COMP2054-ACE: Introduction to big-Oh

ADE-Lec02b

Lecturer: Andrew Parkes

Email: andrew.parkes@Nottingham.ac.uk

from http://www.cs.nott.ac.uk/~pszajp/

Recap
Aim: **Classification of Functions**

- In computer science, we often need a way to group together **functions** by their scaling behaviour, and the classification should
  - Remove unnecessary details
  - Be (relatively) quick and easy
  - Be able to deal with ‘weird’ functions that can happen for runtimes
  - Still be mathematically well-defined
- Experience of CS is that this is best done by the “big-Oh notation and family”

Recap: Big-Oh Notation: Definition***

Definition: Given positive functions $f(n)$ and $g(n)$, then we say that

$$f(n) \text{ is } O(\, g(n) \,)$$

if and only if there exist positive constants c and n_0 such that

$$f(n) \leq c\, g(n) \qquad \text{for all } n \geq n_0$$

THIS DEFINITION IS VITAL – PLEASE QUESTION, LEARN AND UNDERSTAND ALL PARTS OF IT.

RECAP EXERCISE

$f(n)$ is $O(\,g(n)\,)$ iff **exist** $c,\ n_0$ s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- It is easy to show directly from the definition that the function:

  $f(n) = 1$ is $O(n)$

  I.e. we have both
  1 is O(n) - "strange but true"
  1 is O(1) - "natural"

Nothing in the definition forces a choice of a "best" or "most useful" g(n).

EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \le c\ g(n)$ **for all** $n \ge n_0$

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \le c\,g(n)$ **for all** $n \ge n_0$

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is its big-Oh behaviour?

ANS: It is O(n).

Proof: if $n \ge 1$ then $f(n) \le n$

hence can take $c=1$ $n_0=1$

EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \leq c\ g(n)$ **for all** $n \geq n_0$

- Consider the function:

  $f(n) = n$ if n is even

  1 if n is odd

What is its big-Oh behaviour?

ANS: It is O(n).

Proof: if $n \geq 1$ then $f(n) \leq n$

hence can take $c=1$ $n_0=1$

Note: despite the two cases, "even"/"odd", the definition requires to provide <u>one</u> c and <u>one</u> n0.

EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \le c\ g(n)$ **for all** $n \ge n_0$

- Consider the function:

  f(n) = n if n is even

  1 if n is odd

Is it O(1) ?

