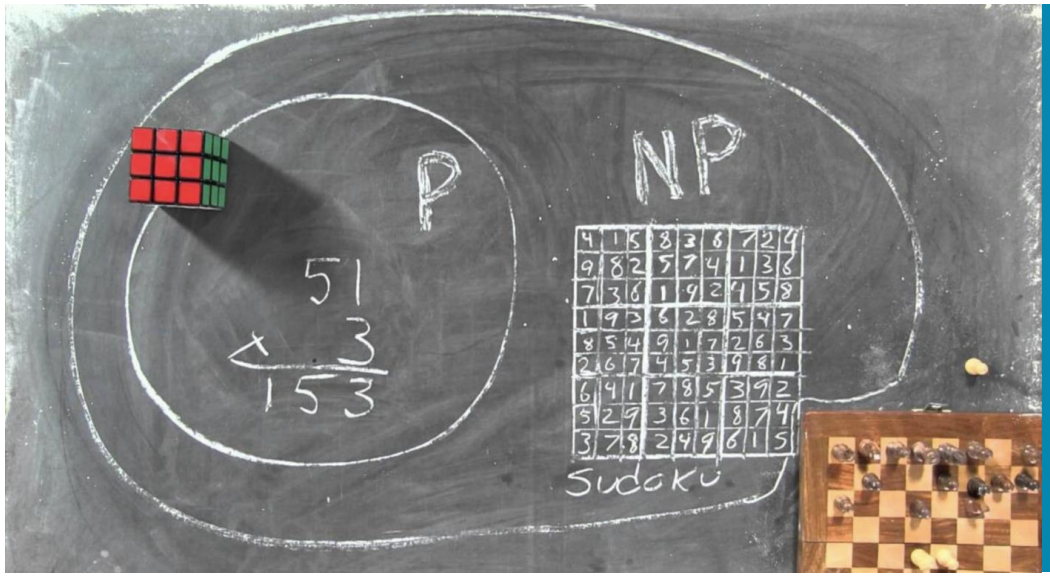



# Complejidad Algorítmica

**Unidad 2: Algoritmos voraces, programación dinámica y problemas P-NP**

**Módulo 15: Problemas de Tipo P-NP**

Ing. Patricia Reyes Silva
pcsiprey@upc.edu.pe

# Complejidad Algorítmica

Semana 15 / Sesión 1

**Objetivo**
Comprender los diferentes tipo de problemas computables.

**MÓDULO 15: Problemas de Tipo P- NP**

## Contenido

1. Importancia de definir y clasificar los tipos de problemas
2. Tipos de Problemas computacionales
3. Teoría de la Complejidad: Definiciones de Problemas
    - P
    - NP
    - NP-Hard
    - NP-Complete

## Preguntas / Conclusiones

# 1. Importancia de definir y clasificar los tipos de problemas

- Como informáticos, trabajamos a diario para construir y desarrollar nuevas soluciones a los problemas que enfrentamos.
- Un paso importante en este proceso es definir el tipo de problema y clasificarlo.
- Este paso nos lleva a saber si el problema que estamos tratando de resolver es solucionable o no antes de continuar con el proceso de desarrollo.
- Nos valemos de:

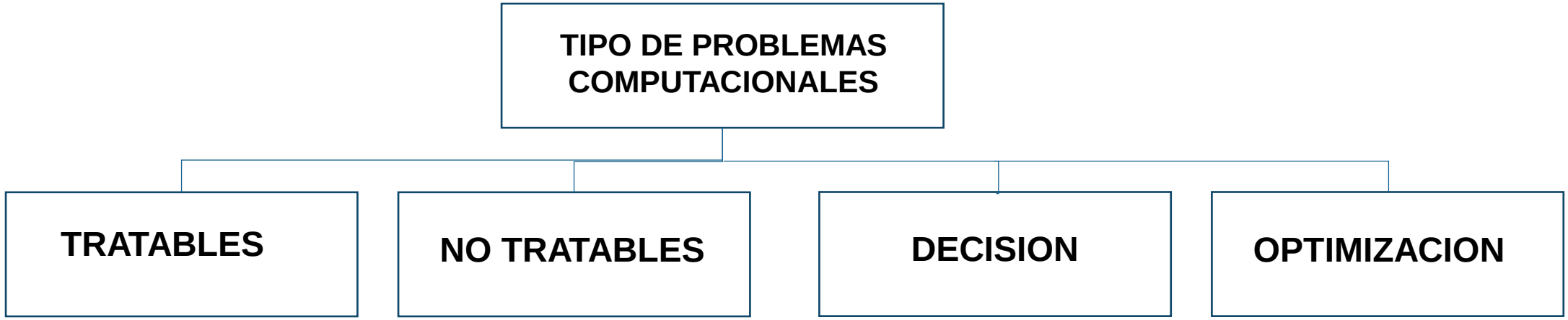
> La **teoría de la complejidad** (como subcampo de la informática) que se encarga de clasificar los problemas en un conjunto de categorías que especifican como debiera ser su solución.

