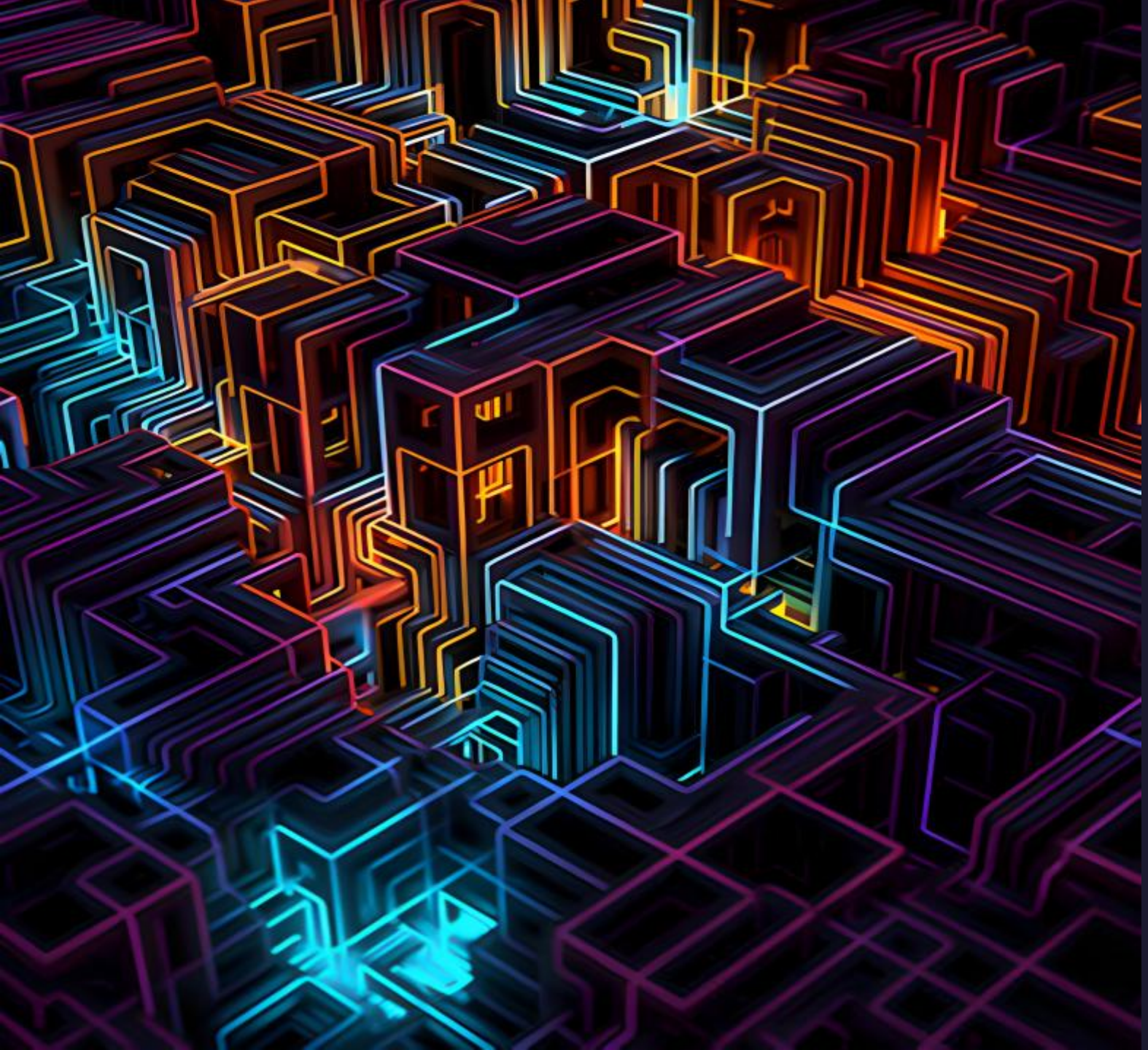


Introducción al curso

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

Sobre mi

- Ingeniería en tecnologías de la información (2014)
- Maestría en ciencias computacionales (2016)
- Doctorado en ciencias en ingeniería y tecnologías computacionales (2021)
- Data scientist (2022)
- Profesora de planta desde 2022

Áreas de interés:

optimización combinatoria, metaheurísticas, computo evolutivo, fitness lanscape analysis, machine learning, data science

Mis cursos:

Programación de estructuras
Implementación de métodos computacionales
→ Análisis y diseño de algoritmos avanzados
Compiladores

Modelación del aprendizaje con IA
Análisis y diseño de sistemas basados en el conocimiento

Soluciones con aplicación de tecnología

Comunicación: Teams

email: valentina.narvaez@tec.mx

Menti

¿Qué aprenderás?

