

Análisis de Algoritmos

Fundamentos y ejemplos prácticos sobre
cómo analizar algoritmos de manera
eficiente.

Análisis de Algoritmos

Análisis de Algoritmos

Fundamentos y ejemplos prácticos que demuestran la importancia y aplicabilidad del análisis de algoritmos en la informática moderna.

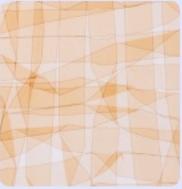


2010年10月31日在内蒙古呼和浩特进行教师培训。



Eficiencia y Efectividad

El análisis de algoritmos evalúa la eficiencia y efectividad de los algoritmos en términos de tiempo y recursos. Es fundamental para garantizar el funcionamiento óptimo de los programas informáticos.



Importancia en Informática

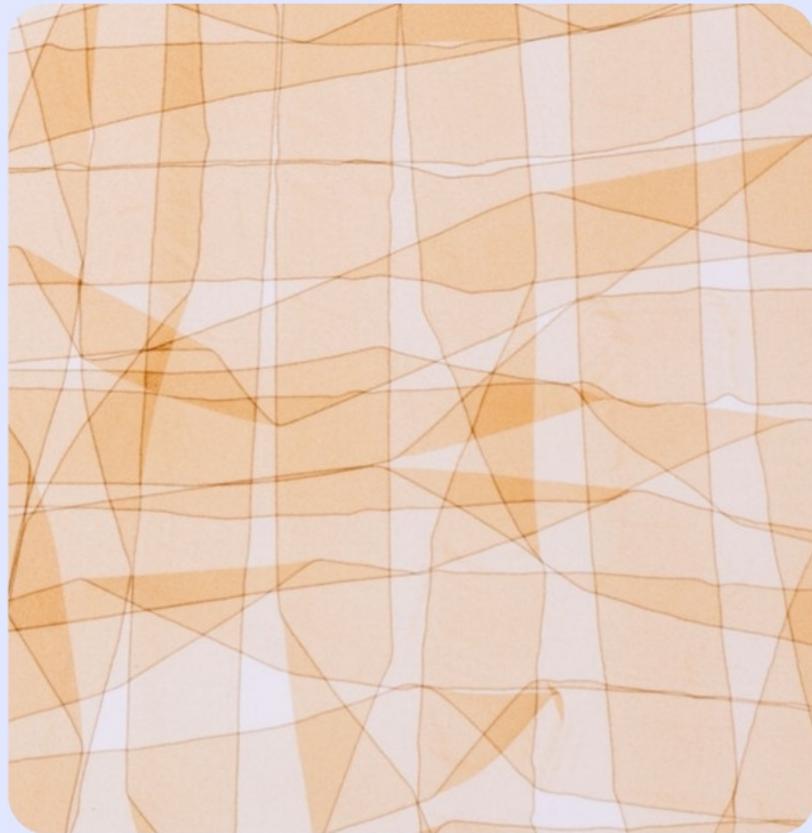
Este análisis es esencial en informática ya que permite a los desarrolladores seleccionar las soluciones más adecuadas para problemas específicos, optimizando así el rendimiento general de las aplicaciones.



Definición e Importancia del Análisis de Algoritmos

Eficiencia y Efectividad

El análisis de algoritmos evalúa la eficiencia y efectividad de los algoritmos en términos de tiempo y recursos. Es fundamental para garantizar el funcionamiento óptimo de los programas informáticos.



Importancia en Informática

Este análisis es esencial en informática ya que permite a los desarrolladores seleccionar las soluciones más adecuadas para problemas específicos, optimizando así el rendimiento general de las aplicaciones.



Tiempos de Ejecución de Algoritmos

Análisis de la variabilidad de los tiempos de ejecución según el tamaño de la entrada.

Pequeñas Entradas

Los algoritmos generalmente muestran un rendimiento óptimo en entradas pequeñas, con tiempos de ejecución que permiten un análisis rápido y eficiente.

Grandes Entradas

A medida que el tamaño de la entrada aumenta, los tiempos de ejecución pueden dispararse, evidenciando la importancia de seleccionar algoritmos eficientes para grandes volúmenes de datos.

Pequeñas Entradas

Los algoritmos generalmente muestran un rendimiento óptimo en entradas pequeñas, con tiempos de ejecución que permiten un análisis rápido y eficiente.

Grandes Entradas

A medida que el tamaño de la entrada aumenta, los tiempos de ejecución pueden dispararse, evidenciando la importancia de seleccionar algoritmos eficientes para grandes volúmenes de datos.

Tiempos de Ejecución de Algoritmos

Análisis de la variabilidad de los tiempos de ejecución según el tamaño de la entrada.

Pequeñas Entradas

Los algoritmos generalmente muestran un rendimiento óptimo en entradas pequeñas, con tiempos de ejecución que permiten un análisis rápido y eficiente.

Grandes Entradas

A medida que el tamaño de la entrada aumenta, los tiempos de ejecución pueden dispararse, evidenciando la importancia de seleccionar algoritmos eficientes para grandes volúmenes de datos.

