Algorithms III

Complexity Analysis

David Esparza Alba

January 29, 2021

Universidad Panamericana

Outline

1. Complexity Analysis
2. Big O notation
3. Most common running times

Complexity Analysis

What is it?

The complexity of an algorithm can be defined as how the performance of the algorithm changes as we change the size of the input data, this is reflected in the running time and in the memory space of the algorithm.

Big O notation

Big O

We use the so-called “big O notation” to measure the running time of an algorithm, we say that our algorithm runs in $O(f(n))$, if the algorithm executes *at most* $c \cdot f(n)$ operations to process an input of size n, where c is a positive constant and f is a function of n.

Most common running times

Most common running times

- **Constant Complexity.** Their performance does not change with the increase of the input size. We say these algorithms run in $O(1)$ time.
- **Linear Complexity.** The performance of these algorithms behaves linearly as we increase the size of the input data. For example, for 1 element, the algorithm executes c operations, for 10 elements, it makes $10c$ operations, for 100, it executes $100c$ operations, and so on. We say that these algorithms run in $O(n)$ time.

Most common running times

- **Logarithmic Complexity.** The performance increases in a logarithmic way as we increase the input data size. Meaning that the if we have 10 elements, the algorithm will execute $\log 10$ operations, which is around 2.3. For 1000 elements, it makes around of 7 operations, depending on the base of the logarithm. In most of the cases we deal with base-2 logarithms, where:

$$\log_2 n = \frac{\log n}{\log 2}$$

  Since $1/\log 2$ is a multiplicative factor, then we say that these algorithms run in $O(\log n)$ time.

Most common running times

- **Polynomial Complexity.** For this kind of algorithms their performance grows in a polynomial rate according to the input data. Some of the most famous sorting algorithms, the *Bubble Sort*, runs in quadratic time, $O(n^2)$. A well known algorithm that runs in cubic time, $O(n^3)$ is the matrix multiplication.

Most common running times

- **Exponential Complexity.** These are the worst cases and must be avoided if possible, since for a small increment in the input data, their performance grows considerably. Problems that require exponential solutions, can be considered as the hardest ones. We say that these algorithms run in (a^n) time, for some value of $a > 1$.