- Técnicas de diseño de algoritmos
- Problemas de decisión, optimización y como resolverlos (o no)
- Computación: la definición matemática de problemas y los algoritmos para resolverlos

¿Qué deberías recordar?

- Estructuras de datos: arreglos, listas, grafos, arboles, pilas, filas…
- Algoritmos de búsqueda y ordenamiento
- C / C++ / Python
- Complejidad
- Recursividad
- Pseudocódigo y matemáticas discretas

¿Por qué es importante?

Es el núcleo de las ciencias computacionales

Competencias

- Fundamentación de sistemas computacionales
  Fundamenta el comportamiento de procesos computacionales y de tecnologías de información con base en principios de ciencias naturales y las matemáticas
- Compromiso con la sustentabilidad
  Aplica estándares, normas y principios de sustentabilidad en el desarrollo de sistemas computacionales y de tecnologías de información.

De área

- Desarrollo de Algoritmos Computacionales
  Soluciona problemas generando algoritmos computacionales eficientes bajo modelos y herramientas de las ciencias computacionales

Disciplinar

- Razonamiento para la Complejidad
  Integra diferentes tipos de razonamiento en el análisis, síntesis y solución de problemas, con disposición al aprendizaje continuo

Transversal

Plan de evaluación

Actividades formativas		**Evidencias integradoras**	
50%		**50%**	
Actividades en clase	20%	Situación problema 1	10%
Actividades fuera de clase	30%	Situación problema 2	10%
		Exámenes	30%

Actividades en clase: se completan durante la sesión, su entrega depende de haber asistido.

Actividades fuera de clase: 1 o 2 por semana

Exámenes: 3, incrementales
En Canvas, con Respondus
Si, incluyen código

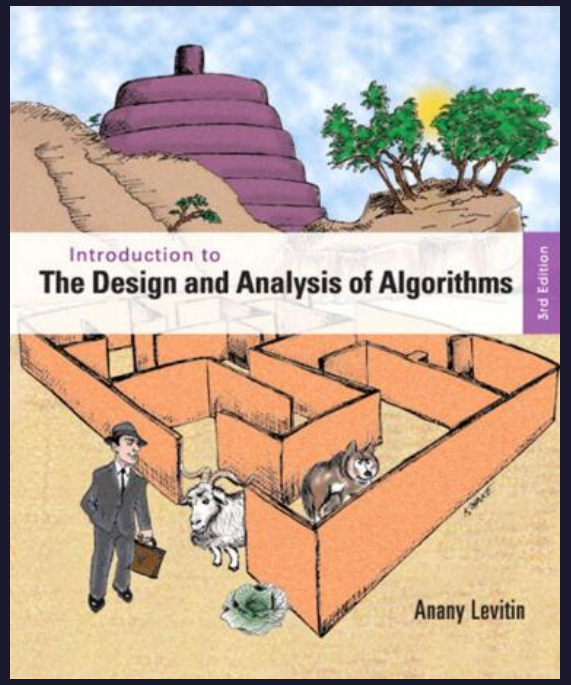
Roadmap

Semanas (aprox)	Temas	SP
1 a 4	Problemas y técnicas de diseño de algoritmos	Situación problema 1: cadenas de ADN
5 a 6	Procesamiento avanzado de strings	
7 a 9	Grafos	Situación problema 2: redes de agua
10 a 11	Geometría computacional	
12 a 14	Búsqueda en optimización	
15	Evaluaciones finales	