Complejidad Constante $O(1)$

La complejidad constante $O(1)$ indica que el tiempo de ejecución no cambia sin importar el tamaño de la entrada. Es el caso más eficiente.

Complejidad Lineal $O(n)$

$O(n)$ indica que el tiempo de ejecución crece linealmente en relación al tamaño de la entrada. Ejemplo: recorrer un array.

Complejidad Cuadrática $O(n^2)$

$O(n^2)$ es típico en algoritmos de fuerza bruta, donde se requieren dos bucles anidados, como en la búsqueda de pares de elementos.

Complejidad Logarítmica $O(\log n)$

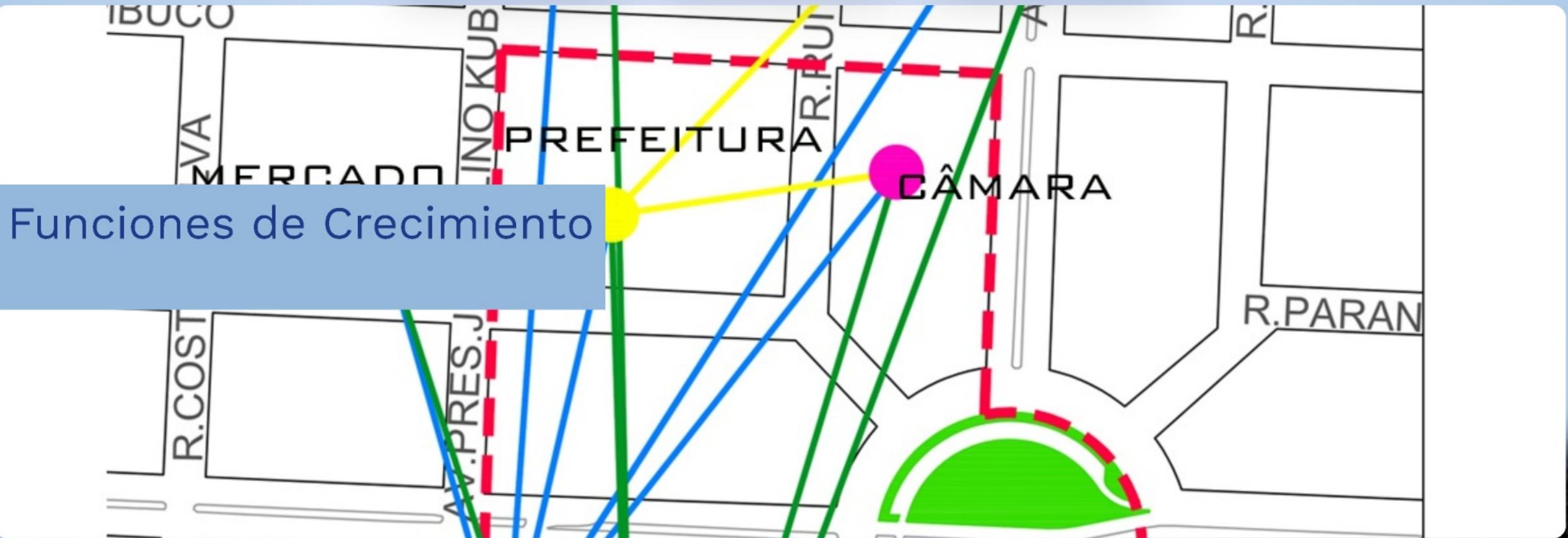
La complejidad logarítmica $O(\log n)$ es típica en algoritmos que dividen el problema en partes más pequeñas, como la búsqueda binaria.

Complejidad $O(n \log n)$

La complejidad $O(n \log n)$ es común en algoritmos de ordenamiento eficientes, como el Merge Sort.

Complejidad Exponencial $O(2^n)$

La complejidad $O(2^n)$ representa algoritmos que crecen exponencialmente, como los que resuelven problemas de combinatoria.





Complejidad Constante $O(1)$

La complejidad constante $O(1)$ indica que el tiempo de ejecución no cambia sin importar el tamaño de la entrada. Es el caso más eficiente.



Complejidad Logarítmica $O(\log n)$

La complejidad logarítmica $O(\log n)$ es típica en algoritmos que dividen el problema en partes más pequeñas, como la búsqueda binaria.



Complejidad Lineal $O(n)$

$O(n)$ indica que el tiempo de ejecución crece linealmente en relación al tamaño de la entrada. Ejemplo: recorrer un array.



Complejidad $O(n \log n)$

La complejidad $O(n \log n)$ es común en algoritmos de ordenamiento eficientes, como el Merge Sort.



Complejidad Cuadrática $O(n^2)$

$O(n^2)$ es típico en algoritmos de fuerza bruta, donde se requieren dos bucles anidados, como en la búsqueda de pares de elementos.



Complejidad Exponencial $O(2^n)$

La complejidad $O(2^n)$ representa algoritmos que crecen exponencialmente, como los que resuelven problemas de combinatoria.