- Y en la **teoría de la complejidad**, nos encontraremos con términos tales como:
  - ❑ “este es un problema NP ”
  - ❑ “este problema se puede resolver en tiempo P ”,
  - ❑ “¡ NP no es igual a P !”

Entendamos que significan estos términos…

# 2. Tipos de Problemas computacionales

- Un primer acercamiento a los problemas computacionales:




- Problemas que **se pueden solucionar en un tiempo razonable** (complejidad o **tiempo polinomial**).

- Algunos problemas son intratables, a medida que crecen, **no podemos resolverlos en un tiempo razonable**.

• Un problema de decisión trata de **responder una pregunta de sí/no**

- Un problema de optimización trata de encontrar una solución óptima (**minimizar / maximizar un resultado**)

# 2. Tipos de Problemas computacionales

**PROBLEMAS TRATABLES**

- Problemas que **se pueden solucionar en un tiempo razonable** (tienen una complejidad o tiempo polinomial).

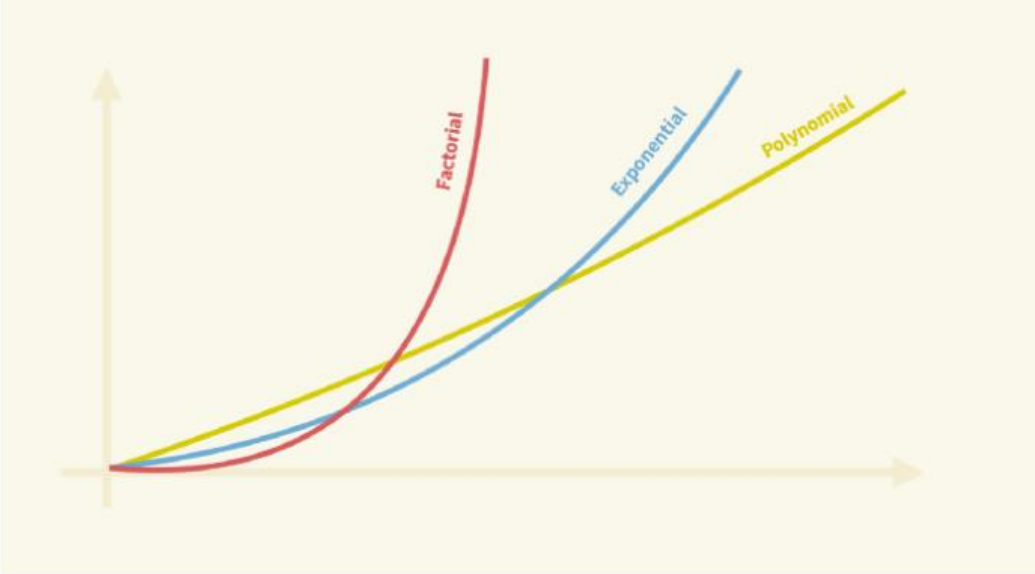
Tiempo razonable = tiempo polinomial


- Hemos aprendido en este curso, que para una entrada de tamaño n, el tiempo de ejecución en el peor de los casos es **O(nk)** para alguna constante **k**.
- La complejidad (eficiencia) de los algoritmos puede representarse por la notación asintótica Big O:
  - **Complejidades**: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(2^n)$, $O(n^n)$, $O(n!)$
    - ❑ **En Tiempo polinomial:** **$O(n^2)$, $O(n^{33})$, $O(1)$, $O(n \lg n)$**
    - ❑ **En tiempo no-polinomial:** $O(2^n)$, $O(n^n)$, $O(n!)$