Exámenes

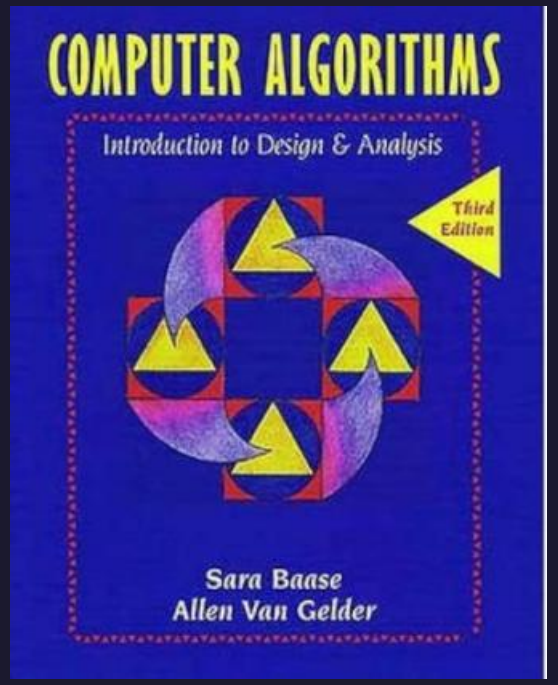
- 1: Problemas y técnicas de diseño y strings
- 2: Temas anteriores + Geometría computacional
- 3: Final. Todos los temas.

Libros

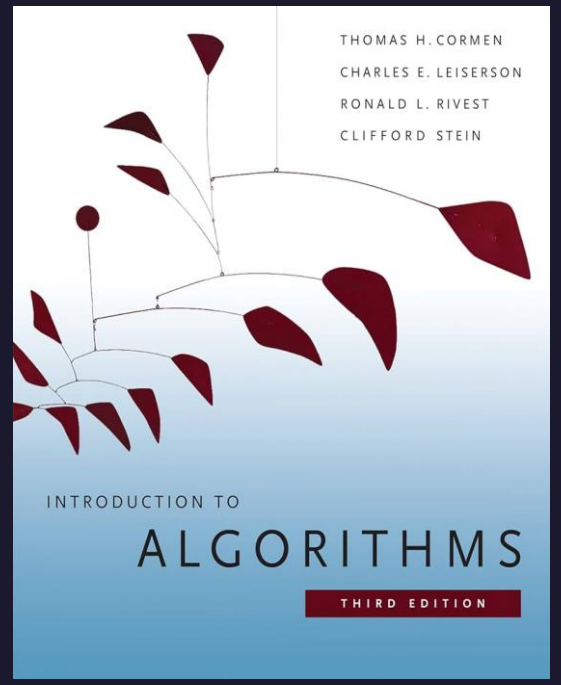


Introduction to The Design and Analysis of Algorithms
Anany Levitin, 3rd Ed

Algoritmos: análisis, diseño e implementación
Luis H. González G., Víctor M. de la Cueva H., Pedro O. Pérez M.



Computer algorithms. Introduction to Design and Analysis
Sara Baase, Allen Van Gelder



Introduction to Algorithms
Thomas H. Cormen, 3rd Ed

Políticas departamentales (y de EIC)

Revisa POLITICAS DEPARTAMENTALES en Canvas

Puntualidad: primeros 10 minutos de la sesión, se capturará la asistencia en la plataforma oficial.

Política de asistencias y competencias:
Asistencia mínima del 80% al total de las sesiones se requiere para las actividades de aprendizaje relacionadas con las subcompetencias. Al acumular mas de 20% de faltas (6 inasistencias), se pierden los puntos.

Justificaciones: Tienes situaciones extraordinarias (salud, concursos, etc.). ¡Contáctame!

Actividades: No hay entregas tardías. Las actividades en clase solo se toman en cuenta si existe registro de asistencia presencial a la sesión y la actividad se entrego en tiempo y forma durante la misma.

Códigos: de autoría propia, con dominio y capacidad de modificarlo, sin IA, sin trivializar con librerías. Honestidad académica ante todo.

Entrega de actividades

Entregas de actividades, incluyendo siempre TODO lo siguiente:

- Links a Colab / Replit
- Tus scripts, usado comentarios para explicar tu código y su complejidad en notación Big-O
- Archivos de entrada (cuando el problema lo requiera)
- Pdfs con capturas de resultados, y/o textos de investigaciones (cuando aplique)

- No entregues .zip ni .rar
- Nunca uses acentos en los scripts, archivos de entrada ni nombres de archivos
- Al trabajar colaborativamente, no olvides incluir todos los nombres
- Cuando haya secciones individuales dentro de un trabajo grupal, indica a quien corresponden

Contacto y asesorías

Contacto preferido: Teams de la clase (link en Canvas)

