Notación Big O

Definición

$T(N) = O(f(N))$, si y solo si existen dos constantes c, $N_0 > 0$ de forma que $T(N) \leq c*f(N)$ para todo $N > N_0$

En términos prácticos, se calcula así:

- Se considera el peor escenario (la mayor cantidad posible de operaciones)
- Se realiza un análisis asintótico (enfocarse en valores grandes de N)
- No se presta atención a términos constantes o de orden menor

Solución 1

```
Leer A
Para i = 2 to A-1:
   Si A % i = 0:
      Imprimir i
```

Número de operaciones:

$$1 + (A - 2)(1 + 1') \rightarrow f(N) = 2N - 3$$

Luego, podemos decir que ese algoritmo tiene un orden de complejidad (*time complexity*) ***O(N)***

Solución 2

```
Leer A
Para i = 2 to A/2:
   Si A % i = 0:
      Imprimir i
```

Número de operaciones:

$$1 + \left(\frac{A}{2} - 1\right)(1 + 1') \rightarrow f(N) = N - 1$$

Luego, podemos decir que ese algoritmo es ***O(N)***

Solución 3

```
Leer A
Para i = 2 to √A:
   Si A % i = 0:
      Si A/i ≠ i
         Imprimir i
      De lo contrario:
         Imprimir i, A/i
```

Número de operaciones:

$$1 + \left(\sqrt{A} - 1\right)(1 + 1' + 1'') \rightarrow f(N) = 3\sqrt{N} - 2$$

Luego, podemos decir que ese algoritmo es $\boldsymbol{O(N^{\frac{1}{2}})}$

Utilidad de la notación Big O

Aunque esta notación no es rigurosa, y no se debe usar como un predictor exacto del tiempo de ejecución de un algoritmo, si da cuenta de su tasa de crecimiento cuando varia el tamaño de la entrada. Por ejemplo, si un algoritmo es $O(N^2)$ podemos decir que su tasa de crecimiento es cuadrática.

Decimos entonces que un algoritmo es mejor (más eficiente) que otro si su “O” es menor. En otras palabras si su tiempo de ejecución (determinado por la cantidad de operaciones básicas) considerando el peor escenario crece más lentamente a medida que se aumenta el tamaño de la entrada.