1)/2 = 1 \cdot n^2$ for n ≥ 1. If n ≥ 1 then n ≠ 0 and so

$$n(n-1)/2 = (n^2 - n)/2 = n^2/2 - n/2 \leq 1 \cdot n^2 = n^2$$

Note, since $n(n-1)/2$ is actually < n^2, it is also ≤ n^2

Θ

It's nice to be able to say that something is *no worse* than something else.

But what about saying it is *no better* either?

It's straightforward. We've already said that $\mathrm{f}(n) = \Theta\big(g(n)\big)$ and that $\mathrm{f}(n) = O\big(g(n)\big)$.

If *f* = *O(g)* says that *f* is no worse than *g,* then it is also saying that *g* is no better than *f*.

Θ

Suppose it is true that $f = O(g)$ and also that $g = O(f)$.

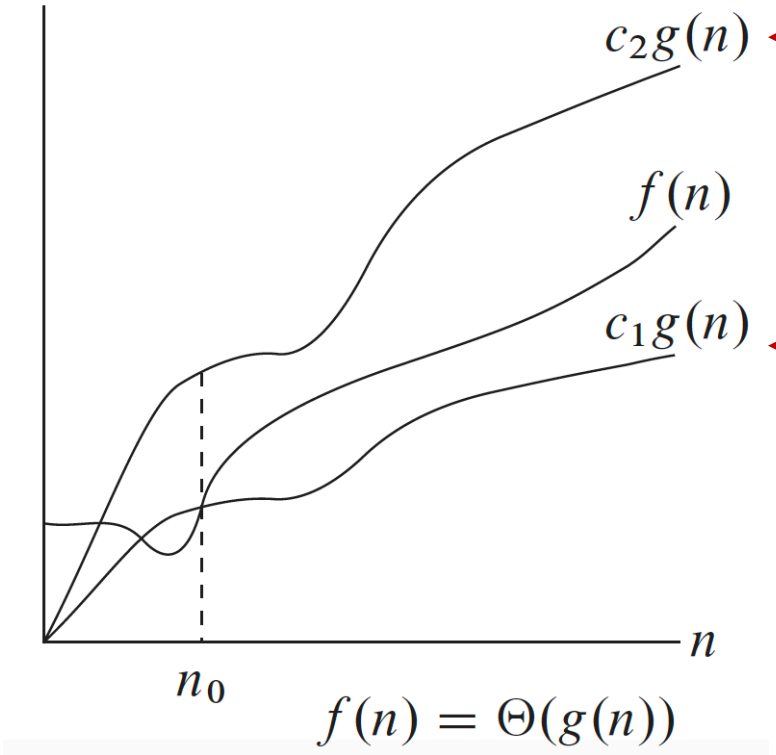
This says that f is no worse than g, and f is no better than g. In other words, f scales up about as well as g, so in terms of efficiency, f is **equivalent** to g.

When $f = O(g)$ and $g = O(f)$, then we may write $f = \Theta(g)$.

In English, it says "f is equivalent to g in efficiency."

Visualise it

Remember, as we've said $f = \Theta(g)$, this means:



f(n) is *no* worse than *g(n)*

f(n) is *no better* than *g(n)*

Θ proof for insertion sort

We've already shown that $n(n-1)/2 = O(n^2)$, so now just show that $n^2 = O(n(n-1)/2)$.

If we choose c = 3 and $n_0 = 3$, then:

$$n^2 \leq c \cdot n(n-1)/2 \text{ for all } n \geq n_0$$

Because:

$$\frac{3n(n-1)}{2} = \frac{3}{2}n^2 - \frac{3}{2}n = n^2 + \frac{n^2}{2} - \frac{3n}{2} = n^2 + \frac{(n^2-3n)}{2} \geq n^2$$

since $n \geq 3$ so that $\frac{(n^2-3n)}{2} \geq 0$

And now, since we have $\frac{n(n-1)}{2} = O(n^2)$ and $n^2 = \frac{n(n-1)}{2}$, we have:

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

Comparing functions

What we've shown is that the relational properties of real numbers also apply to asymptotic comparisons.