¿Todos los problemas se pueden resolver en tiempo polinomial? **No**

# 2. Tipos de Problemas computacionales

**PROBLEMAS TRATABLES**



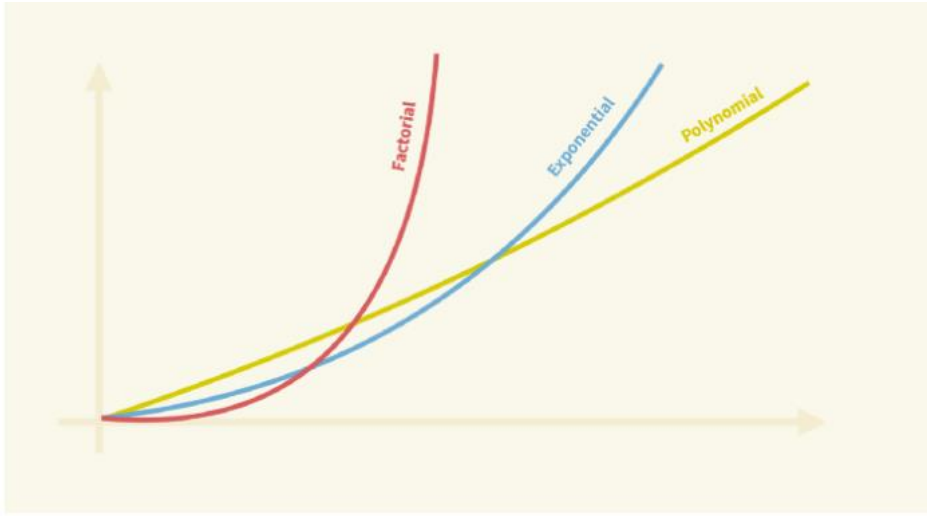

La complejidad temporal de un algoritmo se describe como una función asintótica que depende del tamaño de entrada del algoritmo. Se hace una distinción principal entre las funciones de complejidad **Factorial** , **Exponencial** y **Polinomial** .

**RECORDEMOS:**

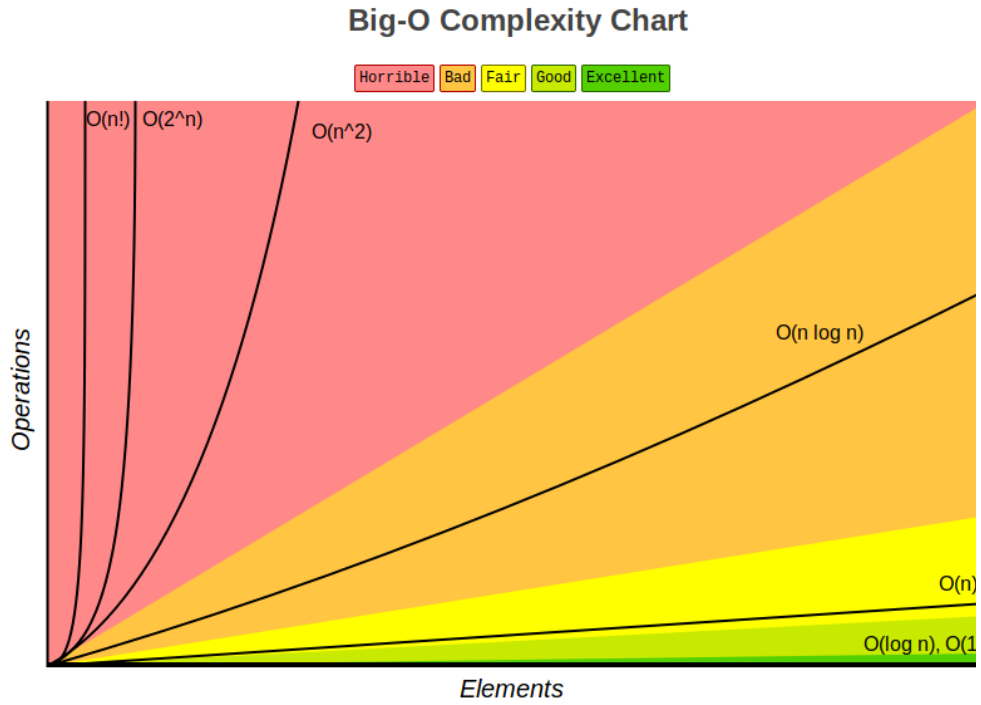
- La complejidad del tiempo se refiere a cuántos pasos se necesitan para resolver un problema y cómo la cantidad de pasos requeridos aumenta con el tamaño del problema.
- Dado un algoritmo, la complejidad temporal del algoritmo se describe como una función asintótica que depende del tamaño de entrada del algoritmo.
- Se utiliza para clasificar los algoritmos de acuerdo con cómo crece su número de pasos o requisitos de espacio a medida que crece el tamaño de entrada.

