Complexity:

Time and space behaviour

Chapter 20 of the textbook

Evaluation

How fast is my code?

In ghci we can test this by writing

```
> :set +s
```

We can undo this command with

```
> :unset +s
```

Limitations with timing code

Timing your code can be a great idea! But it has some limitations.

- This is testing, and testing is always limited
    - What if there is some ‘corner case’ I didn’t test where my code runs slower?
- Particular to the computer I’m working on
- (Unless you’re very careful) sensitive to other things running on my computer which could slow things down unexpectedly

What would a principled, reliable, portable, approach to describing the speed of code look like?

Algorithmic complexity

How many steps does it take to execute an algorithm given an input of size n?

Complexity of functions

Consider:

```
    f n = 2 × n² + 4 × n + 13
```

This function has three components:

- a constant, 13
- a term 4 × n
- a term 2 × n^2

As the value of n increases:

- the constant component is unchanged
- the 4 × n component grows linearly
- the 2 × n^2 component grows quadratically

Complexity of functions

Input	Quadratic part	Linear part	Constant part
1	11%	21%	68%
10	79%	16%	5%
100	98%	2%	0.06%
1,000	99.8%	0.2%	0.0006%
10,000	99.98%	0.02%	0.000006%
100,000	99.998%	0.002%	0.00000006%
1,000,000	99.9998%	0.0002%	0.0000000006%

Domination

In the function

```
    f n = 2 × n² + 4 × n + 13
```

The quadratic part **dominates** the other parts: for big inputs the other parts hardly matter at all.

Moreover the 2 attached to the quadratic part is also not very significant: the domination would happen for any constant multiplier

So if we are interested in big inputs, the $\mathbf{n^2}$ part is the most significant information

Complexity of functions: “big-Oh” **O**

For large n, the quadratic term **dominates** in f.

So, we say that f is of “**order** n^2” and we write **O**(n^2)

A function `f::Integer->Integer` is **O(g)** if there are positive integers m and d such that for all n≥m:

```
    f n ≤ d × (g n)
```

i.e., that f is **bounded above** by g: for sufficiently large n the value of f is no larger than a multiple of the function g.

e.g., f from the previous slide is **O**(n^2), since for n≥1:

