# Algorithms, efficiency and O-notation

## algorithm

- finite set of well-defined instructions for accomplishing some task
- will begin in an initial state and terminate in a defined end state.

## algorithm efficiency

- speed: the time it takes for an algorithm to complete
- space: maximum amount of memory used up by the algorithm at any time of its execution

## measuring algorithm efficiency

- do not count the exact number of machine instructions or machine words required
- but estimate the running time and/or space of algorithms

## rules

- running time and space requirement is given for the worst case input of the problem to be solved
- make running time and space requirement depend on the size of the input to the problem

$\Rightarrow$ running time and space requirement are functions $T : \mathbb{N} \to \mathbb{N}$ and $S : \mathbb{N} \to \mathbb{N}$

- $T(n)$ and $S(n)$ express the running time and space requirement for input size $n$

## Example 1

- suppose the running time of an algorithm is $T(n) = 5n^2 + 3n + 72$ seconds, where $n$ is the size of the problem.
- simplification by dropping all constants and lower terms.

$\Rightarrow$ drop $3n$ and 72 (lower-order terms with respect to $n^2$) and constant 5

- running time of the algorithm is $O(n^2)$.

# Algorithms, efficiency and O-notation

- The O-notation is a mathematical notation used to describe the asymptotic behavior of functions.
- It allows us to indicate that we do not care for constants and lower-order terms.
- Its purpose is to characterize a function's behavior for very large inputs in a simple but rigorous way that simplifies the comparison of algorithms concerning their running time and space requirements.
- More precisely, the symbol $O$ is used to describe an asymptotic upper bound for the magnitude of a function in terms of another, usually simpler, function.
- There are also symbols for lower bounds and tight bounds which are not discussed here.

# Algorithms, efficiency and O-notation

**Definition 1**

Suppose $f$ and $g$ are two functions from integers to integers. We say that $f$ is $O(g)$ if and only if there exists some constants $n_0$ and $c > 0$ such that

$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

- Note that the $n_0$ is the minimum problem size for which $f$ is dominated by $g$.
- $c$ is a constant, i.e. it cannot depend on $n$.

## Example 2

Consider the functions $f$ and $g$ defined as follows:

$$\begin{aligned} f(n) &= 6n^4 - 2n^3 + 5 \\ g(n) &= n^4 \end{aligned}$$

Now for $n \geq 1$ the following inequality holds:

$$\begin{aligned} f(n) = 6n^4 - 2n^3 + 5 &\leq 6n^4 + 2n^3 + 5 \\ &\leq 6n^4 + 2n^4 + 5n^4 \\ &\leq 13n^4 \\ &= 13 \cdot g(n) \end{aligned}$$

With $c = 13$ and $n_0 = 1$ we can conclude $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. That is, we have found the constant $c$ and minimum input size $n_0$ as required in Definition 1. So $f$ is $O(g)$.

# Algorithms, efficiency and O-notation

- By $O(g)$ we refer to the set of functions $f$ such that $f$ is $O(g)$.
- That is, $O(g)$ is the set of functions dominated by $g$.
- When using the O-Notation, one does not always have to give explicit names to functions, like $f$ and $g$ as in the example above.
- Instead, one uses polynomials: For example, if we want to refer to the set $O(g)$ with $g$ defined by $g(n) = n^2$ we simply write $O(n^2)$.
- Then we can, for example, state that the running time of some algorithm is $O(n^2)$.
- This means that the function $f$ describing the dependency of the running time of the algorithm on the input size is $O(n^2)$.
- Table 1 shows some commonly used functions for specifying asymptotic upper bounds.
- Table 2, Table 3 and Figure 1 give more explanations, shows sample values and plots of most of these functions.

Table 1: Some commonly used functions for specifying asymptotic upper bounds.

| notation | name | example |
|---|---|---|
| $O(1)$ | constant | determining cost of an edit operation |
| $O(\log n)$ | logarithmic | finding an element in a sorted array of length $n$ |
| $O(n)$ | linear | determining the identity of two sequences both of length $n$ |
| $O(n \log n)$ | quasilinear | determining an optimal chain of $n$ matches, see Genome Informatics lecture |
| $O(n^2)$ | quadratic | determining the edit distance of two sequences both of length $n$ |
| $O(n^c)$, $c > 1$ | polynomial | find secondary structure of RNA-sequence $s$ with minimum free energy ($n = \|s\|$, $c = 3$, see Genome Informatics lecture) |
| $O(c^n)$ | exponential | recursive counting of all alignments |
| $O(n!)$ | factorial | enumerate all subsets of set of size $n$ |

Table 2: Explanations of commonly used functions for $O$-Notation.

| | |
|---|---|
| $O(1)$ | All instructions of the program are executed at most only a few times, independent of the size of the input. |
| $O(\log n)$ | The program gets slightly slower as $n$ grows. Whenever $n$ doubles, $\log n$ increases by a constant. |
| $O(n)$ | Usually, a small amount of processing is done on each input element. Whenever $n$ doubles, then so does the running time. This is the optimal situation for an algorithm that must process $n$ inputs/produce $n$ outputs. |
| $O(n \log n)$ | This often arises in algorithms dividing the problem into smaller subproblems, solving them independently and then combining solutions. When $n$ doubles the running time more than doubles (but not much more). |
| $O(n^2)$ | This arises when algorithms process pairs of all data items (e.g. double-nested loop). Whenever $n$ doubles, the running time increases fourfold. |
| $O(n^3)$ | This arises when algorithms process triples of all data items (e.g. triple-nested loop). Whenever $n$ doubles, the running time increases eightfold. |
| $O(2^n)$ | Exponential run times arise naturally as solutions to problems using a "brute-force" approach. Whenever $n$ doubles, the running time squares. |

Table 3: Commonly used functions with sample values

| $\log n$ | $\sqrt{n}$ | $n$ | $n \log n$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---:|---:|---:|---:|---:|---:|---:|---:|
| 3 | 3 | 10 | 30 | $10^2$ | $10^3$ | $1.02 \cdot 10^3$ | $3.62 \cdot 10^6$ |
| 6 | 10 | $10^2$ | 600 | $10^4$ | $10^6$ | $1.27 \cdot 10^{30}$ | $\approx 10^{158}$ |
| 9 | 31 | $10^3$ | $9 \cdot 10^3$ | $10^6$ | $10^9$ | $1.07 \cdot 10^{301}$ | $\approx 10^{2568}$ |
| 13 | 100 | $10^4$ | $1.3 \cdot 10^5$ | $10^8$ | $10^{12}$ | very large | very large |
| 16 | 316 | $10^5$ | $1.6 \cdot 10^6$ | $10^{10}$ | $10^{15}$ | very large | very large |
| 19 | 1 000 | $10^6$ | $1.9 \cdot 10^7$ | $10^{12}$ | $10^{18}$ | very large | very large |

Figure 1: Plots of some commonly used functions $f$ for specifying asymptotic upper bounds shown for $n \leq 8$ and $f(n) \leq 9$.