Horarios de asesoría:
Lunes, martes, jueves - 5:00pm
Miércoles - 10:00am - 12:00pm o 3:00pm - 5:00pm

Mi espacio físico en campus:
Aulas 3, piso 4, área de oficinas de profesores de planta

Mi email: valentina.narvaez@tec.mx
Zoom: https://itesm.zoom.us/my/valentina.narvaez

Técnicas de diseño de algoritmos

Algunas ya las conoces un poco

Fuerza bruta

- Búsqueda secuencial
- Selection-sort
- Bubble-sort

Divide and conquer

- Merge-sort
- Quick-sort

Decrease and conquer

- Insertion sort
- Búsqueda binaria

Programación dinámica

- Recursión terminal

Algoritmos avaros (greedy)

- Alg. de Dikjstra
- Alg. de Kruskal
- Alg. Prim

Otras las conocerás

Branch and bound

- Backtracking
- A* y sus variantes

Heurísticas y metaheurísticas

- Hill-climber
- Local search
- Simmulated annealing

Así que... ¿problemas y algoritmos?

¿Problemas?

Coloquialmente, situaciones adversas:

"La empresa esta perdiendo clientes"
"Nuestra app tiene un bug y crashea"
"El auto no enciende"

En ciencias computacionales:

La teoría de la computación y la complejidad se basan en problemas de decisión, cuya respuesta es si o no

*¿Es 275 * 16 = 156?*

¿Existe el valor x en el arreglo A?

¿Hay una ruta de longitud k entre el nodo A al nodo B de un cierto grafo?

Un puzzle: cruce del puente

Estas cuatro personas deben cruzar un puente oscuro y frágil…

Velocidades:

Ana	1 min	Lola	5 min
Carlos	2 min	Paco	10 min

Reglas:

- Todos inician del mismo lado
- Pueden cruzar máximo dos personas a la vez
- Para cruzar, se necesita llevar la linterna, siempre
- Solo hay una linterna
- Quienes crucen, lo hacen a la velocidad del mas lento

¿Cuál es el menor tiempo para que cruce el grupo completo y en que orden lo hacen?

¿Existe una sol. de costo $k <= 19$? ¿$k \leq 18$? ¿$k \geq 17$?

Así que... ¿problemas y algoritmos?

Algoritmo

Secuencia finitas de pasos para resolver un problema.

Dado un problema como entrada, produce la solución al problema como salida

Los algoritmos tienen complejidad

- Espacial: ¿cuánto espacio requiere?
- Temporal: ¿cuánto tiempo toma? / ¿cuántas operaciones realiza?

Menor complejidad, mas eficiencia

Importante: Nunca medir complejidad en números absolutos, si no como función del tamaño del problema

La complejidad captura el orden de crecimiento

Notaciones asintóticas

Big-O (O-grande):

Todas las funciones $f(n)$ que crecen **no mas rápido** que $c * g(n)$ son $O(g(n))$

$f(n) \in O(g(n))$, si $f(n) \leq c * g(n)$,

donde c es constante

El algoritmo toma máximo $g(n)$, por alguna constante despreciable c

Esta notación es la mas usual

Big-θ (Gran-theta):

Todas las funciones $f(n)$ que crecen **tan rápido** como $c * g(n)$ son $\theta(g(n))$

$$f(n) \in \theta(g(n)),$$
si $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

donde c_1 y c_2 son constantes

El algoritmo toma entre $g(n) * c_1$ y $g(n) * c_2$

Big-Ω (Gran-omega):

Todas las funciones $f(n)$ que crecen **al menos tan rápido** como $c * g(n)$ son $\Omega(g(n))$

$$f(n) \in \Omega(g(n)),$$

si $f(n) \geq c * g(n)$

donde c es constante

El algoritmo toma mínimo $g(n)$, por una constante despreciable c

Analizando algoritmos

❖TIP: revisa las variables de control ¿Qué función captura mejor cuántas vueltas da el ciclo?