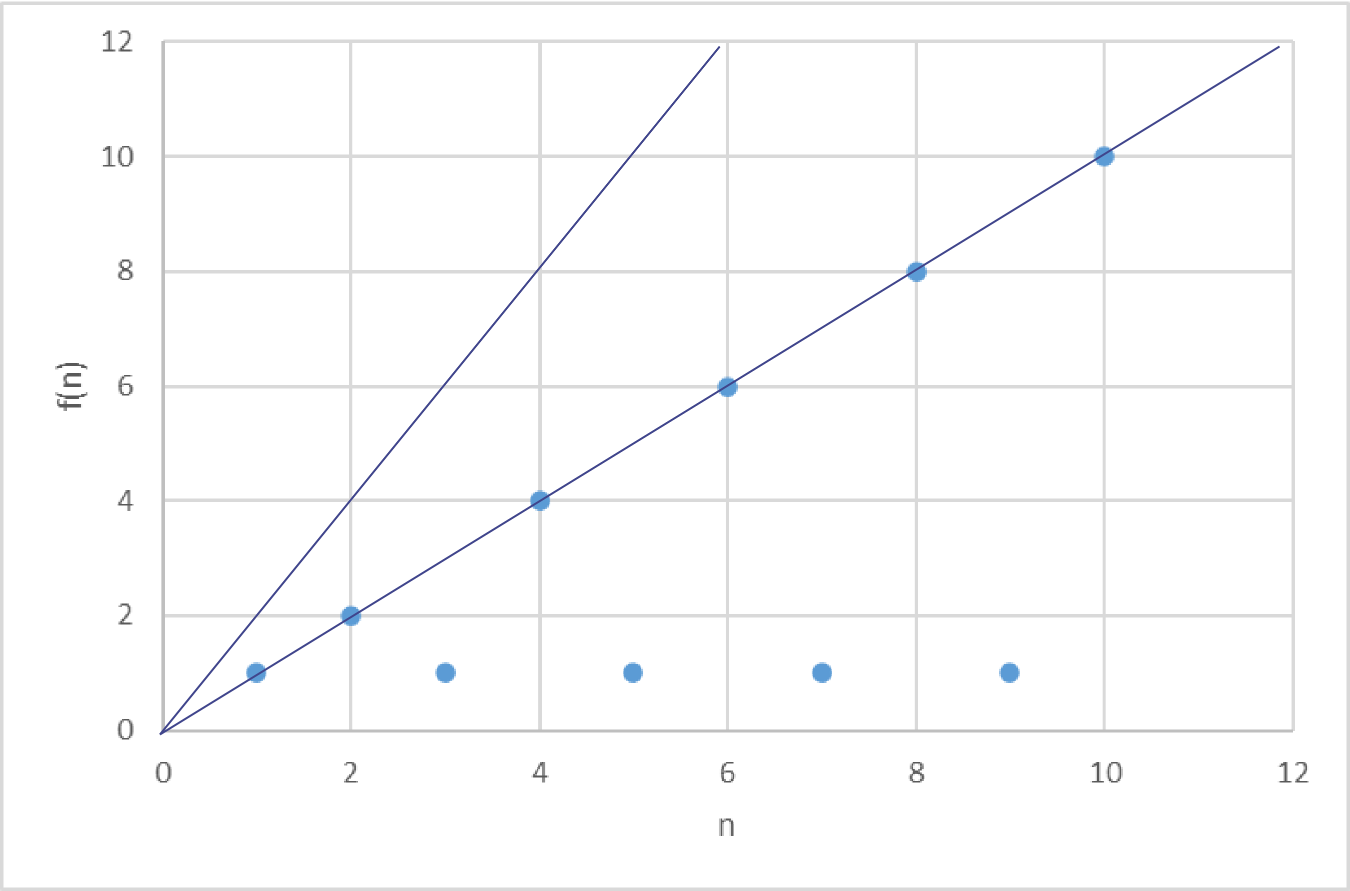
ANS: No, as we would need

Exists c,n0. n ≥ c 1 forall n >= n0

And this fails for the cases n is even.

Big-Oh Graphically

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \leq c\ g(n)$ **for all** $n \geq n_0$



EXERCISE

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is the limit of the ratio $f(n)/n$ as n tends to ∞ (infinity, i.e. becomes very large)?

EXERCISE (ans)

- Consider the function:

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{if } n \text{ is odd} \end{cases}$$

What is the limit of the ratio $f(n)/n$?

ANS: It does not have a limiting ratio !

$$f(n)/n = \begin{cases} 1 \text{ if } n \text{ is even, limit is } 1 \\ 1/n \text{ if } n \text{ is odd, limit is } 0 \end{cases}$$

Shows how “big-Oh” handles weird functions and is different from limits.

Ratios vs. big-Oh

- $f(n)$ can be $O(g(n))$ even if the ratio $f(n)/g(n)$ does not exist
- Hence, big-Oh can be used in situations that ratios cannot
- The possibility of 'weird functions' means that big-Oh is more suitable than ratios for doing analysis of efficiency of programs

EXERCISE

$f(n)$ is $O(\, g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

  f2(n) = n if n is even

  4 if n is odd

What is its big-Oh behaviour?

EXERCISE

$f(n)$ is $O(\, g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- Consider the function:

  $f2(n) = n$ if n is even

  4 if n is odd

What is its big-Oh behaviour?

ANS: It is O(n).

