# Guía de Estudio 1: Fundamentos de Problemas Polinomiales y Superpolinomiales

Andrés Oswaldo Calderón Romero, PhD.

October 20, 2025

## 1 Lección 1: Fundamentos de Problemas Polinomiales y Superpolinomiales

En el ámbito del análisis de algoritmos, las complejidades de tiempo "polinomial" y "superpolinomial" representan categorías distintas de eficiencia algorítmica. La complejidad de tiempo polinomial, denotada como $O(n^k)$ para alguna constante $k$, indica algoritmos que pueden completar sus tareas en un tiempo razonable a medida que aumenta el tamaño de la entrada ($n$). Por el contrario, la complejidad de tiempo superpolinomial, que incluye tiempos exponenciales ($O(2^n)$), se considera ineficiente, ya que el tiempo requerido para ejecutar estos algoritmos crece exponencialmente con el tamaño de la entrada.

### 1.1 Complejidad de Tiempo Polinomial

#### 1.1.1 Definición:

Un algoritmo se considera de tiempo polinomial si su tiempo de ejecución está acotado por una función polinomial del tamaño de la entrada.

#### 1.1.2 Ejemplos:

- Tiempo constante ($O(1)$)
- Tiempo logarítmico ($O(\log n)$)
- Tiempo lineal ($O(n)$)
- Tiempo cuadrático ($O(n^2)$)
- Otras funciones polinomiales como $O(n^3)$ o $O(n^4)$.

#### 1.1.3 Eficiencia:

Los algoritmos de tiempo polinomial se consideran eficientes y escalables, lo que significa que pueden manejar entradas relativamente grandes sin una degradación significativa del rendimiento.

## 1.2 Complejidad de Tiempo Superpolinomial

### 1.2.1 Definición:

Un algoritmo es de tiempo superpolinomial si su tiempo de ejecución no está acotado por ninguna función polinomial del tamaño de la entrada.

### 1.2.2 Ejemplos:

- Tiempo exponencial ($O(2^n)$)
- Tiempo factorial ($O(n!)$)
- Cualquier complejidad temporal que crezca más rápido que una función polinomial.

### 1.2.3 Eficiencia:

Los algoritmos de tiempo superpolinomial se consideran generalmente ineficientes y pueden volverse impracticables incluso para tamaños de entrada moderados.

## 1.3 ¿Por Qué Hacer esta Distinción?

La distinción entre complejidad de tiempo polinomial y superpolinomial es fundamental en el diseño y análisis de algoritmos porque:

### 1.3.1 Escalabilidad:

Los algoritmos de tiempo polinomial son escalables, lo que significa que su rendimiento no se degrada significativamente a medida que aumenta el tamaño de la entrada.

### 1.3.2 Practicidad:

Los algoritmos de tiempo superpolinomial pueden volverse intratables computacionalmente incluso para tamaños de entrada relativamente pequeños, lo que los hace poco prácticos para aplicaciones del mundo real.

### 1.3.3 Clases de Complejidad:

Clases de complejidad como P (tiempo polinomial) y NP (tiempo polinomial no determinista) se basan en esta distinción, destacando la importancia de esta separación en la teoría de la computación.

## 1.4 Ejemplos de Problemas Polinomiales y Superpolinomiales

### 1.4.1 Problemas Polinomiales:

- Búsqueda en un arreglo ($O(n)$)
- Ordenamiento de una lista ($O(n \log n)$)

- Encontrar el camino más corto en un grafo ($O(n^2)$)

#### 1.4.2 Problemas Superpolinomiales:

- El Problema del Viajero (NP-difícil, sin solución polinomial conocida)
- Resolver un Sudoku (todos los algoritmos conocidos son superpolinomiales)
- Factorización de números grandes (sin solución polinomial conocida)

# 2 Contenido Adicional

## 2.1 Clase 16: Complejidad: P, NP, NP-completitud, Reducciones [3]

Video de la clase en YouTube
Video de la clase en MIT OpenCourseWare

## 2.2 Resumen del Video

En esta clase sobre NP-completitud, Erik Demaine explica conceptos fundamentales sobre problemas de decisión y sus complejidades computacionales, enfocándose particularmente en la distinción entre **P** (problemas que se pueden resolver en tiempo polinomial) y **NP** (problemas que se pueden verificar en tiempo polinomial no determinista). Utiliza el problema **3SAT** como ejemplo, donde las variables y sus negaciones deben asignarse valores de verdad para satisfacer una fórmula.