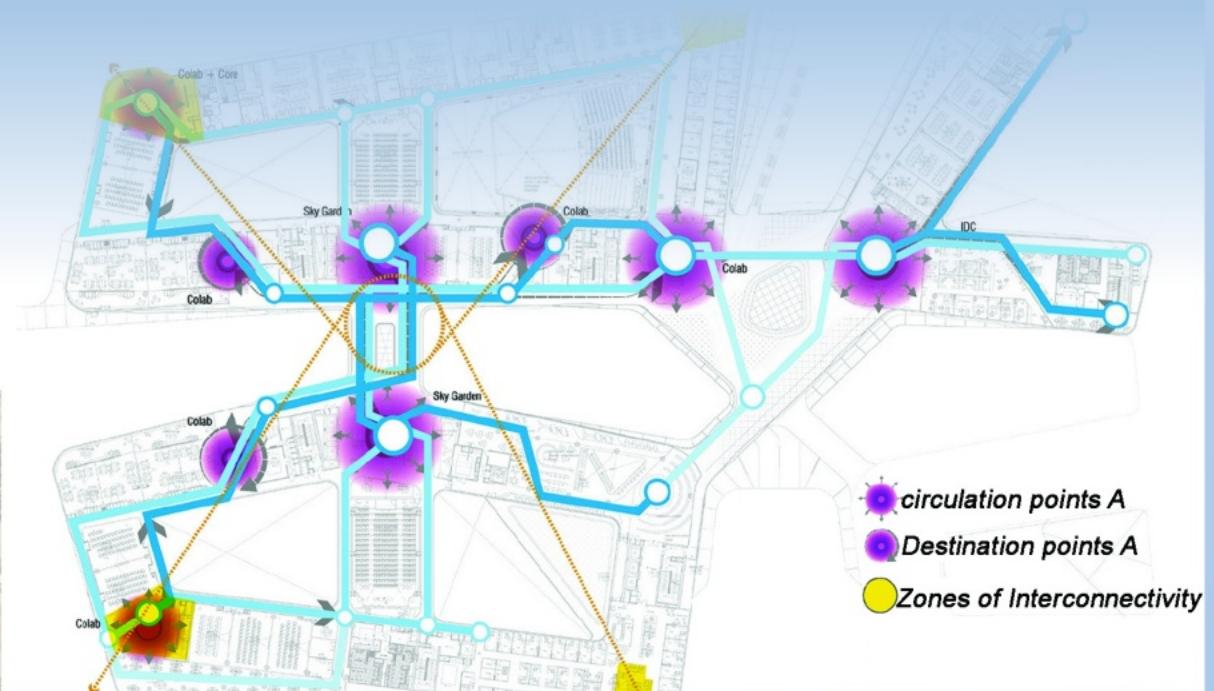
Notación Big-O: Comprendiendo el Rendimiento de los Algoritmos



Notación Big-O: Comprendiendo el Rendimiento de los Algoritmos



La notación Big-O proporciona una manera estandarizada de expresar la complejidad temporal y espacial de un algoritmo. Esto permite a los desarrolladores y científicos de la computación evaluar y comparar la eficiencia de diferentes algoritmos en función del tamaño de la entrada. Las reglas generales que la rigen incluyen la identificación de la función de mayor crecimiento, ignorar constantes y términos de menor orden, y enfocarse en el comportamiento asintótico al escalar los datos.



Limitaciones de la Notación Big-O

La notación Big-O no considera factores como la constante de tiempo y el rendimiento en la práctica. Por ejemplo, un algoritmo con $O(n)$ puede ser más lento en comparación con otro de $O(n^2)$ para tamaños de entrada pequeños debido a diferencias en implementaciones o constantes ocultas.