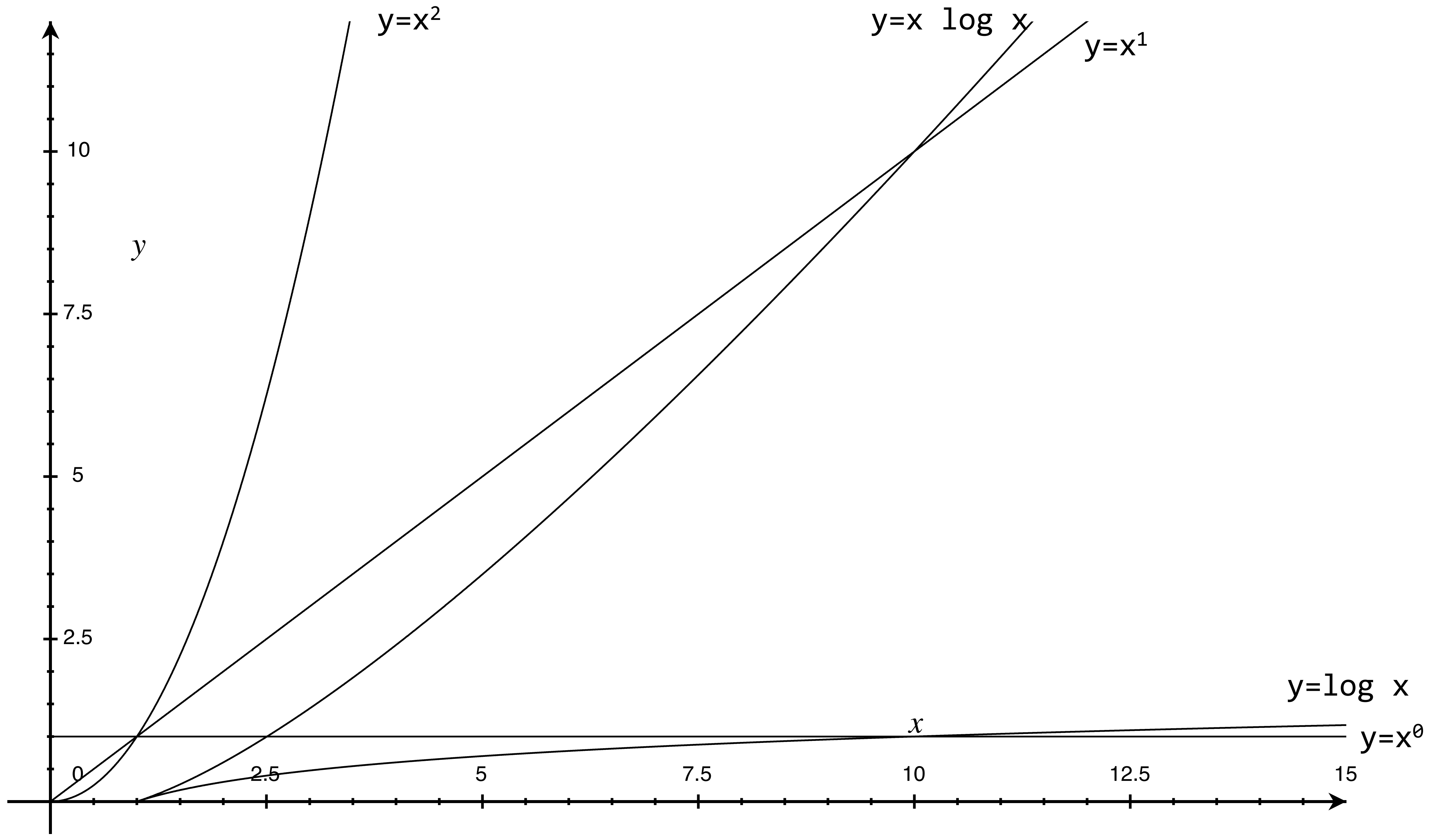
$$2 \times n^2 + 4 \times n + 13 \leq 2 \times n^2 + 4 \times n^2 + 13 \times n^2 \leq 19 \times n^2$$