Proof: if $n \ge 4$ then $f2(n) \le n$

hence can take $c=1$ $n_0=4$

"other providers of c and n0 are available"

EXERCISE

$f(n)$ is $O(\,g(n)\,)$ iff **exist** c, n_0 s.t.
$f(n) \le c\,g(n)$ **for all** $n \ge n_0$

- Consider the function:

  f2(n) = n if n is even

  4 if n is odd

What is its big-Oh behaviour?

ANS: It is O(n).

Not a (full) proof “from the definition”:

for even case: pick c=1 n0=1

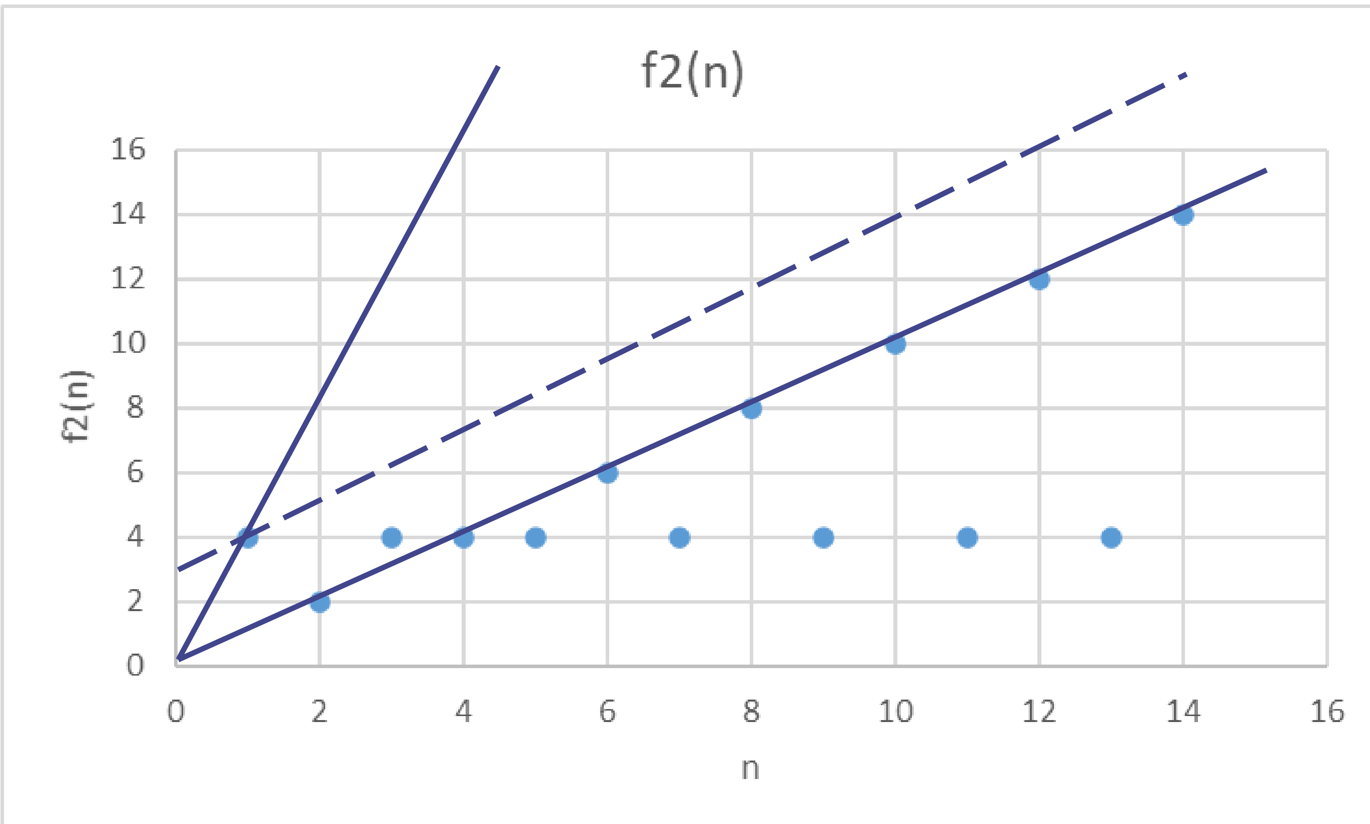
for odd case: pick c=4 n0=1.

