

**Instituto Tecnológico de Tijuana**

**Subdirección Académica**

**Departamento de Sistemas y Computación**

**Semestre agosto-diciembre 2018**

**Carrera:** Ing. En Sistemas Computacionales Serie SC1A

**Materia:** Estructura de Datos

**Unidad 1 – Análisis de Algoritmos, Complejidad de un espacio y ejecución y Eficiencia de Algoritmo**

**Alumno:**15211883 - Angeles Valadez Jonathan

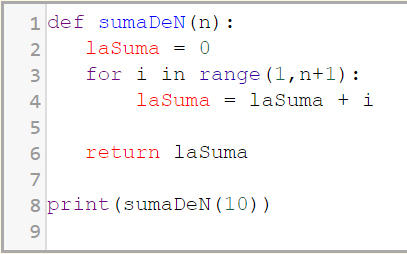
**Profesor:** Ray Brunett Parra Galaviz

**¿Qué es análisis de algoritmos?**

Es muy común que los estudiantes principiantes de ciencias de la computación comparen sus programas entre sí. También usted puede haber notado que es común que los programas de computadora se vean muy similares, especialmente los más simples. A menudo surge una pregunta interesante. Cuando dos programas resuelven el mismo problema pero se ven diferentes, ¿es un programa mejor que el otro?

Con el fin de responder esta pregunta, tenemos que recordar que hay una diferencia importante entre un programa y el algoritmo subyacente que el programa está representando. Como dijimos en el Capítulo 1, un algoritmo es una lista genérica, paso a paso, de instrucciones para resolver un problema. Es un método para resolver cualquier caso del problema de tal manera que dada una entrada particular, el algoritmo produzca el resultado deseado. Un programa, por otro lado, es un algoritmo que ha sido codificado en algún lenguaje de programación. Pueden existir muchos programas para el mismo algoritmo, dependiendo del programador y del lenguaje de programación que se esté utilizando.

Esta función resuelve un problema familiar: calcular la suma de los primeros n enteros. El algoritmo utiliza la idea de una variable acumuladora que se inicializa en 0. La solución itera entonces a través de los n enteros, agregando cada uno a la variable acumuladora.



El análisis de algoritmos se ocupa de compararlos con base en la cantidad de recursos computacionales que utiliza cada algoritmo. Queremos ser capaces de considerar dos algoritmos y decir que uno es mejor que el otro, porque es más eficiente en su uso de esos recursos o simplemente tal vez porque utiliza una menor cantidad.

En este punto, es importante pensar más en lo que realmente queremos decir con recursos computacionales. Hay dos formas diferentes de ver esto. Una forma es considerar la cantidad de espacio o memoria que un algoritmo requiere para resolver el problema. La cantidad de espacio requerida por una solución suele ser dictada por el caso particular del problema. De vez en cuando, sin embargo, hay algoritmos que tienen requisitos de espacio muy específicos, y en esos casos seremos muy cuidadosos al explicar las variaciones.

Notación O-grande

Al tratar de caracterizar la eficiencia de un algoritmo en términos del tiempo de ejecución, independientemente de cualquier programa o computadora en particular, es importante cuantificar el número de operaciones o pasos que el algoritmo requerirá. Si se considera que cada uno de estos pasos es una unidad básica de cálculo, entonces el tiempo de ejecución de un algoritmo puede expresarse como el número de pasos necesarios para resolver el problema. Decidir sobre una unidad básica de cálculo apropiada puede ser un problema complicado y dependerá de cómo se implemente el algoritmo.

Una buena unidad básica de cálculo para comparar los algoritmos de sumatoria mostrados anteriormente podría ser contar el número de instrucciones de asignación realizadas para calcular la suma. En la función sumaDeN, el número de instrucciones de asignación es 1 (laSuma=) más el valor de *n* (el número de veces que ejecutamos laSuma = (laSuma +i). Podemos denotar esto por una función, digamos *T*, donde T(n)=1+n. El parámetro *n* a menudo se denomina el “tamaño del problema”, y podemos interpretar la función como “*T(n)* es el tiempo que se necesita para resolver un problema de tamaño *n*, a saber, *1+n* pasos”.

Los científicos de la computación prefieren llevar esta técnica de análisis un poco más allá. Resulta que el número exacto de operaciones no es tan importante como determinar la parte más dominante de la función T(n). En otras palabras, a medida que el problema se hace más grande, una parte de la función T(n) tiende a dominar la parte restante. Este término dominante es lo que, al final, se utiliza para la comparación. La función **orden de magnitud** describe la parte de T(n) que más rápido crece a medida que aumenta el valor de *n*. El orden de magnitud es a menudo llamado notación **O-grande** (por “orden”) y se escribe como O(f(n)). Esta notación proporciona una aproximación útil al número real de pasos en el cálculo. La función f(n) brinda una representación sencilla de la parte dominante de la función T(n) original.

**Bibliografía**

[En Línea]: <http://interactivepython.org/runestone/static/pythoned/AlgorithmAnalysis/NotacionOGrande.html>

[En Línea]: <http://interactivepython.org/runestone/static/pythoned/AlgorithmAnalysis/QueEsAnalisisDeAlgoritmos.html>