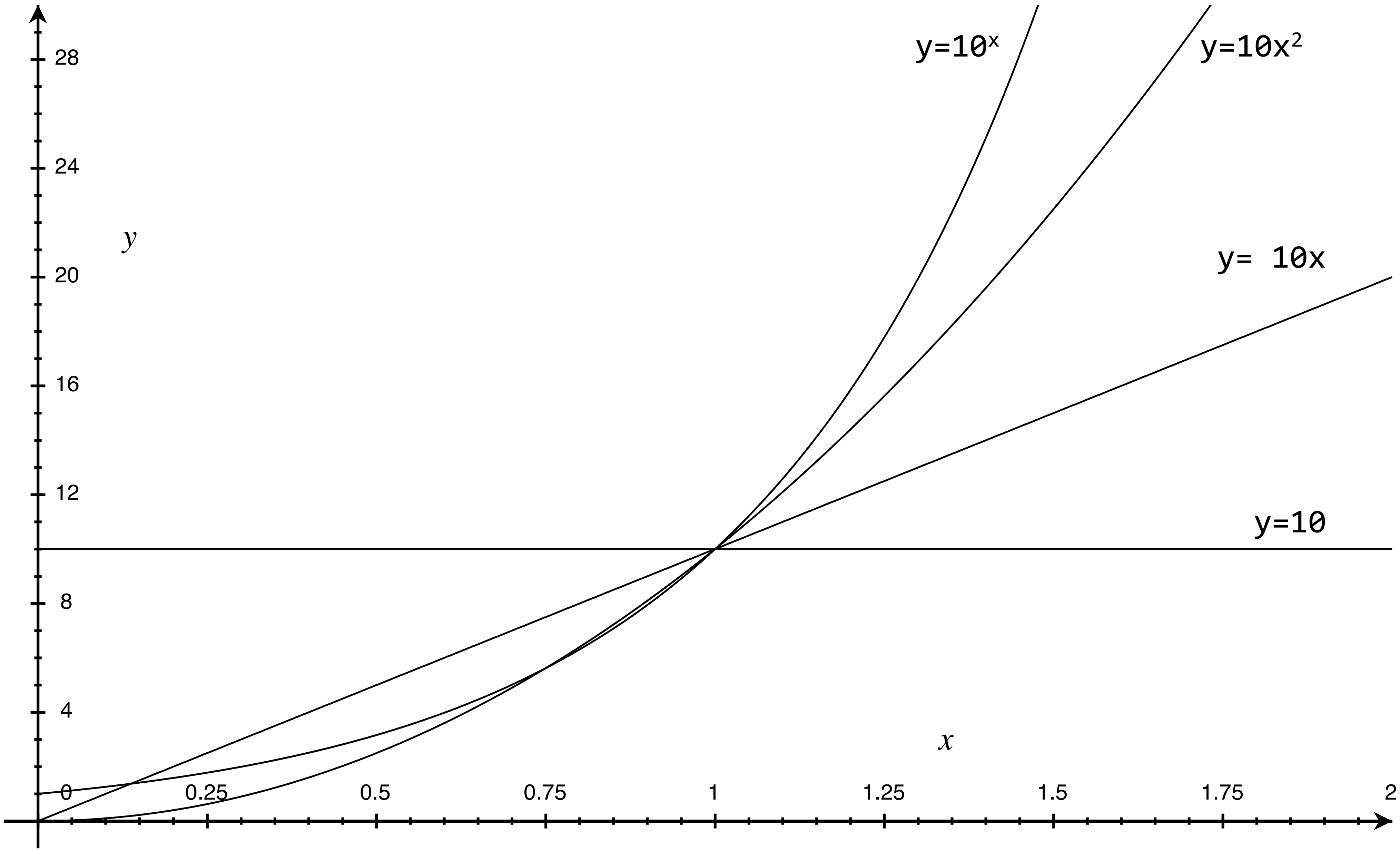
Tight bounds

By the definition of the previous slide, f is also in **O**(n^3).

When we ask the big-Oh class of a function, unless otherwise stated we want a **tight bound** - the slowest growing function that works for f

- Technical: f is in **O**(n^2), but n^2 is also in **O**`(f)`. This does not hold of **O**(n^3)


y=x²
y=x log x
y=x¹
y=log x
y=x⁰
y
x
10
7.5
5
2.5
0
2.5
5
7.5
10
12.5
15



y=10ˣ
y=10x²
y= 10x
y=10
y
x
28
24
20
16
12
8
4
0
0.25
0.5
0.75
1
1.25
1.5
1.75
2

Significance to computing

Suppose you measure the speed of your program at various output sizes n, and determine that its speed is expressed by the function

$$f\ n = 2 \times n^2 + 4 \times n + 13$$

The precise constants, 2, 4, and 13, almost certainly depend on your computer, and would differ from mine.

But the big O class – $\mathbf{O}(n^2)$ – of your code is very unlikely to differ between computers.

Therefore communicating the big O class is the most relevant and important piece of information you could give me.

Best vs Average vs Worst case

Sometimes different inputs of the same size perform very differently.

Finding a specified element in a list might be very fast if it is at the front of a list, but slow if it is at the end, or not in the list at all

Therefore we may have different functions for the best case, the worst case, and (if we have some reasonable distribution) the average case

Convention – unless otherwise stated, big O notation refers to the worst case.

Space efficiency

Time is not the only relevant dimension to measure performance on.

If your code wastes **space** – working memory – then it is more likely to run out of space and crash, and will also run somewhat slower

Space can also measured (e.g. in bytes) and expressed as a function of the input size

Again, big O notation is the right tool for expressing this function