Reflexivity:

$$\mathrm{f}(n) = \Theta\big(\mathrm{f}(n)\big)$$

$$\mathrm{f}(n) = O\big(\mathrm{f}(n)\big)$$

This mean: f(n) is related to $\boldsymbol{\Theta}$(f(n))

Symmetry:

$\mathrm{f}(n) = \Theta\big(\mathrm{g}(n)\big)$ if and only if $\mathrm{g}(n) = \Theta\big(\mathrm{f}(n)\big)$

This means: If f(n) is related to $\boldsymbol{\Theta}$(g(n)), then $\boldsymbol{\Theta}$(f(n)) is related to g(n)

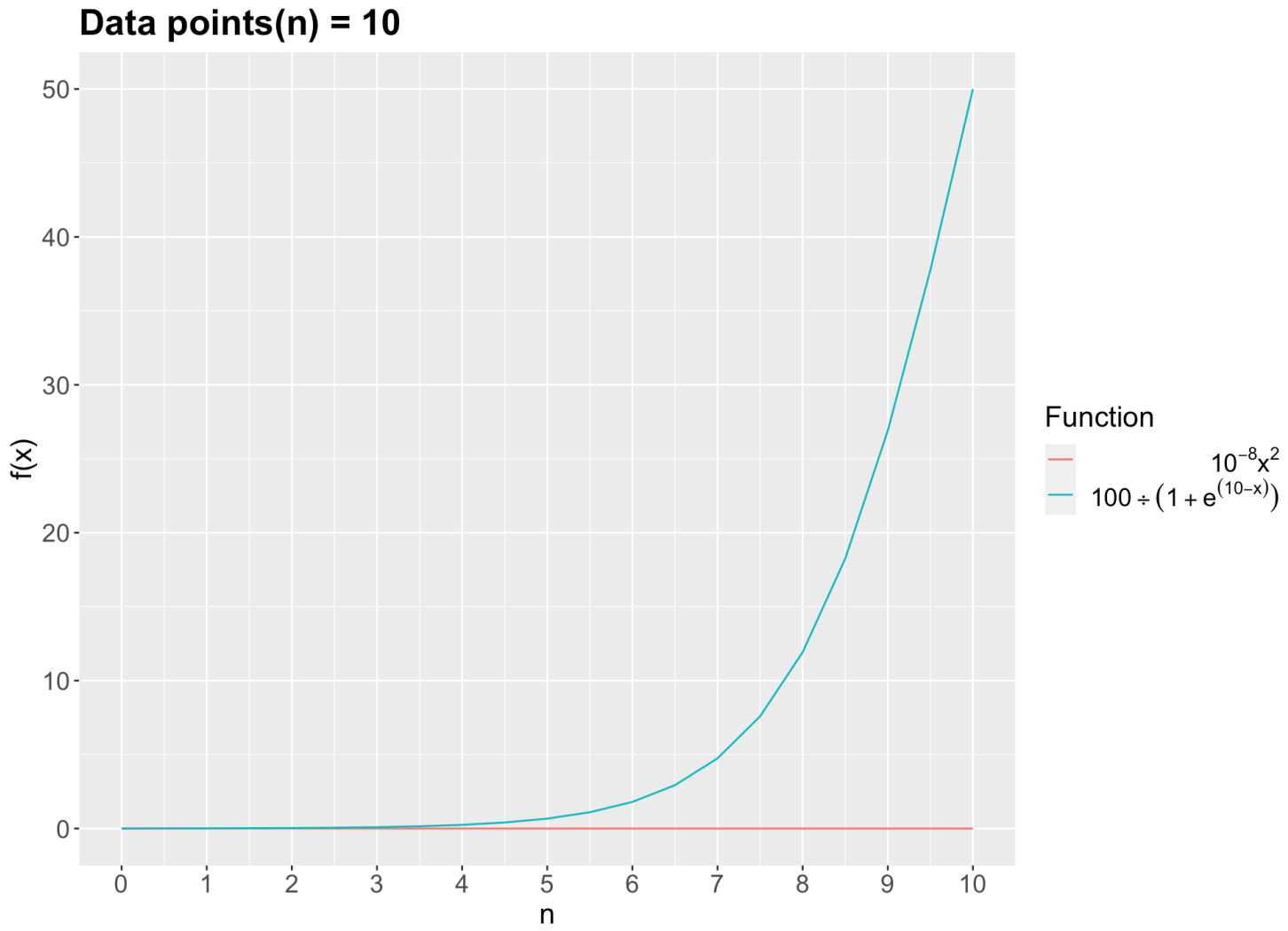
Suggested reading

Asymptotic notation is discussed in section 3.1 of the textbook.

