What is the execution time of my program?

- I can measure the number of seconds for a given input
  - My trusty iPhone 13 takes 5 seconds (😀) to compute the eigenvalues of a web link matrix of size $10^9 \times 10^9$ (PageRank)
  - This is not very useful information (not much can be inferred from it except that I have a fast processor in my phone)
- More interesting is to see how the execution time *depends* on the input size (for *predicting* the time)
  - This is a *function* not a *number*
- Even more interesting is to see how the execution time changes when the input size *increases without bound*
  - This is called *asymptotic analysis*
  - What happens when the web link matrix gets bigger and bigger?

Asymptotic analysis and computational complexity

- What *function* gives the execution time in function of the input size, when the size *increases without bound*?
  - Asymptotic analysis is finding an approximation whose error tends to zero when a parameter tends to infinity
- If we know this function, we can infer many things
  - What is the time for a given input size?
  - What is the maximum input size for a given time?
  - How does the maximum input size change
    if I buy a computer that is 256 times faster?
- Computational complexity is the use of asymptotic analysis to study the execution time and memory use of programs
  - The word « complexity » is used in a different way than in everyday life

Fast-growing functions are bad

- Assume *f(n)* is the time in microseconds for input size *n*
- What is the *time* for a given input size?

Size \ *f(n)*	*n*	*400n*	$2n^2$	n^4	2^n
1	1 μs	400 μs	2 μs	1 μs	2 μs
1000	1000 μs	0,4 s	2 s	11d 14h	5×10^{291} y
1000000	1 s	400 s	23d 4h	3×10^{10} y	5×10^{301020} y

Red = we can't wait that long

- What is the *maximum input size* for a given time?

f(n) \ Time	1 second	1 minute	1 hour
n	1×10^6	6×10^7	3.6×10^9
400n	2500	150 000	9×10^6
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Red = we can't do big problems

Constant factors can be ignored

- How does the maximum input size *m* change if I buy a computer that is 256 times faster?

f(n)	New maximum size
n	$256m$
$400n$	$256m$
$2n^2$	$16m$
n^4	$4m$
2^n	$m+8$

Size is multiplied 😀

Size is incremented 😞

- The exponential function 2^n is *very bad*: size is only incremented, not multiplied (so it never gets very big!)
- The constant factor (n versus $400n$) *does not matter*

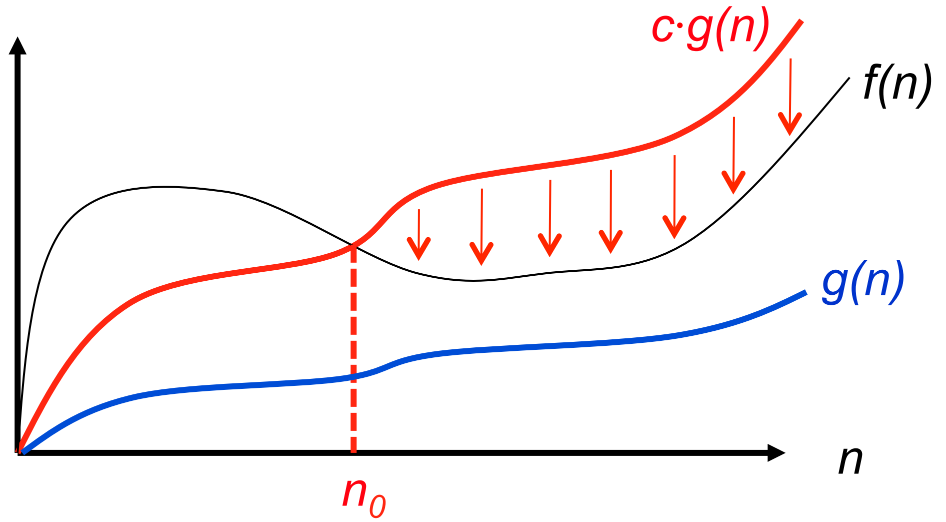
A mathematical concept: big-O notation

- Big-O notation captures the intuitions of "size increases without bound" and "constant factors can be ignored":

  $$f(n) \in O(g(n))$$

  means that $f(n)$ has upper bound $g(n)$ with some constant factor c and given that n is sufficiently large (bigger than some n_0)

$$\boxed{f(n) \in O(g(n)) \quad \textbf{\textit{iff}} \quad \exists c > 0, \exists n_0 \geq 1 \text{ such that } \forall n \geq n_0 : f(n) \leq c \cdot g(n)}$$



Starting from n_0, $f(n)$ is always below $c \cdot g(n)$

Using big-O notation

- $2n+10 \in O(n)$ since $2n+10 \leq 4 \times n$ for $n \geq 5$
- *This is the best upper bound: we can't do better than $g(n)=n$*

- $2n+10 \in O(n^2)$ since $2n+10 \leq 1 \times n^2$ for $n \geq 5$
- *This is not a very good upper bound (see previous example)*

- $2^{100} \in O(1)$ since $2^{100} \leq 2^{100} \times 1$ for $n \geq 1$
- *Constants are constants no matter how large!*

- $3n^2 + 10n \log_{10} n + 125n + 100 \in O(n^2)$

  since $3n^2 + 10n \log_{10} n + 125n + 100 \leq 4 \times n^2$ for $n \geq 148$

  We *keep dominant terms* and *remove constant factors*

- The function *g(n)* is called the *temporal complexity* of the program