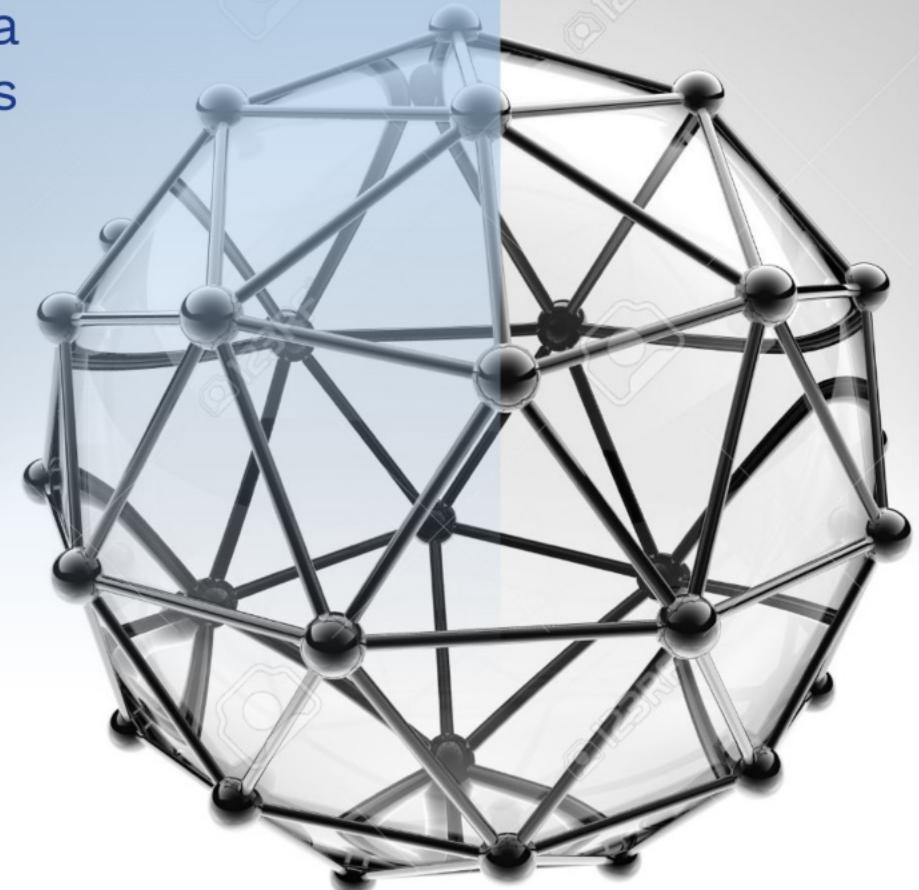
Consideraciones Prácticas

Existen situaciones donde el análisis Big-O no proporciona una imagen completa. Un algoritmo puede tener una excelente complejidad teórica, pero su rendimiento podría verse afectado por la naturaleza de los datos de entrada, lo que se debe tener en cuenta en la práctica.



Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones

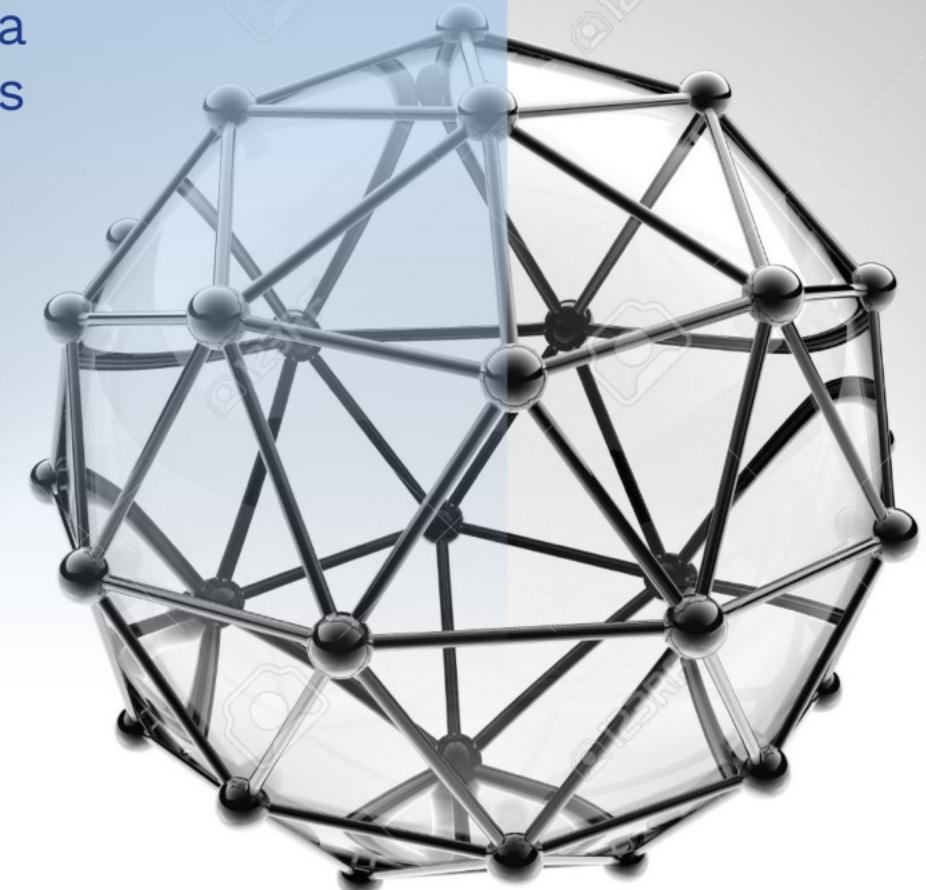


Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones

Definición del análisis de algoritmos

El análisis de algoritmos es fundamental para evaluar la eficiencia de las soluciones.