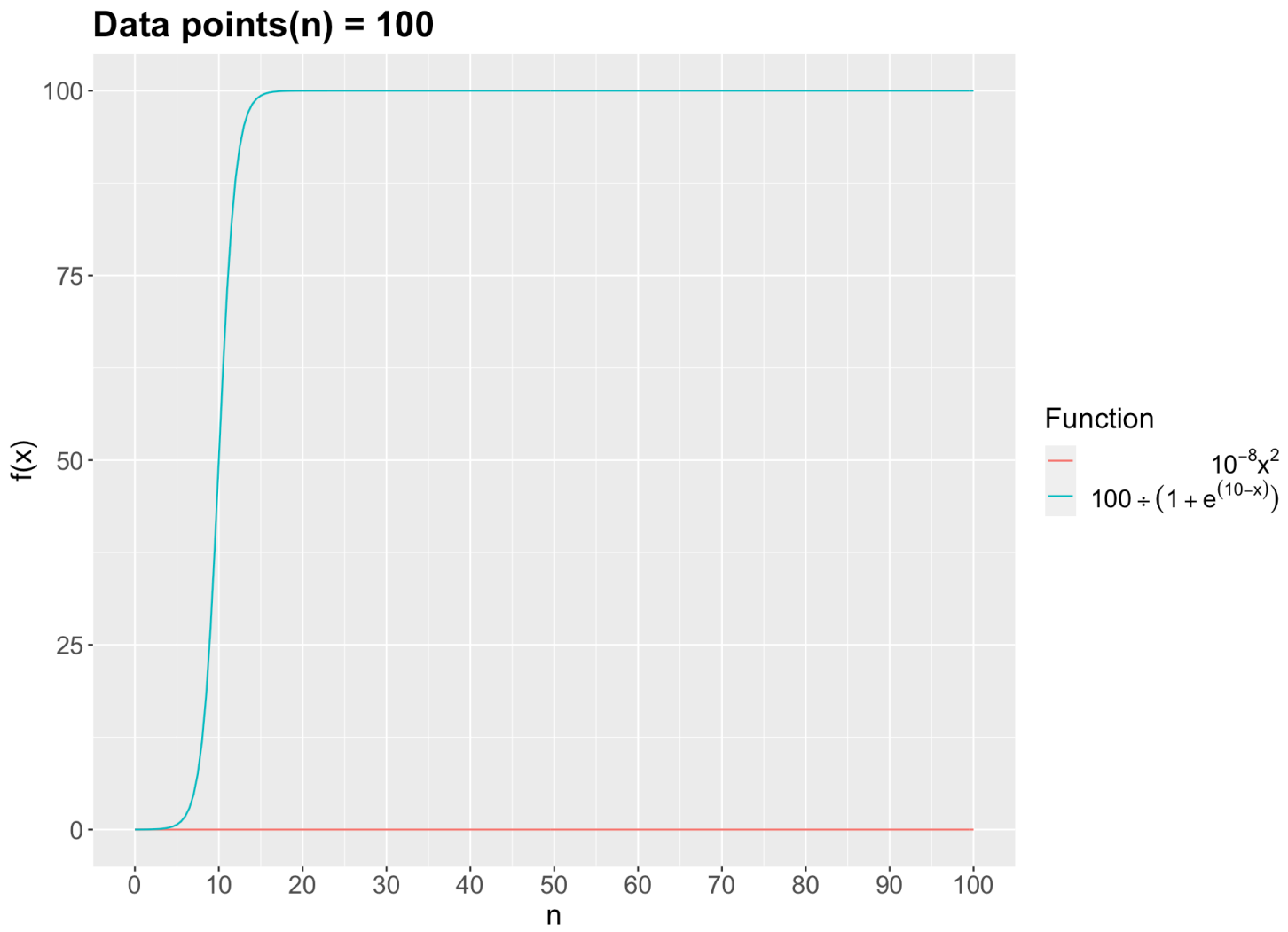
Insertion sort is analysed in section 2.2.