- Un ciclo, incrementos constantes

```
23    for (i = 0; i < n ; i++)
24    {   // Some code
25
26    }
27
```

- Dos ciclos anidados, incrementos constantes

```
29    for (i = 0; i < n ; i++)
30    {   for (j = 0; j < n ; j++)
31        {   // Some code
32
33        }
34    }
35
```

- Ciclos con división / multiplicación alterando la variable de control

```
36    for (i = 0; i< n; i++ )
37    {   j = n;
38        while (j > 1)
39        {   // Some code
40
41            j = j / 2;
42        }
43    }
44
```

Analizando algoritmos

❖TIP: revisa las variables de control ¿Qué función captura mejor cuántas vueltas da el ciclo?

- Un ciclo, incrementos constantes

```
23    for (i = 0; i < n ; i++)
24    {   // Some code
25
26    }
27
```

$O(n)$

- Dos ciclos anidados, incrementos constantes

```
29    for (i = 0; i < n ; i++)
30    {   for (j = 0; j < n ; j++)
31        {   // Some code
32
33        }
34    }
35
```

$O(n^2)$

- Ciclos con división / multiplicación alterando la variable de control

```
36    for (i = 0; i< n; i++ )
37    {   j = n;
38        while (j > 1)
39        {   // Some code
40
41            j = j / 2;
42        }
43    }
44
```

