Tarea 1 - Análisis y Diseño de Algoritmos – Comparación de Crecimiento de Funciones.

Cristóbal Flores Villegas

21.324.739-5

Nrc: 8218

1. Introducción:

En esta tarea, se analizan diversas funciones de uso frecuente en el estudio de la complejidad de los algoritmos. El objetivo es comparar su crecimiento al aumentar el tamaño de la entrada (nnn) y comprender la relación entre la complejidad de un algoritmo y su tiempo de ejecución. Para ello, se desarrolló un programa en Python que gráfica, en un mismo plano, las siguientes funciones para 1 ≤ n ≤ 10.001:

- n
- n 2
- log n
- n 3
- 2 n
- $n!$
- n n
- n log n
- n log n 2
- n 2 log n

2. Metodología:

Se utilizo matplotlib para poder graficar, para valores muy grandes (por ejemplo, n^n, n!) se implementaron funciones auxiliares con manejo de errores (try/except) que devuelven float('inf') cuando Python no puede representar numéricamente la magnitud de la función (OverflowError). Se empleó la **escala logarítmica** en el eje yyy (plt.yscale('log')) para visualizar mejor las diferencias de crecimiento.

3. Orden de crecimiento (Pregunta a)

Basado en la teoría de crecimiento de funciones, así como en la gráfica obtenida para valores grandes de nnn, el orden de menor a mayor es aproximadamente:

$\text{Log}(n) < n < n\log(n) <$ *n* log (*n*^2) < *n*^2 < *n*^2 log (*n)* < *n* ^ 3 < 2 ^ *n* < *n* ! < *n*^*n*

Explicación breve:

1. Log(*n*) crece más lento que cualquier **potencia** de n
2. *n* log (*n*^2)) no cambia el orden general, pero es un factor más grande que Log(*n*)
3. Entre polinomios, *n* < *n*^2 < *n* ^ 3.
4. 2 ^ *n* (exponencial) eventualmente supera a cualquier polinomio.
5. *n* ! crece más rápido que 2 ^ *n* , y *n*^*n* es la función que finalmente sobrepasa a todas.

4. Algoritmos "eficientes" (Pregunta b)

En análisis de algoritmos, se considera que **un algoritmo es "eficiente"** si su complejidad de tiempo crece a lo sumo de manera **polinómica** con respecto al tamaño de la entrada. De esta manera:

- **Eficientes**:

  Log(*n*), *n*, *n* log (*n*), *n* log (*n*^2), *n*^2, *n*^2 log (*n), n* ^ 3

  Se incluyen aquí todas las variantes polinomiales y subpolinomiales (logarítmicas).

- **No eficientes**:

  2 ^ *n*, *n* !, *n*^*n*

  Son funciones **exponenciales** (o peores que exponenciales), que se vuelven inviables para nnn relativamente grande.

5. Conclusiones

- El programa demuestra cómo, para valores altos de ***n***, las funciones factorial y exponenciales se disparan a infinito, confirmando su alto costo computacional.
- La escala logarítmica en el eje ***y*** facilita la comparación, aun cuando varias funciones terminan en inf para ***n ≈ 10 000.***

  Los resultados coinciden con la teoría clásica de complejidad de algoritmos: **polinomios** y menores (Log(*n*), *n*, etc.) se consideran manejables, mientras que funciones como(2 ^ *n, n* !, *n*^*n)* suelen ser prohibidamente costosas en la práctica. 2 ^ *n*, *n* !, *n*^*n*