(In lean: your proof would fail)

EXERCISE

$f(n)$ is $O(\ g(n)\ )$ iff **exist** c, n_0 s.t.
$f(n) \le c\, g(n)$ **for all** $n \ge n_0$

- "Messy at small n" ; fine past n=4



Exercises (offline)

Defn: Given functions $f(n)$ and $g(n)$, then we say that

$$f(n) \text{ is } O(\ g(n)\ )$$

if and only if there exist positive constants c and n_0 such that $f(n) \le c\, g(n)$ for all $n \ge n_0$

- From the definition, show that
  - (3n-6) is O((4n+5))
  - (3n-6) is O(n)
  - (4n+5) is O(n)
- Notes
  - The first shows that g(n) does not need to be "nice"
  - The last two 'suppress details' as desired

Comment on some details:

- f(n) = (3n-6) is not always positive, but only is negative on a finite number of values of n
- Hence, to be more strict:
  - Could instead consider f(n) = max(0,3n-6) is really meant
  - Or could just require that n_0 is taken large enough that f(n) is positive: that is require

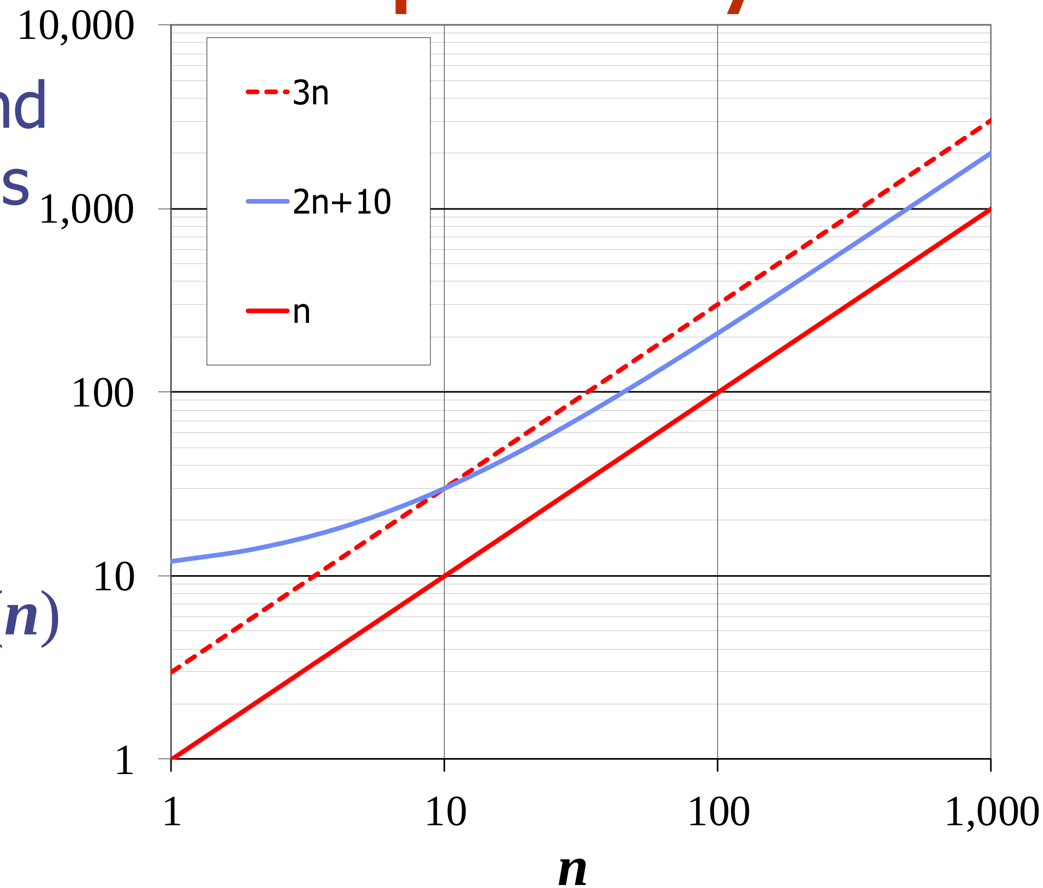
$$\forall\, n \geq n_0. \qquad 0 \leq f(n) \leq c\, g(n)$$

