

## Basic Principles of Complexity Analysis

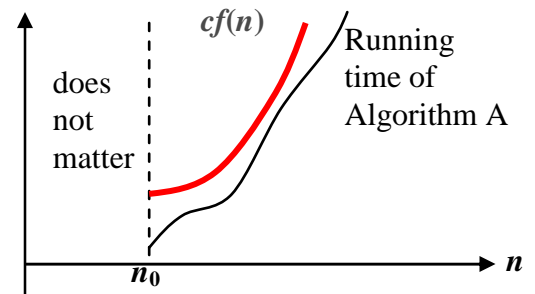
The **worst-case efficiency** of an algorithm is its efficiency for the worst-case input of size  $n$ . In the worst case, algorithm *equalMatrices* from Handout 2 performs  $n^2$  comparisons (if the matrices  $A$  and  $B$  are equal, the algorithm compares all  $n^2$  elements).

The **best-case efficiency** of an algorithm is its efficiency for the best-case input of size  $n$ . In the best case, *equalMatrices* performs one comparison if  $A[0,0] \neq B[0,0]$ .

The **average-case efficiency** of an algorithm is its efficiency for a “typical” or “random” input of size  $n$ .

The average-case analysis is more complicated than the worst-case analysis and it is beyond the scope of this module. In this course, we mainly consider the worst-case efficiency.

Algorithm A is order  $f(n)$  - denoted  $O(f(n))$  - if constants  $c$  and  $n_0$  exist such that A requires no more than  $cf(n)$  operations to solve a problem of size  $n \geq n_0$ .



You can simplify the analysis of algorithms:

- Determine the number of times the basic operations are performed (in the worst case) for an instance of size  $n$ .
- Find the dominating term in an algorithm's growth-rate function. The low-order terms can be ignored. For example, if a basic operation is performed  $5n^3 + 4n^2 + 3n$  times, the dominating term is  $5n^3$ .
- Ignore a multiplicative constant:  $5n^3$  is in  $O(n^3)$ .

*Examples:*

If the algorithm performs  $100n + 5$  basic operations for an input of size  $n$ , then the time complexity of the algorithm is  $O(n)$ .

$2n^2 + 3n$  is in  $O(n^2)$

$5n^3$  is in  $O(n^3)$

$7$  is in  $O(1)$

$2\log_2 n$  is in  $O(\log_2 n)$

**Common mistakes:**

~~$O(2n^2)$~~        $O(n^2)$   
 ~~$O(n^2 + n)$~~        $O(n^2)$

Only the dominant term is needed; the constants and the lower order terms should be removed.

As  $n$  gets large, the term with the highest power of  $n$  will come to dominate, so that all other terms can be neglected.

**Efficiency analysis ignores multiplicative constants and focuses on the order of growth**

Exercises:

The number of basic operations of eight algorithms are given below. Use the formal definition of big-Oh in order to determine their time complexities.

$$100n + 5$$

$$2n^2 + 3n$$

$$5n^3$$

$$7$$

$$2\log n$$

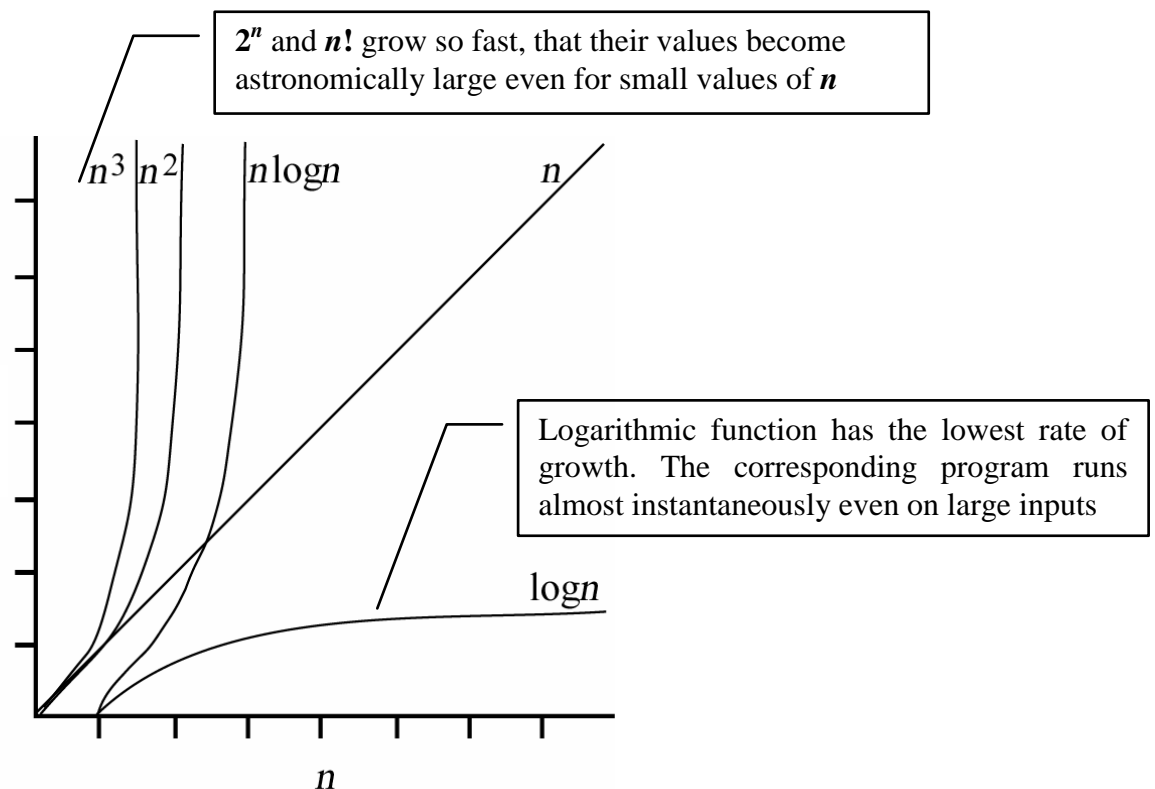
$$3n \log n + n$$

$$100n - 5$$

$$2n^2 - 3n$$

### Common efficiency classes

Class	Name	Examples
1	constant	finding the minimum value in an ordered array
$\log n$	logarithmic	binary search
$n$	linear	sequential search
$n \log n$	linear logarithmic	advanced sorting
$n^2$	quadratic	elementary sorting
$n^3$	cubic	matrix multiplication
$2^n$	exponential	combinatorial problems
$n!$	factorial	combinatorial problems



Algorithms that require  $2^n$  or  $n!$  operations are practical for solving only problems of very small size.

### Conclusions

- Both time and space efficiencies are measured as functions of the algorithm input size.
- Time efficiency is measured by counting the number of times the algorithm's basic operations are executed.
- Different algorithms solving the same problem may have different efficiencies. For such algorithms, we need to distinguish between the worst-case, best-case and average-case efficiencies.
- The most important characteristic of the algorithm is the order of growth of the algorithm's running time.