

# Big-*O* Notation

## COMS10018 - Algorithms

Dr Christian Konrad

# Big $O$ Notation

**Definition:**  $O$ -notation (“Big  $O$ ”)

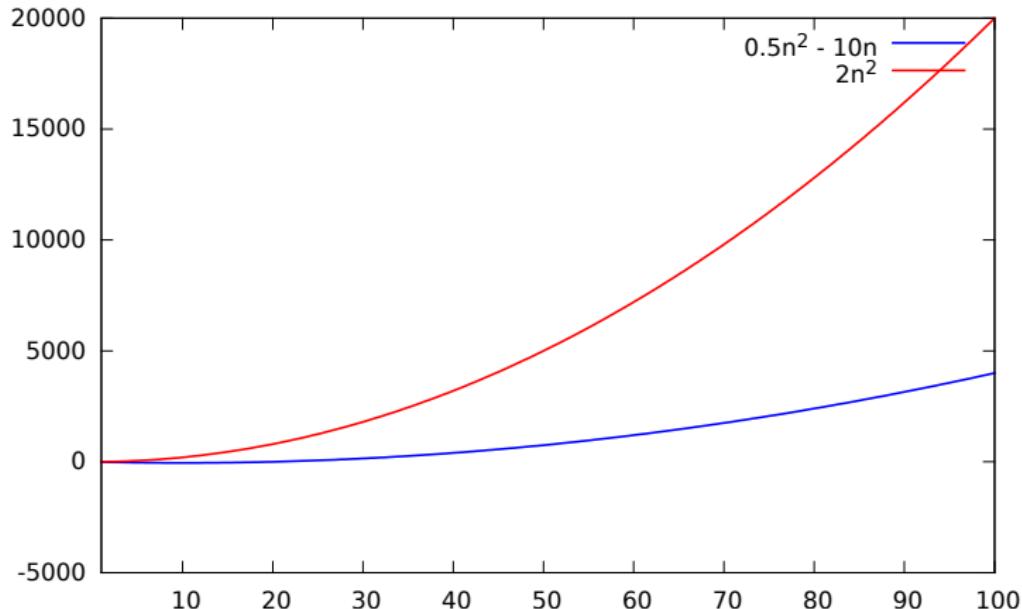
Let  $g(n)$  be a function. Then  $O(g(n))$  is the set of functions:

$$O(g(n)) = \{f(n) : \text{There exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

**Meaning:**  $f(n) \in O(g(n))$  : “ $g$  grows asymptotically at least as fast as  $f$  up to constants”

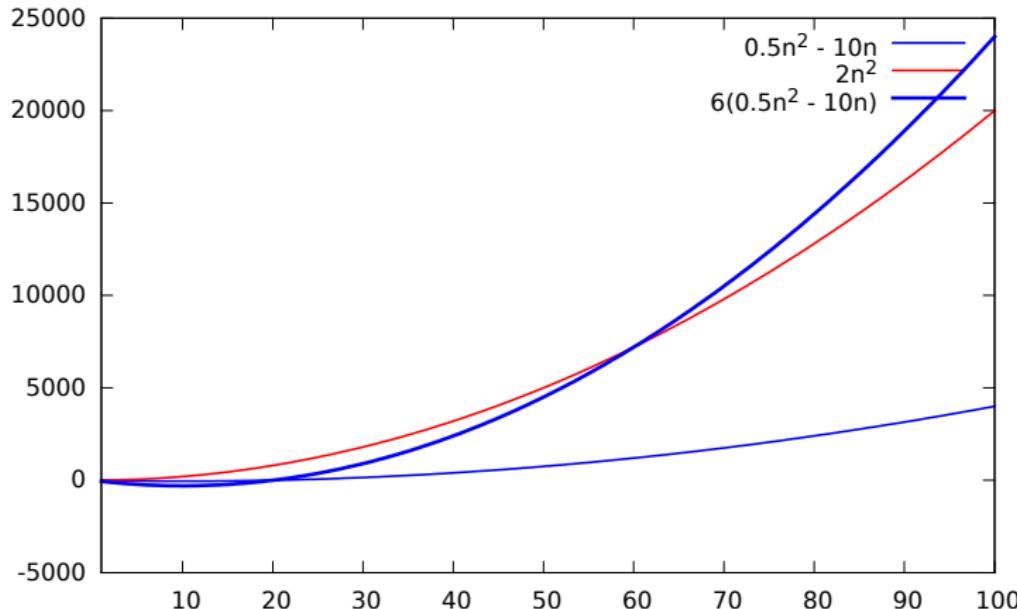
# O-Notation: Example

**Example:**  $f(n) = \frac{1}{2}n^2 - 10n$  and  $g(n) = 2n^2$



# O-Notation: Example

**Example:**  $f(n) = \frac{1}{2}n^2 - 10n$  and  $g(n) = 2n^2$



**Then:**  $g(n) \in O(f(n))$ , since  $6f(n) \geq g(n)$ , for every  $n \geq n_0 = 60$

# More Examples/Exercises

## Recall:

$$O(g(n)) = \{f(n) : \text{There exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