Either of these capture the intent that “f grows no faster than g at large enough n”

Big-Oh Notation: Graphically

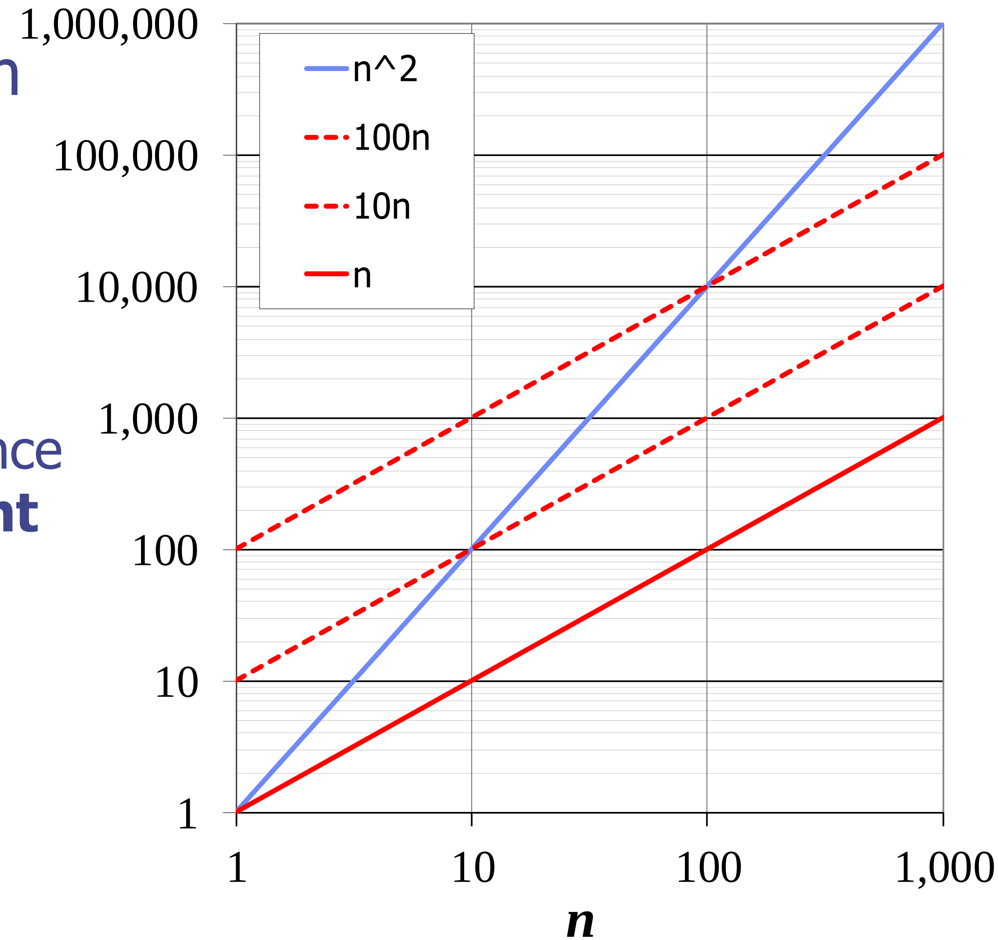
- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$
- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c **must be a constant**
- Can also see this graphically:



Big Oh as a binary relation

- "f is O(g)" can be regarded as a binary relation between two functions f and g
  - Could be written: "$O(f, g)$" or "$f \; O \; g$"
- Generally, binary relations, R, are characterised by potential properties:
  - Reflexive: x R x
  - Symmetric: x R y iff y R x
  - Transitive: x R y & y R z → x R z

Which of these are satisfied by big-Oh?