# 2. Tipos de Problemas computacionales

## PROBLEMAS TRATABLES




La complejidad temporal de un algoritmo se describe como una función asintótica que depende del tamaño de entrada del algoritmo. Se hace una distinción principal entre las funciones de complejidad **Factorial** , **Exponencial** y **Polinomial** .



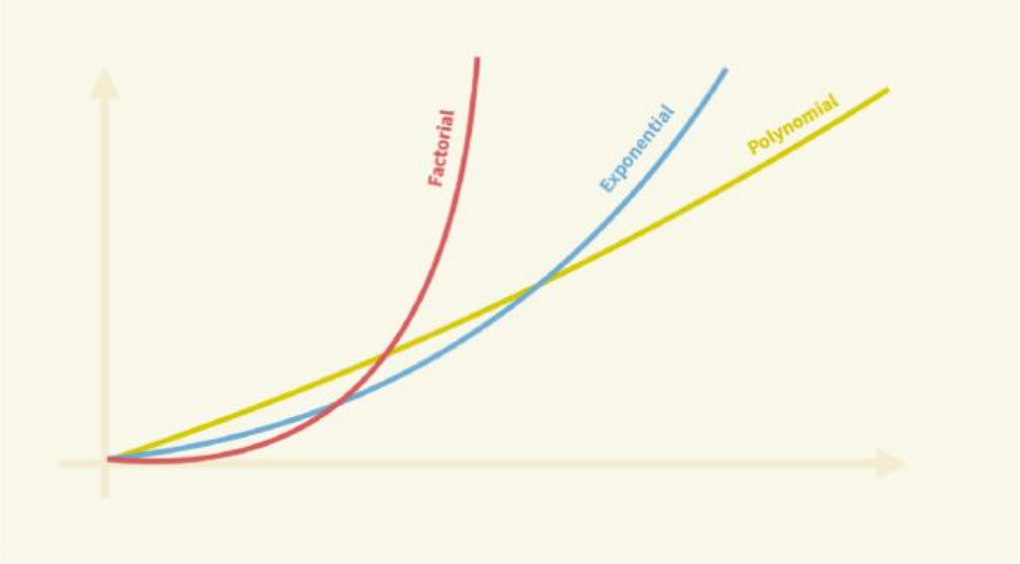

| | |
|---|---|
| O(*1*) | Orden constante |
| O(*log n*) | Orden logarítmico |
| O(*n*) | Orden lineal |
| O(*n log n*) | Orden cuasi-lineal |
| O($n^2$) | Orden cuadrático |
| O($n^3$) | Orden cúbico |
| O($n^a$) | Orden polinómico |
| O($2^n$) | Orden exponencial |
| O(*n!*) | Orden factorial |

Órdenes de complejidad comunes.

**PROBLEMAS TRATABLES** ➡ **En Tiempo polinomial: O($n^a$) O($n^2$), O($n^3$), O(1), O(n lg n)**

# 2. Tipos de Problemas computacionales

**PROBLEMAS TRATABLES**




La complejidad temporal de un algoritmo se describe como una función asintótica que depende del tamaño de entrada del algoritmo. Se hace una distinción principal entre las funciones de complejidad **Factorial**, **Exponencial** y **Polinomial**.

Los algoritmos que tienen una complejidad de tiempo polinomial se denominan **"eficientes".**

- Se distingue entre algoritmos que tienen una Complejidad de Tiempo Polinomial, donde su función de complejidad es un Polinomio, a aquellos que tienen una función de complejidad más radical.
- Se considera que el crecimiento polinomial es más moderado que otros, en el sentido de que grandes cambios en el tamaño de entrada no causarían un incremento radical en los pasos requeridos
- Un **polinomio** es una construcción que involucra solo las operaciones de suma, resta, multiplicación y exponentes enteros no negativos, por lo que nunca presenta un crecimiento exponencial o factorial
- La elección del tiempo polinomial para representar el cómputo eficiente parece arbitraria; sin embargo, se ha justificado a lo largo del tiempo.

# 2. Tipos de Problemas computacionales

**PROBLEMAS DE OPTIMIZACIÓN/DECISIÓN**

**Problemas de optimización:**

Es aquel que pregunta:
"¿Cuál es la solución óptima al problema X?"