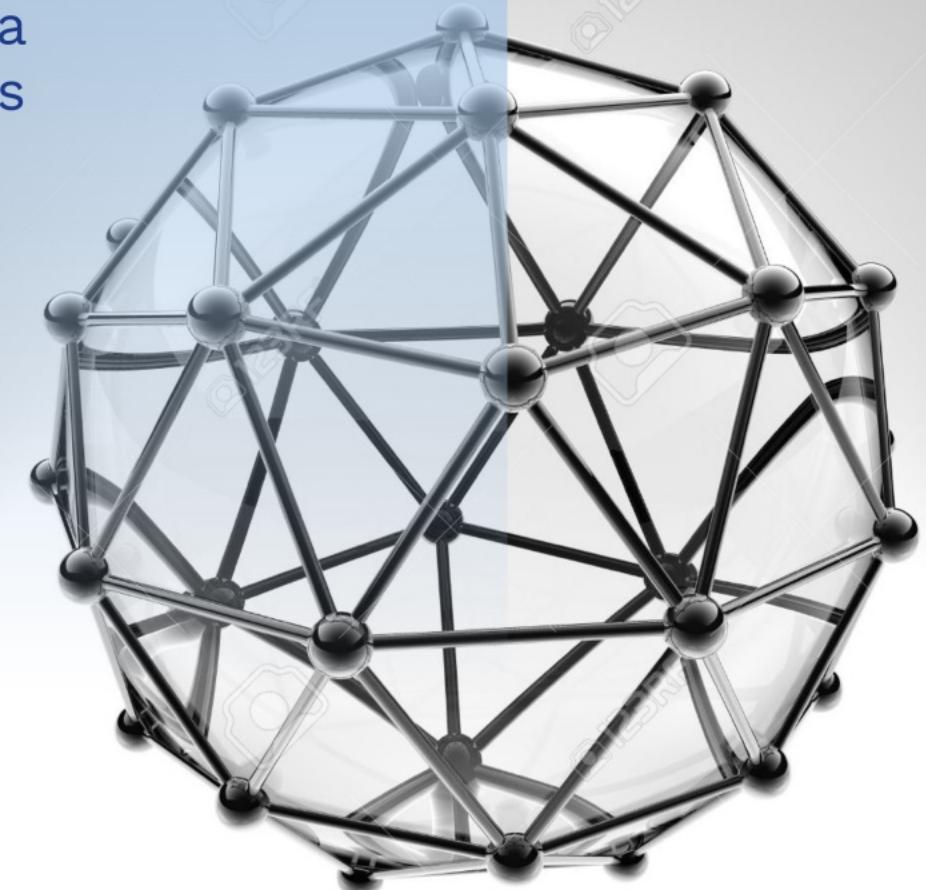


Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones

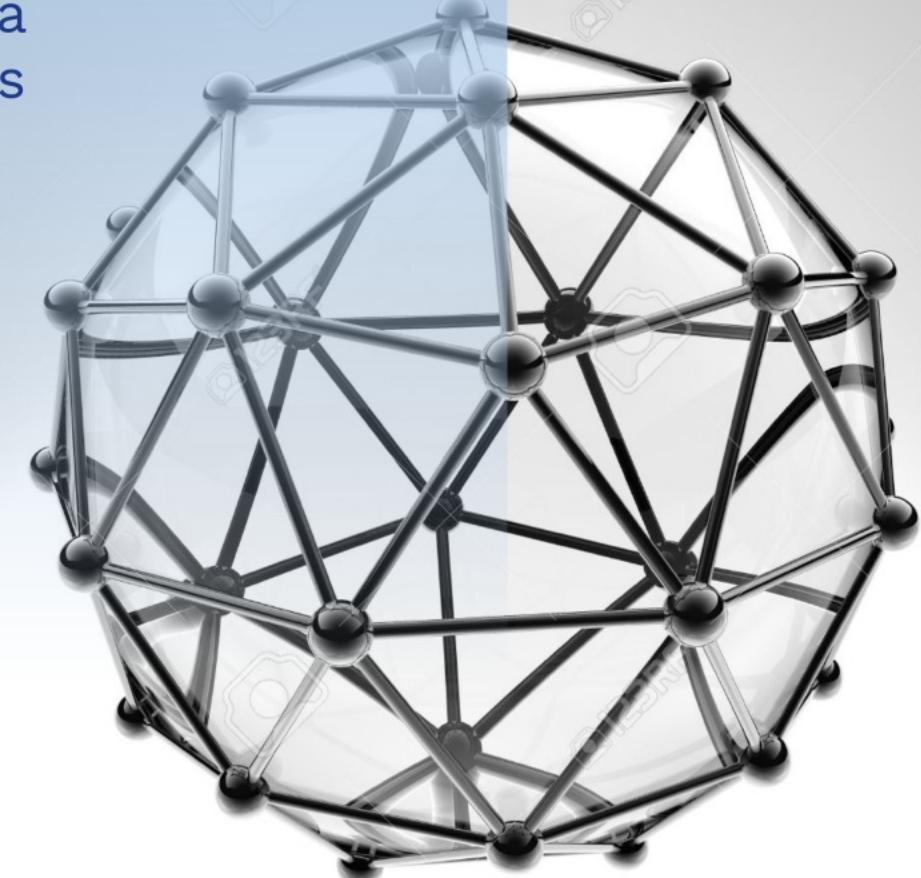
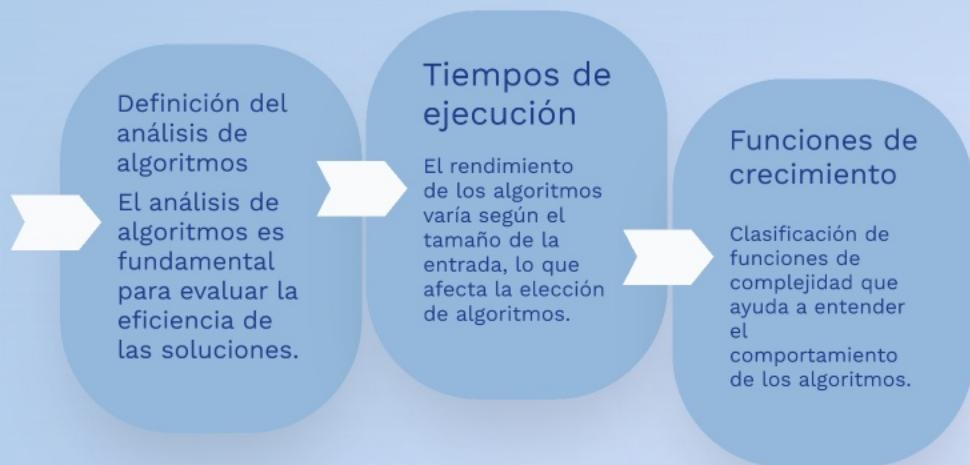
Definición del análisis de algoritmos
El análisis de algoritmos es fundamental para evaluar la eficiencia de las soluciones.