Big-Oh is Reflexive

- This is trivial:
- For any function

f(n) is O(f(n))

as just take c=1, and use $\forall n. f(n) \leq f(n)$

Big Oh is not symmetric

- 1 is O(n)

But (from an earlier example)

- n is not O(1)

It only takes one counterexample to show that we cannot say it is symmetric

Big Oh is transitive

- Given "f is O(g)" and "g is O(h)", then can we show "f is O(h)" ?
  - E.g.
    "1 is O(n)" and "n is O(n^2)"
    forces
    "1 is O(n^2)" ?
- As usual, start by writing down what we know from the definitions:

Big Oh is transitive

- Given "f is O(g)" and "g is O(h)" then can we show "f is O(h) ?
- Exercise: Start by writing down what we know from the definitions:

  exists c1 n1 c2 n2 s.t.

  $f(n) \le c1\ g(n) \quad \text{forall } n \ge n1$

  $g(n) \le c2\ h(n) \quad \text{forall } n \ge n2$

  How do we get to "f is O(h)"?

Big Oh is transitive

- Given "f is O(g)" and "g is O(h)" then can we show "f is O(h)" ?
- exists c1 n1 c2 n2 s.t.
  f(n) <= c1 g(n) forall n >= n1
  g(n) <= c2 h(n) forall n >= n2
- Mult 2nd inequality by c1 (uses c1 >0)
  c1 g(n) <= c1 c2 h(n) forall n >= n2
- Set n3 = maximum(n1,n2), and combine gives

f(n) <= c1 g(n) <= c1 c2 h(n) forall n >= n3

Then can take c = c1*c2 to complete the proof.

Big-Oh as a set

- Big Oh as a binary relation is reflexive and transitive but not symmetric
  - It behaves like "$\subseteq$", "$\in$" or "$\leq$", not like "$=$"
  - One might say $n \in O(n)$, and $2n+3 \in O(n)$, etc.
- So may help to think of "$O(n)$" as a set of functions, with each function f in the set, $f \in O(n)$, satisfying "f is $O(n)$".
  - Or can say: $\{f\} \subseteq O(n)$
  - So then $O(1) \subseteq O(n)$
    - Any function bounded above by a constant, is also bounded above by a linear function
  - This gives a closer match to hierarchical classification, similar to
    Tom $\in$ Cats, Cats $\subseteq$ Mammals

Big-Oh as a set - notation

May help to think of "O(n)" as a set of functions, with each function f in the set, $f \in O(n)$, satisfying "f is O(n)".

- Many texts use "f = O(n)", but
- I dislike this because usual intuition about "=" would make it natural to say:
  - $1 = O(1)$ and $1 = O(n)$ hence $O(1) = O(n)$ which is WRONG
  - E.g. O(n) contains f(n)=1, but O(1) does not
- Also, we cannot swap the order:
  - $1 = O(1)$ does not mean we can write "$1 = O(1)$" which is meaningless

Big-Oh as a set

- So may help to think of "O(n)" as a set of functions, with each function f in the set, $f \in O(n)$, satisfying "f is O(n)".
- This allows to give sense to expressions such as

$$n^{O(1)}$$

  to mean "n to the power of f(n) with f(n) being O(1)".
- Hence, this included n, n^2, n^3, etc
- Can sometimes be used as way to say "is some power law"
  - (Under "big-Oh as worst case of an algorithm" then $n^{O(1)}$ would make no sense!).

Summary & Expectations

- Know the definition of big-Oh well!
  - Understand why it is appropriate in CS
- Be able to apply it, and prove results on big-Oh of simple functions
- Big Oh as a binary relation is reflexive and transitive but not symmetric
  - It behaves like "$\subseteq$", "$\in$" or "$\leq$", not like "$=$"

Exercises (offline)

- Show $7n-2$ is $O(n)$

- Show $3n^3 + 20n^2 + 5$ is $O(n^3)$

- Show $3 \log n + 5$ is $O(\log n)$