Ejemplos:
- 0-1 Mochila
- Mochila fraccionada
- Árbol de expansión mínimo (MST)


**Problemas de decisión:**

Es uno con respuesta sí/no

Ejemplo:

• ¿Tiene un grafo G un MST de peso W?

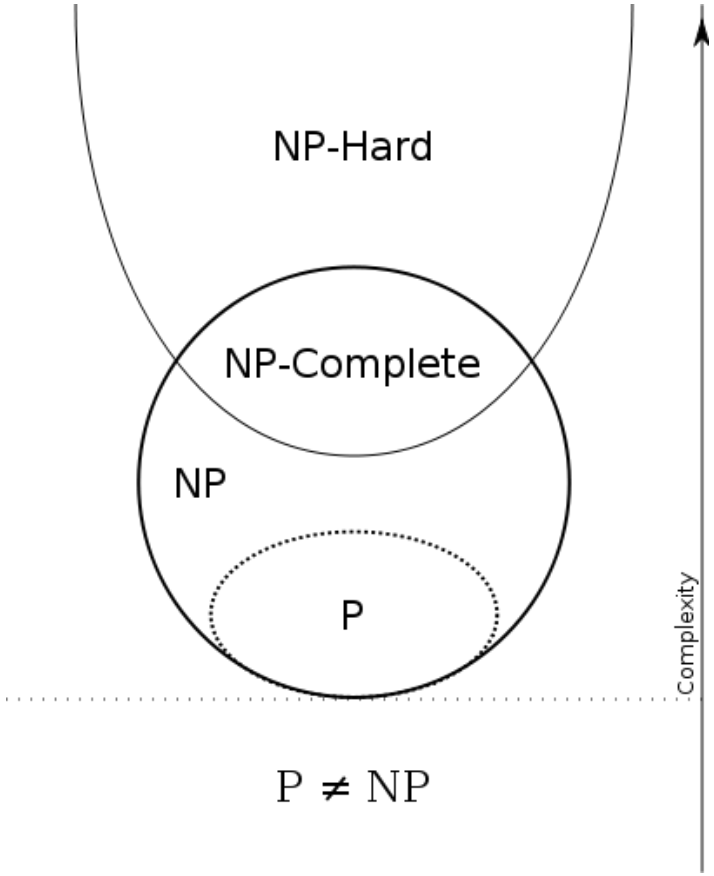

Muchos problemas tendrán versiones de decisión y optimización.

Ej.: Problema del viajero

- **optimización:** encontrar el ciclo hamiltoniano de peso mínimo
- **decisión:** ¿hay un ciclo hamiltoniano de peso k?

# 3. Teoría de la Complejidad: Definición de Problemas

**Definición de Problemas**




**¿Qué significado tiene cada clase de problema?**

# 3. Teoría de la Complejidad: Definición de Problemas

## Problemas P

- **P:** es la clase de problemas que tienen algoritmos deterministas de tiempo polinomial.

**Algoritmos deterministas:**

- Un algoritmo determinista es (esencialmente) uno que siempre calcula la respuesta correcta.
- Una computadora, tal como la conocemos, es una maquina determinista.

**Tiempo Polinomial:**

- Es decir, son resolubles (computables) en O(p(n)), donde p(n) es un polinomio sobre n.

Ejemplos de problemas en P:

- Problema de la Mochila MST
- Algoritmos de clasificación y búsqueda

# 3. Teoría de la Complejidad: Definición de Problemas

## Problemas NP

- **NP:** es la clase de problemas que tienen **algoritmos NO deterministas** de **tiempo polinomial**.

**Algoritmos NO deterministas:**

- Un **algoritmo NO determinista** es (esencialmente) uno que NO siempre calcula la respuesta correcta
- Una **computadora NO determinista** es aquella que puede “adivinar” la respuesta o solución correcta.
- Pensemos una computadora no determinista como una máquina paralela que puede generar libremente una cantidad infinita de procesos.

**Tiempo Polinomial:**

- Es decir, son resolubles (computables) en O(p(n)), donde p(n) es un polinomio sobre n.

**NP** significa = **“No determinista”** + **“Tiempo polinomial”**

# 3. Teoría de la Complejidad: Definición de Problemas

## Problemas NP

**Ejemplos NP:**

- El problema de la Mochila
- MST
- Otros:
    - El problema del vendedor viajero (TSP)
    - La coloración de gráficos
    - El problema de decidir si una determinada fórmula booleana es satisfactoria.