Demaine aclara los criterios para identificar problemas NP-completos: deben pertenecer a **NP** y ser también **NP-duros**, lo que indica que son al menos tan difíciles como todos los problemas en NP. Resalta la importancia de las reducciones en tiempo polinomial, que demuestran que resolver un problema puede llevar a la solución de otro, con la profunda implicación de que si existiera un algoritmo de tiempo polinomial para un problema NP-completo, esto podría significar que **P** es igual a **NP**, un tema de gran debate en teoría de la computación.

Para ilustrar aún más la NP-completitud, Demaine creativamente conecta el problema 3SAT con el videojuego *Super Mario Brothers*, construyendo mecánicas de juego que se asemejan a variables lógicas y cláusulas. Esto no solo ejemplifica aplicaciones del mundo real, sino que también involucra principios teóricos en contextos prácticos.

La clase profundiza en conceptos NP-duros con ejemplos como el problema de **tres emparejamientos dimensionales (3DM)** y el problema de recolección de basura, además de resaltar otros problemas NP-completos, como el problema de la suma de subconjuntos, el problema de partición, el empaque de rectángulos y los rompecabezas generalizados. Cada concepto se aborda a través del lente de las reducciones, demostrando cómo están interconectados dentro del ámbito de la complejidad computacional.

## Puntos Destacados

- **Problema 3SAT**: Problema NP-completo fundamental ilustrado a través de mecánicas de videojuegos.
- **Reducciones en Tiempo Polinomial**: Herramienta clave para demostrar NP-dureza y establecer relaciones entre problemas.
- **Discusión P vs. NP**: Implicación crucial de encontrar algoritmos de tiempo polinomial para problemas NP-completos.
- **Emparejamiento Tridimensional (3DM)**: Desafío avanzado en teoría de grafos que muestra la complejidad NP-dura.
- **Gadgets para Recolección de Basura**: Mecanismo innovador que asegura cobertura en reducciones.
- **Problemas de Suma de Subconjuntos y Partición**: Ejemplos clásicos de NP-completitud que ilustran relaciones intrincadas entre problemas computacionales.
- **Rompecabezas Generalizados**: Problema NP-completo que ilustra los desafíos en la transición entre problemas numéricos y no numéricos.

## Puntos Clave

- **Comprensión de NP y NP-Completo**: Demaine enfatiza la diferencia entre **P** y **NP**, crucial para entender los límites de la eficiencia algorítmica.
- **3SAT como Fundamento**: El problema 3SAT es fundamental para demostrar NP-completitud y sirve como una referencia crítica para reducciones a otros problemas NP.
- **Importancia de las Reducciones**: Las reducciones son esenciales en el diseño de algoritmos, ayudando a probar la NP-completitud al transformar efectivamente un problema en otro.
- **Aplicaciones Interactivas**: La aplicación de conceptos teóricos a escenarios prácticos, como la analogía con Super Mario Brothers, ilustra la interacción entre los principios algorítmicos y los videojuegos.
- **Examinando la Dureza en Varios Problemas**: La clase resalta las amplias implicaciones que estos problemas tienen en escenarios del mundo real.
- **Gadgets Innovadores**: El desarrollo de gadgets para crear representaciones específicas de variables y cláusulas destaca el aspecto creativo de la teoría computacional.
- **Direcciones Futuras**: Se anima a los investigadores a explorar nuevos marcos y metodologías para preguntas no resueltas en la complejidad computacional.

Descargar las notas de clase del Prof. Demaine desde aquí.

# 3 Lecturas Recomendadas

1. Capítulo 34 de Cormen et, al [2]. Secciones: 34.1, 34.2, and 34.3.
2. Capítulo 13 de Baase y Van Gelder [1] (En Español). Secciones: 13.1, 13.2, and 13.3.

# References


[1] Sara Baase and Allen Van Gelder. *Algoritmos Computacionales: Introducción al análisis y diseño*. Pearson Educación de México, S.A. de C.V., Mexico City, Mexico, 3rd edition, 2002. Spanish version of the original English work "Computer Algorithms: Introduction to Design and Analysis" published by Addison-Wesley Longman, Inc.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 4th edition, 2022.

[3] Erik Demaine. Lecture 16: Complexity: P, NP, NP-completeness, reductions. https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/lecture-16-complexity-p-np-np-completeness-reductions/, 2015. Accessed: 2025-10-20.