Solutions

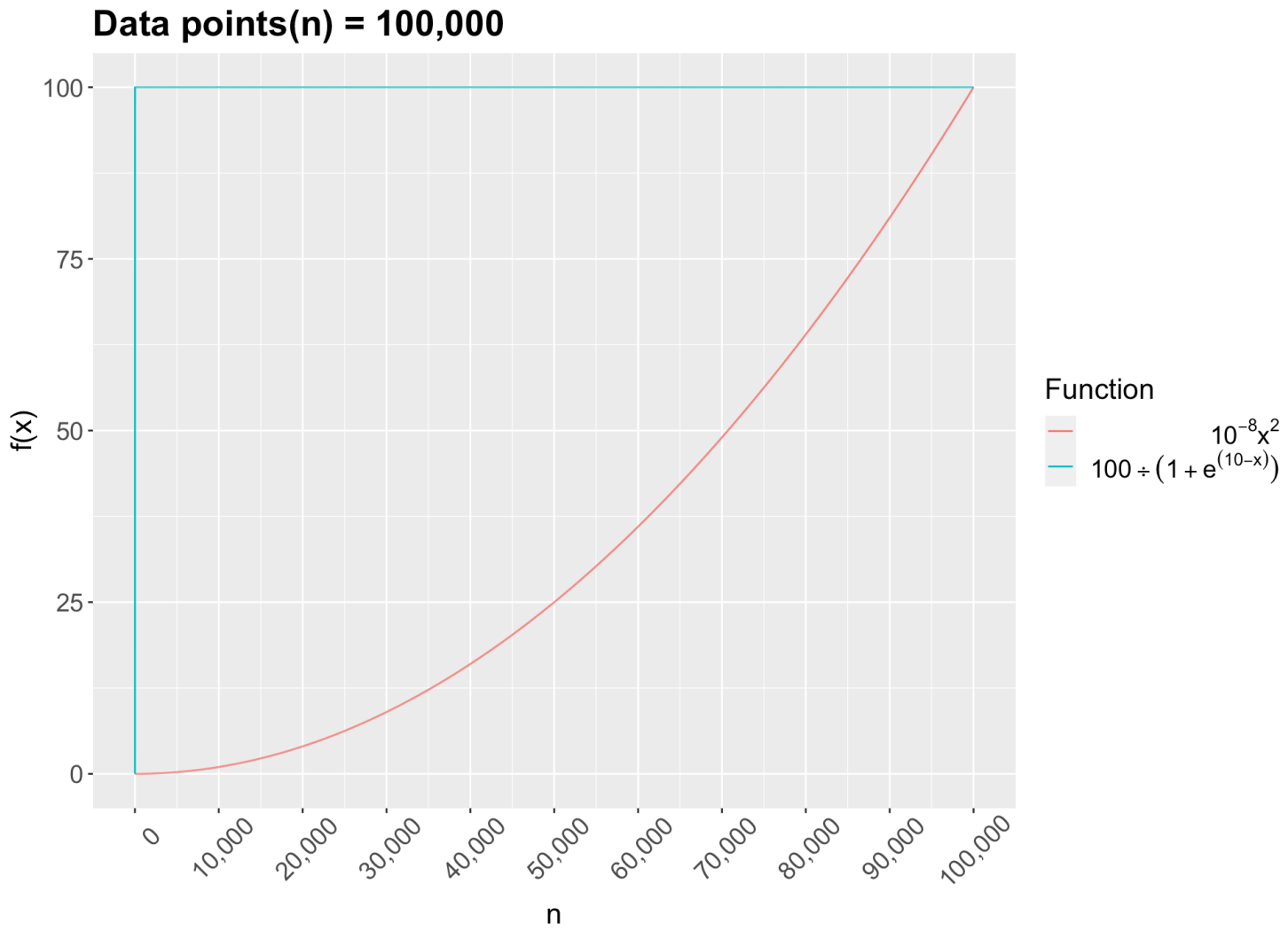
Are plots enough?