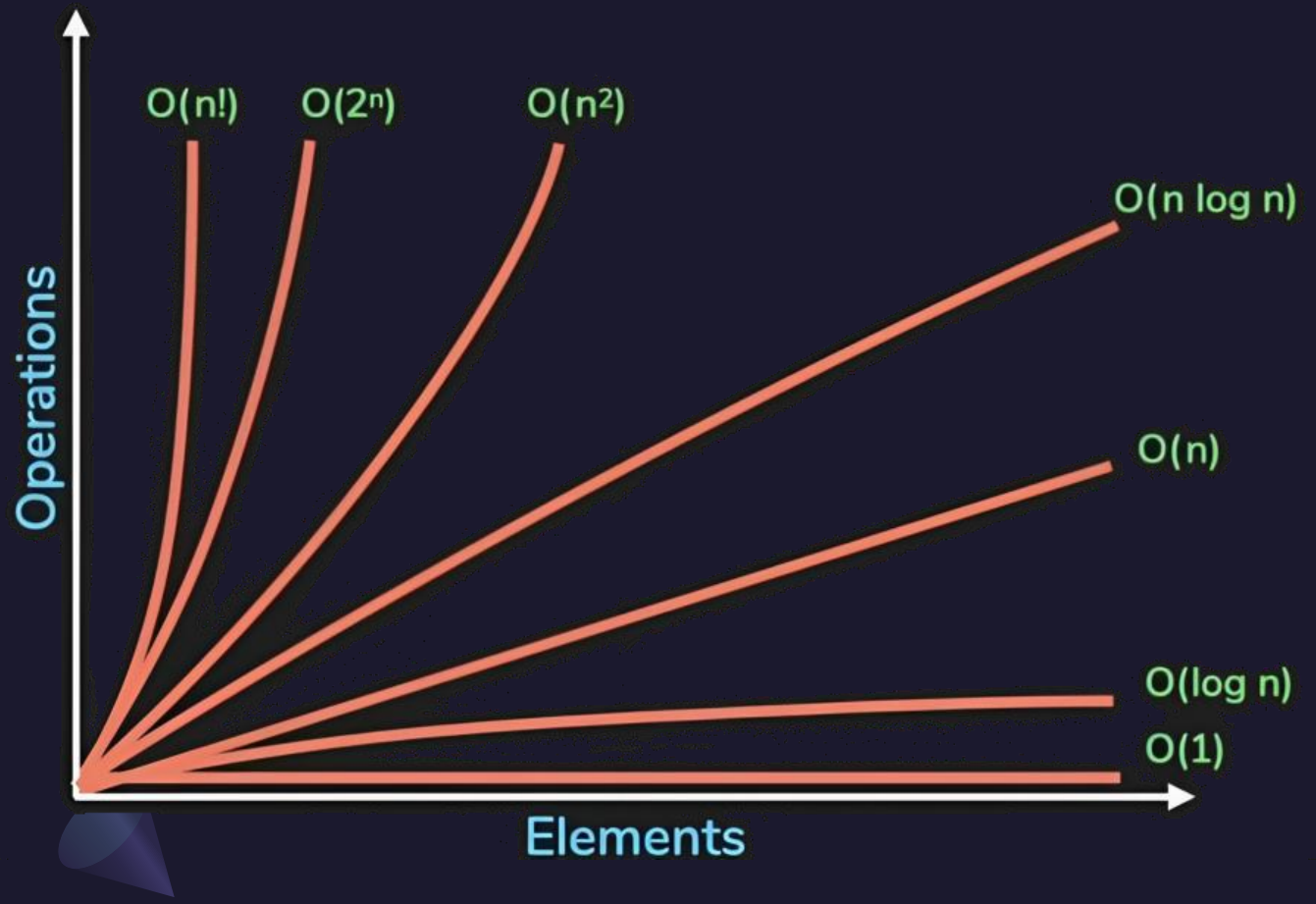
$O(n \log n)$

Comparando complejidades

Rule of thumb: Identifica el término de mayor orden

Mayor complejidad

$O(1)$	Constante
$O(\log n)$	Logarítmico
$O(n)$	Lineal
$O(n \log n)$	Linearítmico
$O(n^2)$	Cuadrático
$O(n^3)$	Cubico
$O(n^m)$	Polinómico
$O(2^n)$	Exponencial
$O(n!)$	Factorial



Teoría de la complejidad: P vs NP

La dificultad de un problema depende de la complejidad del algoritmo mas eficiente que lo resuelva

Menos eficiente

$O(1)$	Constante	Tratable
$O(\log n)$	Logarítmico	
$O(n)$	Lineal	
$O(n \log n)$	Linearítmico	
$O(n^2)$	Cuadrático	
$O(n^3)$	Cubico	
$O(n^m)$	Polinómico	
$O(2^n)$	Exponencial	Intratable
$O(n!)$	Factorial	

Los problemas que tienen algoritmos conocidos de tiempo polinómico (poly-time) o menor, se consideran problemas tratables

Clase P: problemas que una maquina de Turing determinista resuelve en poly-time

¿Determinista? Si hay n operaciones por hacer, se hacen una a la vez

¿No-determinista? ? Si hay n operaciones por hacer, se hacen todas a la vez.

Teoría de la complejidad: P vs NP

La dificultad de un problema depende de la complejidad del algoritmo mas eficiente que lo resuelva

Menos eficiente

$O(1)$	Constante	Tratable
$O(\log n)$	Logarítmico	
$O(n)$	Lineal	
$O(n \log n)$	Linearítmico	
$O(n^2)$	Cuadrático	
$O(n^3)$	Cubico	
$O(n^m)$	Polinómica	
$O(2^n)$	Exponencial	Intratable
$O(n!)$	Factorial	

Hay problemas para los que se cree imposible que existan algoritmos de complejidad menor a exponencial (exp-time), por lo que se consideran problemas intratables

Clase NP: incluye problemas que una (mágica e imposible) máquina de Turing no-determinista resolvería en poly-time… pero creemos que una determinista no puede

O sea, creemos que P != NP

Problemas

¿Cuáles son esos problemas intratables?

De decisión	Sobre verificar si una solución es la "correcta", o si una solución que cumpla ciertas características existe	
De satisfactibilidad	Sobre encontrar una asignación para variables, que satisfaga ciertas condiciones	SAT
De optimización • Combinatoria • Numérica	Sobre encontrar una asignación para variables (discretas o continuas), que resulte en valores mínimos o máximos	TSP Coloreado de grafos Knapsack

¿Porque son tan difíciles?

En general, porque tienen muchísimas soluciones posibles, y no podemos saber que una es la mejor sin tener que revisar todas

Y hacerlo tomaría millones de años…

Problemas de optimización

Los puedes entender como generalizaciones de los problemas de decisión

Muy relevantes por sus aplicaciones prácticas

- Rutas y schedules
- Admin. de recursos y logística
- Construcción de infraestructura
- Secuencias de proteínas
- Estudio de epidemias
- Calibración de modelos de IA (parámetros, pesos en redes neuronales, etc.)

Problemas PO y NPO

Lo mismo que P y NP, pero para problemas de optimización, en lugar de decisión

Problemas NP-Hard

¿Recuerdas que los problemas NP-Complete son problemas de decisión en NP que son todos equivalentes entre si?

Los problemas NP-Hard están en NPO y son equivalentes entre si

Problemas de optimización

Son problemas donde se busca construir la mejor solución posible

¿Cuál es la **ruta mas corta** del nodo A al nodo B?