# 3. Teoría de la Complejidad: Definición de Problemas

## Problemas NP-Hard (NP-Difícil) vs NP-Completos

¿Qué significa **NP-Hard (NP-Dificil)**?

- Muchas veces podemos resolver un problema reduciéndolo a un problema diferente.
- Podemos **reducir el Problema B al Problema A** si, dada una solución al Problema A, puedo construir fácilmente una solución al Problema B. (En este caso, "fácilmente" significa "en tiempo polinomial")

> Un problema es **NP-difícil** si todos los problemas en NP se reducen a tiempos polinómicos.

¿Qué significa **NP-Completo**?

> Un problema es **NP-completo** si el problema **es a la vez NP-Hard y NP**.

Ejemplo: Problema del Clique

# 3. Teoría de la Complejidad: Definición de Problemas

## Reducciones

- Un problema R puede reducirse a otro problema Q si cualquier instancia de R puede reformularse como una instancia de Q.
- Esta reformulación se llama **transformación**.
- Intuitivamente: si R se reduce en tiempo polinomial a Q, R "no es más difícil de resolver" que Q

# 3. Teoría de la Complejidad: Definición de Problemas

## Resumen – Clasificación de Problemas Computacionales

| P (Fáciles) | NP | NP-Hard (Difícil) | NP-Completo |
|---|---|---|---|
| • **P** es un conjunto de problemas que pueden ser resueltos por una máquina de Turing (algoritmo) determinista en tiempo polinomial.<br>• **P** es un subconjunto de NP (cualquier problema que pueda resolverse mediante máquina determinista en tiempo polinomial también puede resolverse por máquina no determinista en tiempo polinomial).<br>• Pero **P≠NP**. | • **NP** es un conjunto de problemas que pueden ser resueltos por una máquina de Turing no determinista en tiempo polinomial.<br>• Algunos problemas se pueden traducir unos a otros de tal manera que una solución rápida a un problema automáticamente nos daría una solución rápida al otro. | • Un problema es **NP-difícil** si un algoritmo para resolverlo puede traducirse en uno para resolver cualquier problema **NP** (tiempo polinomial no determinista).<br>• Por lo tanto, **NP-difícil** significa "al menos tan difícil como cualquier problema **NP**", aunque, de hecho, podría ser más difícil. | • Algunos problemas se pueden traducir unos a otros de tal manera que una solución rápida a un problema automáticamente nos daría una solución rápida al otro.<br>• Estos son los problemas **NP-Completos**: que son a la vez NP y NP-Hard, en los que todos los problemas de NP se pueden traducir, y una solución rápida a dicho problema automáticamente nos daría una solución rápida a todos los problemas de NP. |

# 3. Teoría de la Complejidad: Definición de Problemas

**Resumen – Clasificación de Problemas Computacionales**

| | P | NP | NP-complete | NP-hard |
|---|---|---|---|---|
| Solvable in polynomial time | ✓ | | | |
| Solution verifiable in polynomial time | ✓ | ✓ | ✓ | |
| Reduces any NP problem in polynomial time | | | ✓ | ✓ |

# Conclusiones

1. La **teoría de la complejidad** es uno de los subcampos importantes de las Ciencias de la Computación que se ocupa de clasificar los problemas en función del tiempo que necesitan para resolverse.

2. A la teoría de la complejidad no le importa mucho el algoritmo que usas para resolver un problema específico, sino el problema en sí.

3. Tener los conocimientos necesarios para categorizar y comprender el tipo de estos problemas puede ayudarnos a encontrar mejores respuestas o reducir los problemas a problemas de **tipo P** para satisfacer nuestras necesidades.

4. Según la **teoría de la complejidad,** los algoritmos que tienen una complejidad de tiempo polinomial se denominan **"eficientes"**.

**P =** conjunto de problemas que se pueden resolver en tiempo polinomial.
**Ejemplo:** Problema de la Mochila, …

**NP =** conjunto de problemas para los cuales se puede verificar una solución en tiempo polinomial.
**Ejemplo:** Problema de la Mochila,…, TSP etc.

Ciertamente, P no es lo mismo que NP… **P ≠ NP**

**P:** es la clase de problemas que tienen algoritmos deterministas de tiempo polinomial.

**NP:** es la clase de problemas que tienen **algoritmos NO deterministas** de **tiempo polinomial**.

# PREGUNTAS

Dudas y opiniones