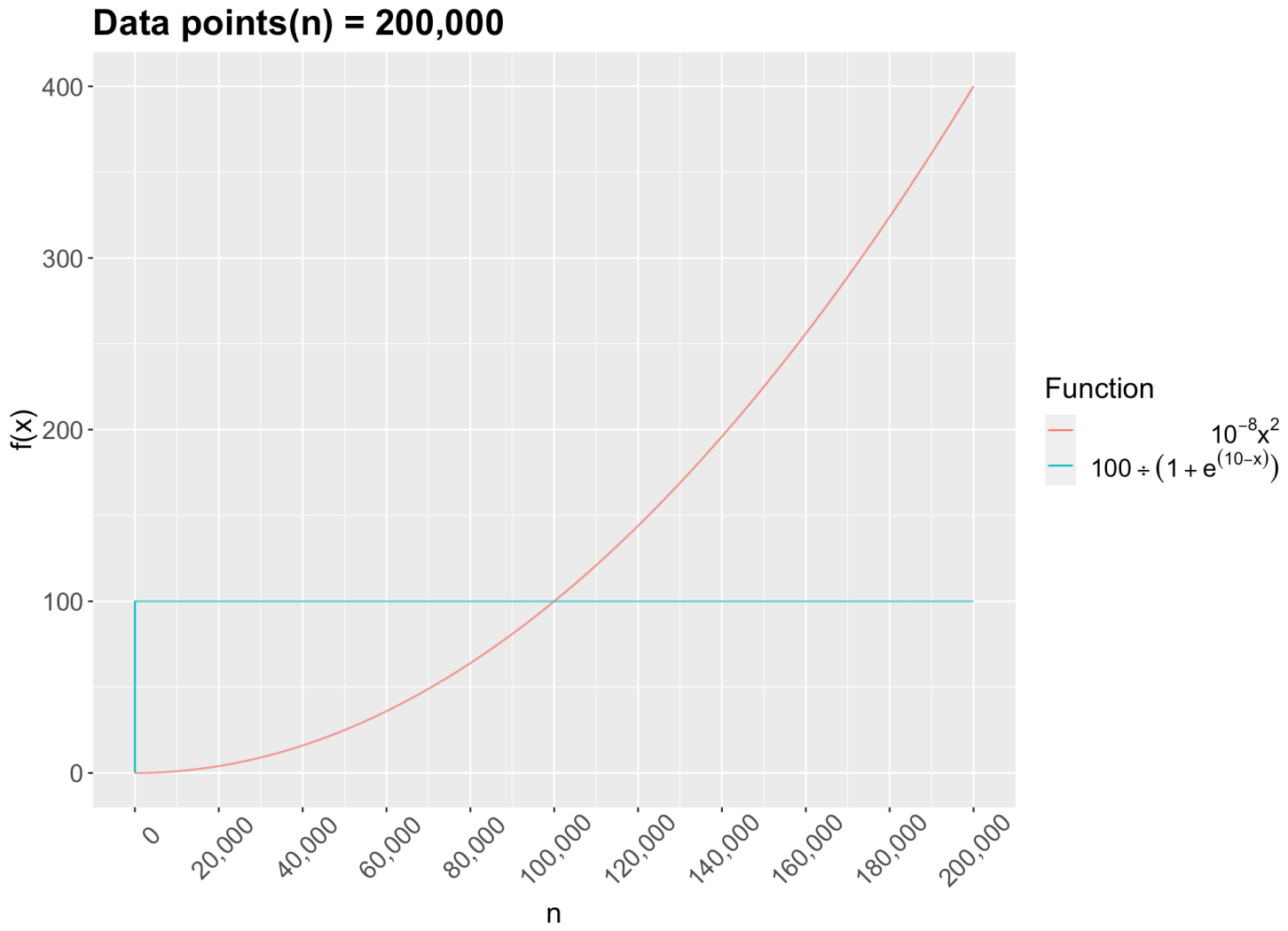
Are plots enough?



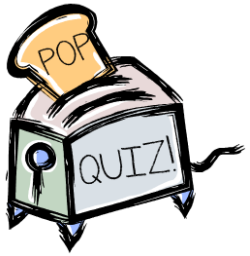
Are plots enough?



Are plots enough?



Pop quiz 2



Are plots enough?

Answers

Yes

No ✓

Otherwise, today's lecture would've been called "*Plots n' stuff*"

Image attributions

Disclaimer: Images and attribution text provided by PowerPoint search. The author has no connection with, nor endorses, the attributed parties and/or websites listed above.