## Exercises:

- $100n \stackrel{?}{\in} O(n)$  Yes, chose  $c = 100, n_0 = 1$
- $0.5n \stackrel{?}{\in} O(n/\log n)$  No: Suppose that such constants  $c$  and  $n_0$  exist. Then, for every  $n \geq n_0$  :

$$0.5n \leq cn/\log n$$

$$\log n \leq 2c$$

$$n \leq 2^{2c}, \text{ a contradiction,}$$

since this does not hold for every  $n > 2^{2c}$ .

# Recipes



**Proving that  $f \in O(g)$ :**

Find constants  $c, n_0$  as in the statement of the definition of Big- $O$ , i.e., such that  $f(n) \leq c \cdot g(n)$ , for all  $n \geq n_0$

**Proving that  $f \notin O(g)$ :**

Proof by contradiction: Assume that constants  $c, n_0$  exist as in the statement of the definition of Big- $O$  and derive a contradiction

# Sum of Two Functions

## Lemma (Sum of Two Functions)

Suppose that  $f, g \in O(h)$ . Then:  $f + g \in O(h)$ .

### Proof.

**To Do:** We need to find constants  $C, N_0$  such that

$$f(n) + g(n) \leq C \cdot h(n), \text{ for every } n \geq N_0.$$

Since  $f \in O(h)$  there exist constants  $c, n_0$  such that

$$f(n) \leq c \cdot h(n), \text{ for every } n \geq n_0.$$

Since  $g \in O(h)$  there exist constants  $c', n'_0$  such that

$$g(n) \leq c' h(n), \text{ for every } n \geq n'_0.$$

Let  $C = c + c'$  and let  $N_0 = \max\{n_0, n'_0\}$ . Then:

$$f(n) + g(n) \leq ch(n) + c'h(n) = C \cdot h(n) \text{ for every } n \geq N_0. \quad \square$$

# Further Properties

## Lemma (Polynomials)

Let  $f(n) = c_0 + c_1 n + c_2 n^2 + c_3 n^3 + \dots + c_k n^k$ , for some integer  $k$  that is independent of  $n$ . Then:  $f(n) \in O(n^k)$ .

**Proof:** Apply statement on last slide  $k = O(1)$  times □

**Attention:** Wrong proof of  $n^2 \in O(n)$ : (this is clearly wrong)

$$\begin{aligned} n^2 &= n + n + \underbrace{n + \dots + n}_{n-2 \text{ times}} = O(n) + O(n) + \underbrace{n + \dots + n}_{n-2 \text{ times}} \\ &= O(n) + \underbrace{n + \dots + n}_{n-2 \text{ times}} = O(n) + O(n) + \underbrace{n + \dots + n}_{n-3 \text{ times}} = \\ &= O(n) + \underbrace{n + \dots + n}_{n-3 \text{ times}} = \dots = O(n). \end{aligned}$$

Application of statement on last slide  $n$  times! (only allowed to apply statement  $O(1)$  times!)

# Runtime of Algorithms

## Tool for the Analysis of Algorithms

- We will express the runtime of algorithms using  $O$ -notation
- This allows us to compare the runtimes of algorithms
- **Important:** Find the slowest growing function  $f$  such that our runtime is in  $O(f)$  (most algorithms have a runtime of  $O(2^n)$ )

## Important Properties for the Analysis of Algorithms

- Composition of instructions:

$$f \in O(h_1), g \in O(h_2) \text{ then } f + g \in O(h_1 + h_2)$$

- Loops: (repetition of instructions)

$$f \in O(h_1), g \in O(h_2) \text{ then } f \cdot g \in O(h_1 \cdot h_2)$$

## Rough incomplete Hierarchy

- Constant time:  $O(1)$  (individual operations)
- Sub-logarithmic time: e.g.,  $O(\log \log n)$
- Logarithmic time:  $O(\log n)$  (FAST-PEAK-FINDING)
- Poly-logarithmic time: e.g.,  $O(\log^2 n), O(\log^{10} n), \dots$
- Linear time:  $O(n)$  (e.g., time to read the input)
- Quadratic time:  $O(n^2)$  (potentially slow on big inputs)
- Polynomial time:  $O(n^c)$  (used to be considered efficient)
- Exponential time:  $O(2^n)$  (works only on very small inputs)
- Super-exponential time: e.g.  $O(2^{2^n})$  (big trouble...)