¿Cuál es la **mayor cantidad de valor** que pueden guardarse en un almacén con cierta capacidad?

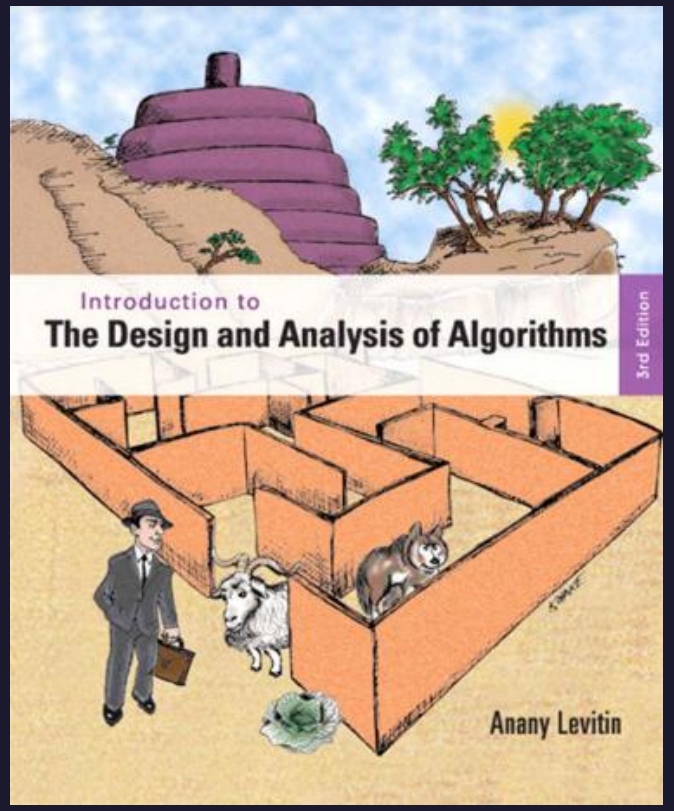
¿Cuál es la **combinación de pesos para el menor error** en una red neuronal?

¿Cómo se define que es "mejor"?

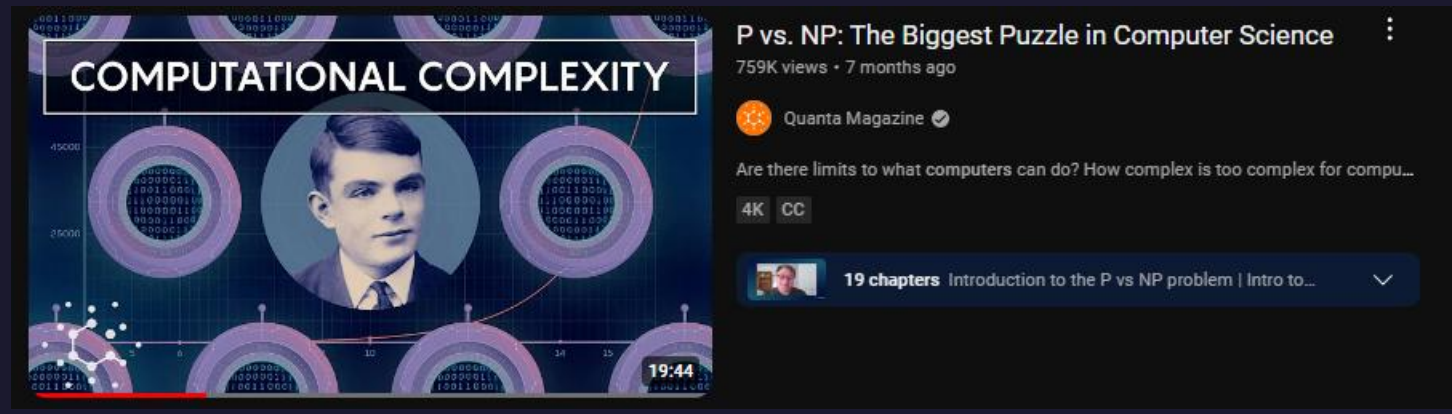
Lecturas y otros recursos sugeridos

Lecturas:

Levitin Capitulo 2, hasta la sección 2.2 (inclusive)



https://youtu.be/pQsdygaYcE4?si=6DGNk2VMZS9czfbZ



https://youtu.be/YX40hbAHx3s?si=KCTbu-m04xixosg0