Análisis de Algoritmos

Etiquetas

Análisis de Algoritmos

El análisis de algoritmos es el estudio de la eficiencia de los métodos o procedimientos (algoritmos) para resolver problemas. Se evalúa principalmente en dos aspectos:

- **Tiempo de ejecución:** Cuánto tiempo tarda el algoritmo en completarse.
- **Uso de recursos (memoria):** Cuánta memoria necesita el algoritmo para funcionar.

Esta evaluación permite comparar distintos algoritmos y elegir el más adecuado para cada situación, especialmente cuando se trabaja con grandes cantidades de datos.

Notaciones Asintóticas

Las notaciones O (Big O), Ω (Omega) y Θ (Theta) describen cómo crecen el tiempo de ejecución (o el uso de memoria, u otro recurso) de un algoritmo a medida que el tamaño de la entrada (n) aumenta.

1. **O grande (Big O)**:
   - Representa un **límite superior** de la complejidad.
   - "f(n) es O(g(n))" significa que la función f(n) (tiempo/recursos del algoritmo) no crece más rápido que g(n) **a partir de cierto punto**.
   - Intuitivamente, describe el **peor caso** o la cota máxima de crecimiento.
2. **Omega (Ω)**:
   - Indica un **límite inferior** de la complejidad.
   - "f(n) es Ω(g(n))" significa que la función f(n) (tiempo/recursos) no crece más lento que g(n) **a partir de cierto punto**.
   - Intuitivamente, describe el **mejor caso** o la cota mínima de crecimiento.
3. **Theta (Θ)**:
   - Se usa cuando f(n) es **tanto O(g(n)) como Ω(g(n))** al mismo tiempo.
   - Indica que la función crece en el mismo orden que g(n).
   - Es decir, "f(n) es Θ(g(n))" → "f(n) es O(g(n)) y f(n) es Ω(g(n))".
   - Describe el comportamiento del algoritmo cuando **la cota superior y la cota inferior son iguales**.

---

2. Ejemplos Clásicos de Complejidad

2.1 Quicksort

Quicksort es un algoritmo de ordenamiento muy utilizado por su eficiencia promedio, aunque su complejidad depende en gran medida de cómo se elija el pivote.

- **Peor caso (Big O):** $O(n^2)$

- Sucede cuando, en cada partición, el pivote divide la lista de forma muy desigual (por ejemplo, si siempre el pivote queda en un extremo).
- En esa situación, para cada partición se reduce muy poco el problema y, finalmente, hay que hacer demasiadas recursiones.

- **Mejor caso (Big Ω):** Ω(n log n)
  - Ocurre cuando el pivote divide la lista en partes aproximadamente iguales.
  - Cada llamada recursiva trabaja en sublistas de tamaño cercano a n/2, lo que nos da esa complejidad n log n.

2.2 Mergesort

Mergesort es otro algoritmo de ordenamiento clásico que opera mediante la división y fusión de sublistas.

- **Peor caso:** O(n log n)
- **Mejor caso:** Ω(n log n)
- **Caso medio:** Θ(n log n)

Aquí es más fácil de analizar porque, aunque la lista ya esté ordenada, mergesort igualmente subdivide y fusiona, lo que conlleva un costo significativo en cada paso. Así, en la práctica, el tiempo es prácticamente igual en cualquier escenario.

2.3 Búsqueda Binaria

La búsqueda binaria se aplica en listas **ordenadas** y reduce el espacio de búsqueda a la mitad en cada paso.

- **Peor caso (O):** O(log n)
  - Incluso en la peor situación, en cada comparación descartamos la mitad de los elementos.
- **Mejor caso (Ω):** Ω(1)
  - Si el elemento que buscamos está justo en el centro en la primera comparación, lo encontramos de inmediato.

2.4 Búsqueda Lineal

La búsqueda lineal compara uno a uno los elementos de una lista sin ordenar (o cuando no podemos aplicar otra estrategia).

- **Peor caso (O):** O(n)
  - Si el elemento buscado está al final de la lista o no existe, se revisan prácticamente todos los elementos.
- **Mejor caso (Ω):** Ω(1)
  - Si el elemento está en la primera posición.

Resumen

- **O (Big O)**: Cota superior → Peor caso.
- **Ω (Omega)**: Cota inferior → Mejor caso.
- **Θ (Theta)**: Orden exacto (siempre crece igual).