Tiempos de ejecución
El rendimiento de los algoritmos varía según el tamaño de la entrada, lo que afecta la elección de algoritmos.



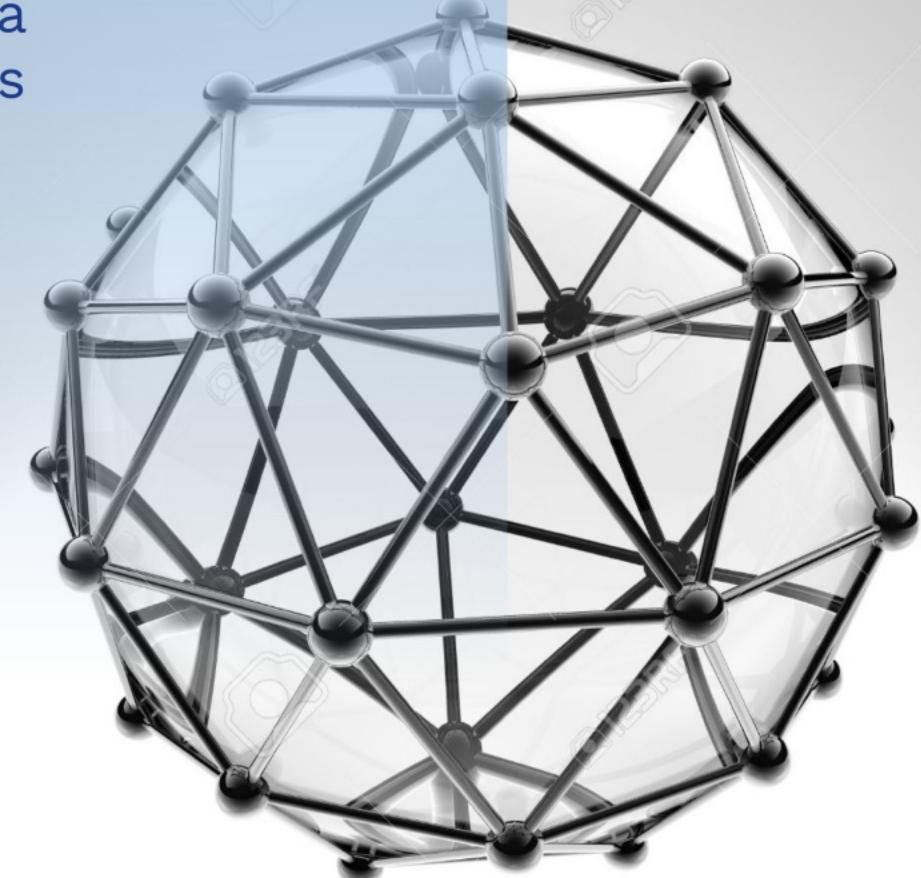
Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones



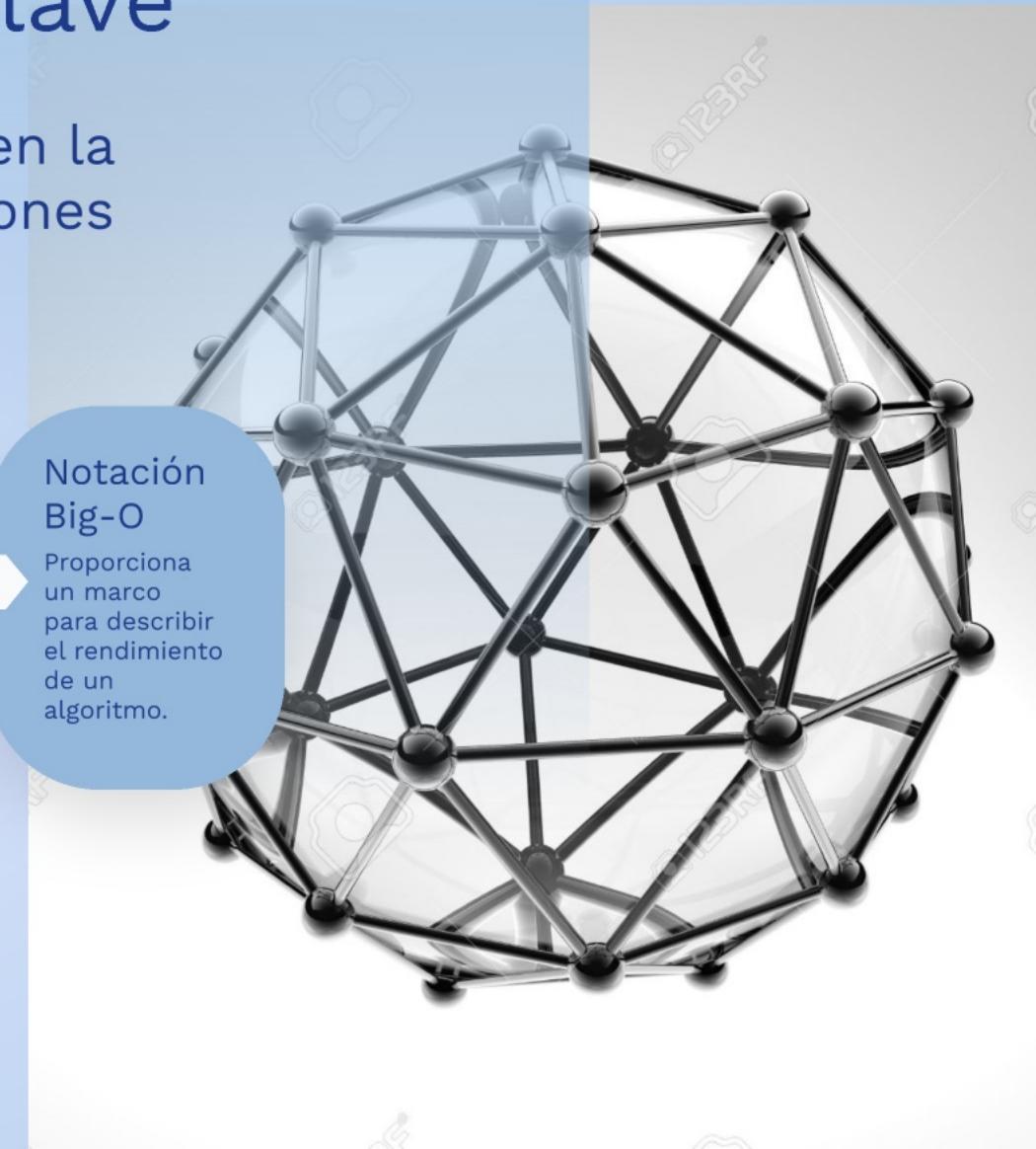
Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones



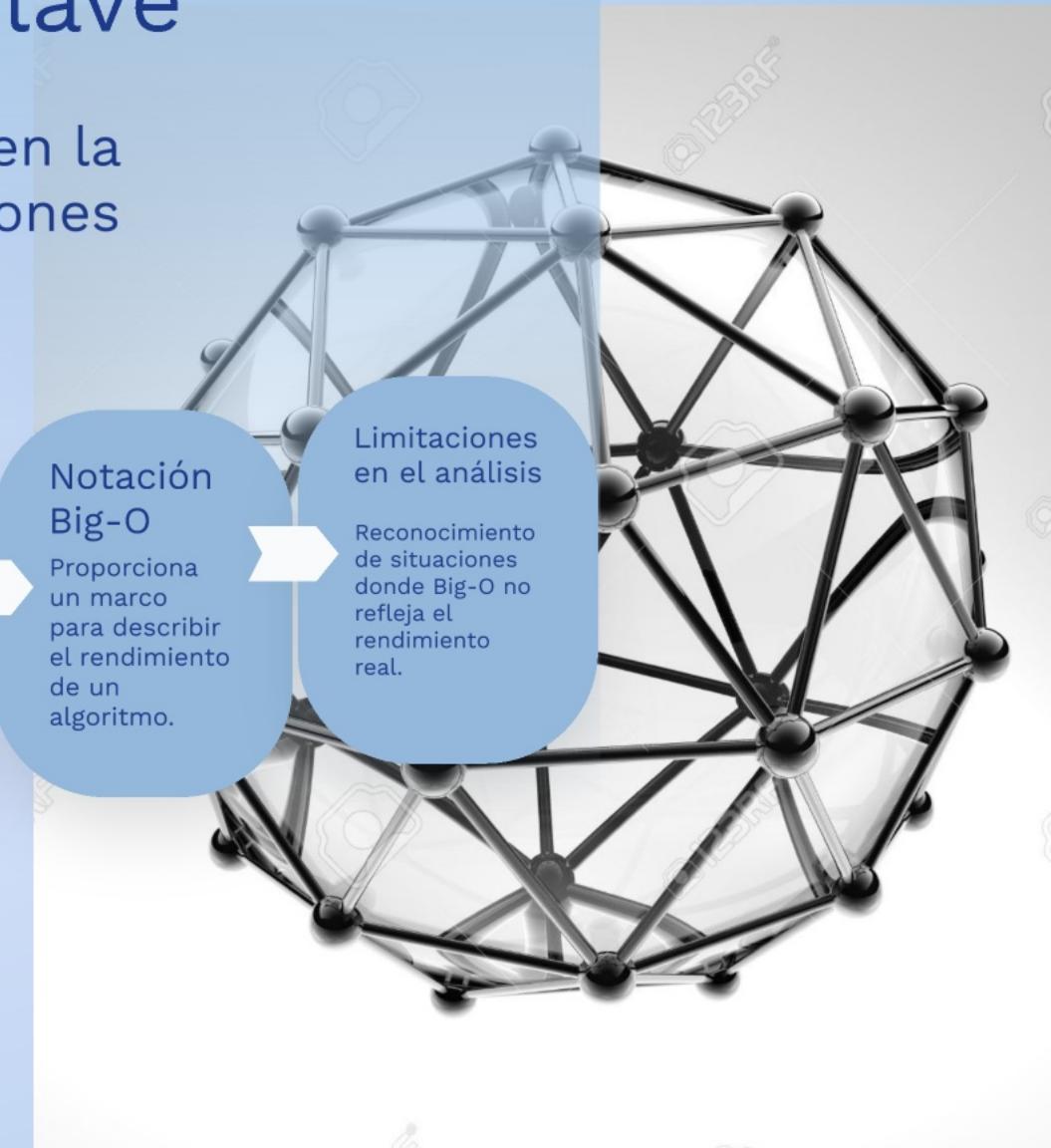
Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones



Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones



Resumen de Conceptos Clave

Importancia del análisis de algoritmos en la eficiencia de soluciones



Funciones de crecimiento más comunes

$c \rightarrow$ Constant (Constante)

$\log N \rightarrow$ Logarithmic (Logarítmica)

$\log^2 N \rightarrow$ Log-squared (Log-cuadrado)

$N \rightarrow$ Linear (Lineal)

$N \log N \rightarrow$ N log N (Linealítmico)

$N^2 \rightarrow$ Quadratic (Cuadrática)

$N^3 \rightarrow$ Cubic (Cúbica)

$2^N \rightarrow$ Exponential (Exponencial)

Constante – O(1)

Acceder a un elemento en un arreglo por índice: arr[5].
(Siempre tarda lo mismo, sin importar el tamaño del arreglo).

Logarítmica – O(log N)

Búsqueda binaria en un arreglo ordenado.
(Cada paso divide el problema a la mitad).

Log-cuadrado – O(log² N)

Algoritmo de búsqueda en matrices ordenadas (dividir filas y columnas logarítmicamente).

Lineal – O(N)

Recorrer un arreglo para encontrar el valor máximo o mínimo.

Linealítica – O(N log N)

Quicksort o Mergesort en el caso promedio.
(Divide y conquista con log N niveles de recursión).

Cuadrática – O(N²)

Ordenamiento por burbuja (Bubble Sort) o inserción en el peor caso.
Comparar todos contra todos en un grafo completo.

Cúbica – O(N³)

Multiplicación de matrices con el algoritmo clásico.
(Tres bucles anidados).

Exponencial – O(2^N)

Resolver el problema de la mochila (fuerza bruta).
Generar todas las posibles combinaciones de subconjuntos.

Funciones de crecimiento con ejemplos

Insertion.java •

▷ ⌂ ⌂ ...

Insertion.java > Insertion > main(String[])

```
1  class Insertion{
2      Run | Debug
3      public static void main(String []args){
4          int []arr = {5, 3, 4, 8, 7, 5, 1, 2, 3};      // 1
5          for(int j = 1; j < arr.length; j++){           // n
6              int actual = arr[j];                      // n
7
8              int i = j-1;                            // n
9              while(i >= 0 && arr[i] > actual){        // n^2
10                  arr[i+1] = arr[i];                  // n^2
11                  i--;                           // n^2
12              }
13              arr[i+1] = actual;                     // n
14
15      }                                         }
```

$3n^2 + 9n + 1 = O(n^2)$

△ 0 Go 1.15.5 β

Live Share

✓ java | ✓ Insertion.java

Java

JavaSE-14

⚠ Go Update Available

ⓘ

Análisis de Algoritmos

Fundamentos y ejemplos prácticos sobre
cómo analizar algoritmos de manera
eficiente.