Recursos Recomendados

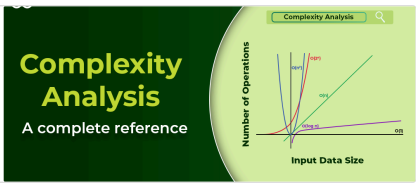
Páginas Web:

- **GeeksforGeeks**

Analysis of Algorithms - GeeksforGeeks

Analysis of Algorithms is crucial in computer science for evaluating the efficiency of algorithms in terms of time and space complexity, utilizing concepts like asymptotic analysis and various notations such as Big-O, Theta, and Big-Omega.

https://www.geeksforgeeks.org/analysis-of-algorithms/



Amplios tutoriales y ejemplos prácticos sobre algoritmos y estructuras de datos.

- **VisuAlgo**

visualising data structures and algorithms through animation - VisuAlgo

VisuAlgo was conceptualised in 2011 by Dr Steven Halim as a tool to help his students better understand data structures and algorithms, by allowing them to learn the basics on their own and at their own pace. Together with his students from the National University of Singapore, a series of visualizations were developed and consolidated, from simple sorting algorithms to complex graph data structures. Though specifically designed for

https://visualgo.net/en

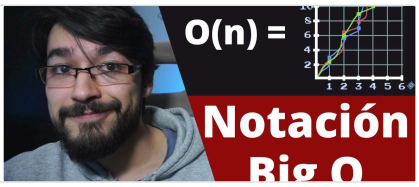
Herramienta visual para comprender el funcionamiento de algoritmos mediante animaciones.

Videos:

Notación Big O | Análisis de algoritmos de forma sencilla

Analizar nuestro código para determinar cómo se comporta con diferentes entradas no es tan difícil una vez aprendes la notación asintótica Big O. Aprende cómo calcular Big O cuando tu código tiene ciclos usando la Búsqueda por Insersión como ejemplo.

https://youtu.be/MyAiCtuhiqQ?si=yqmb90CcOJRFXj_H

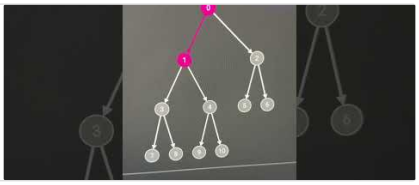


¡Aprende Algoritmos de Programación con este recurso!

¡Los explica paso a paso y con diagramas visuales!

+70 algoritmos en JavaScript, Java y C++

https://youtube.com/shorts/NvQSAWV73fo?si=0UAFfyvLc1MxK7O8



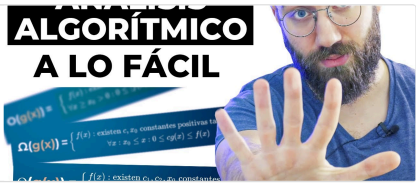
Cómo ANALIZAR tus ALGORITMOS (en Ingeniería Informática) 👨‍💻📈

🧠 Redes sociales 🧠

▶ Twitter: https://twitter.com/bettatech

▶ Instagram: https://www.instagram.com/betta_tech

https://youtu.be/IZgOEC0NIbw?si=neuJ-K-Ds3TDYIKD



Análisis y Diseño de Algoritmos | Qué es un algoritmo? | Es necesario estudiarlos?

El análisis y diseño de algoritmos es tal vez una de las partes más tediosas de aprender a la hora de estudiar desarrollo de software. Pero en realidad es necesario? Qué es un algoritmo?

https://youtu.be/Z-M_I130Rfk?si=BKDaGEDtbk6kkVI0

Análisis y Diseño de Algoritmos

https://youtube.com/playlist?list=PLfBtpqIBIz7oigjkp6SIKYbz2AkAEqFIE&si=nhJT_RTnVMJ9ptfm

Big O para algoritmos Recursivos | Análisis de Algoritmos

La notación Big O se usa para determinar la complejidad de tiempo de nuestros algoritmos. En este video veremos tres métodos para determinar la complejidad Big O de algoritmos recursivos. Usando el método de sustitución, el método del arbol recursivo y el método maestro.

https://youtu.be/dqFS-CXCEVQ?si=G3U07SuNYrycAFqQ

¿Cómo piensa tu PC? Algoritmos de ordenación y complejidad temporal (Big O) explicados

GVGmall 25% Cupón para → NG20
Windows 11 Pro Key (€21): https://biitt.ly/NG11
Windows 10 Pro Key (€14): https://biitt.ly/NG10

https://youtu.be/TnTu1QWaWc0?si=7zUvsoMxmNNqn7VU

Big Oh(O) vs Big Omega(Ω) vs Big Theta(θ) notations | Asymptotic Analysis of Algorithms with Example

Support Simple Snippets by Donations -
Google Pay UPI ID - tanmaysakpal11@okicici
PayPal - paypal.me/tanmaysakpal11

https://youtu.be/1tfdr1Iv6JA?si=LPG6ALAziFtgDBRa