COMPLEJIDAD COMPUTACIONAL

La Teoría de la Complejidad estudia la manera de clasificar algoritmos como buenos o malos y de clasificar problemas de acuerdo a la dificultad inherente de resolverlos.

ALGORITMO: Serie finita de pasos para resolver un problema.

PREGUNTAS ¿Para todos los problemas, existe al menos un algoritmo?

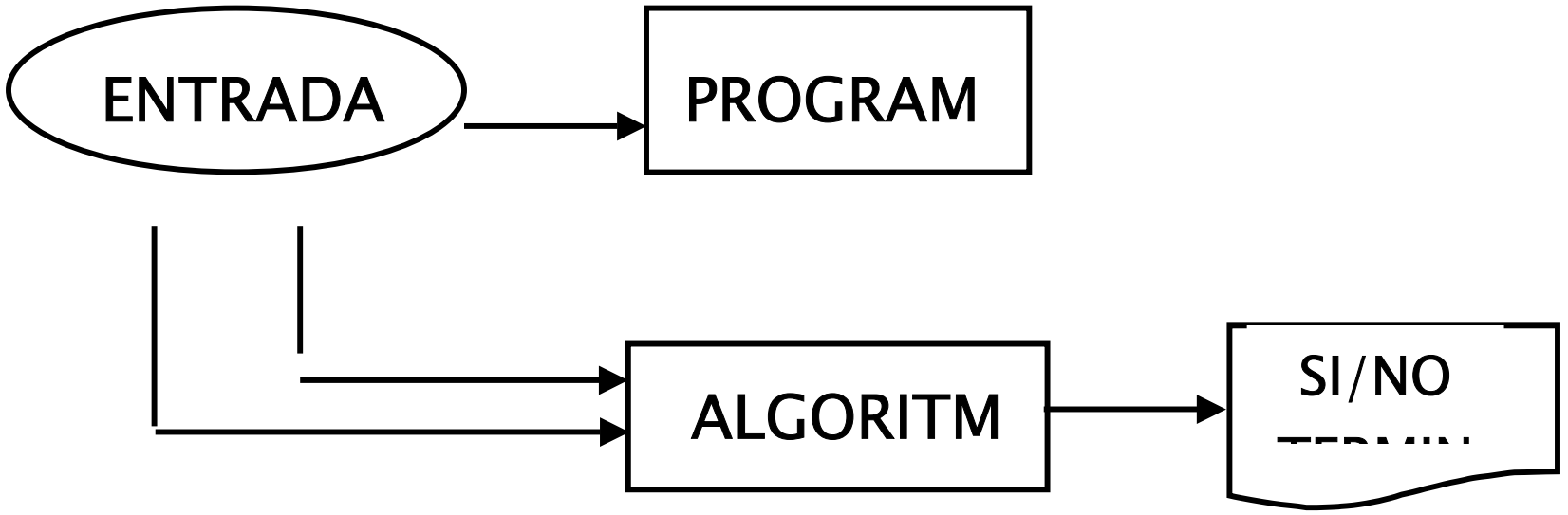
Si existen varios algoritmos para un problema, ¿Cómo hacer una selección en términos de eficiencia?

¿Cómo pueden ser clasificados los problemas mismos en cuanto a la dificultad inherente de resolverlos?

¿Para todos los problemas existe al menos un algoritmo?

Ejemplo:

HALTING PROBLEM: Sea un programa cualquiera con sus entradas, determinar si el programa termina.


ENTRADA
PROGRAM
ALGORITM
SI/NO

MEDIDAS de COMPLEJIDAD

PEOR CASO $f(n) \in O(g(n))$

Si $f(n) \leq c(g(n))$

CASO PROMEDIO $f(n) \in \Theta(g(n))$

si $c(g(n)) < f(n) < c'(g(n))$

MEJOR CASO

$f(n) \in \Omega(g(n))$, Si $f(n) \geq c(g(n))$

COMPLEJIDAD TEMPORAL

La complejidad temporal depende del tamaño de las entradas n.

EJEMPLO: $t(n) = 60n^2 + 5n + 1$ PEOR CASO

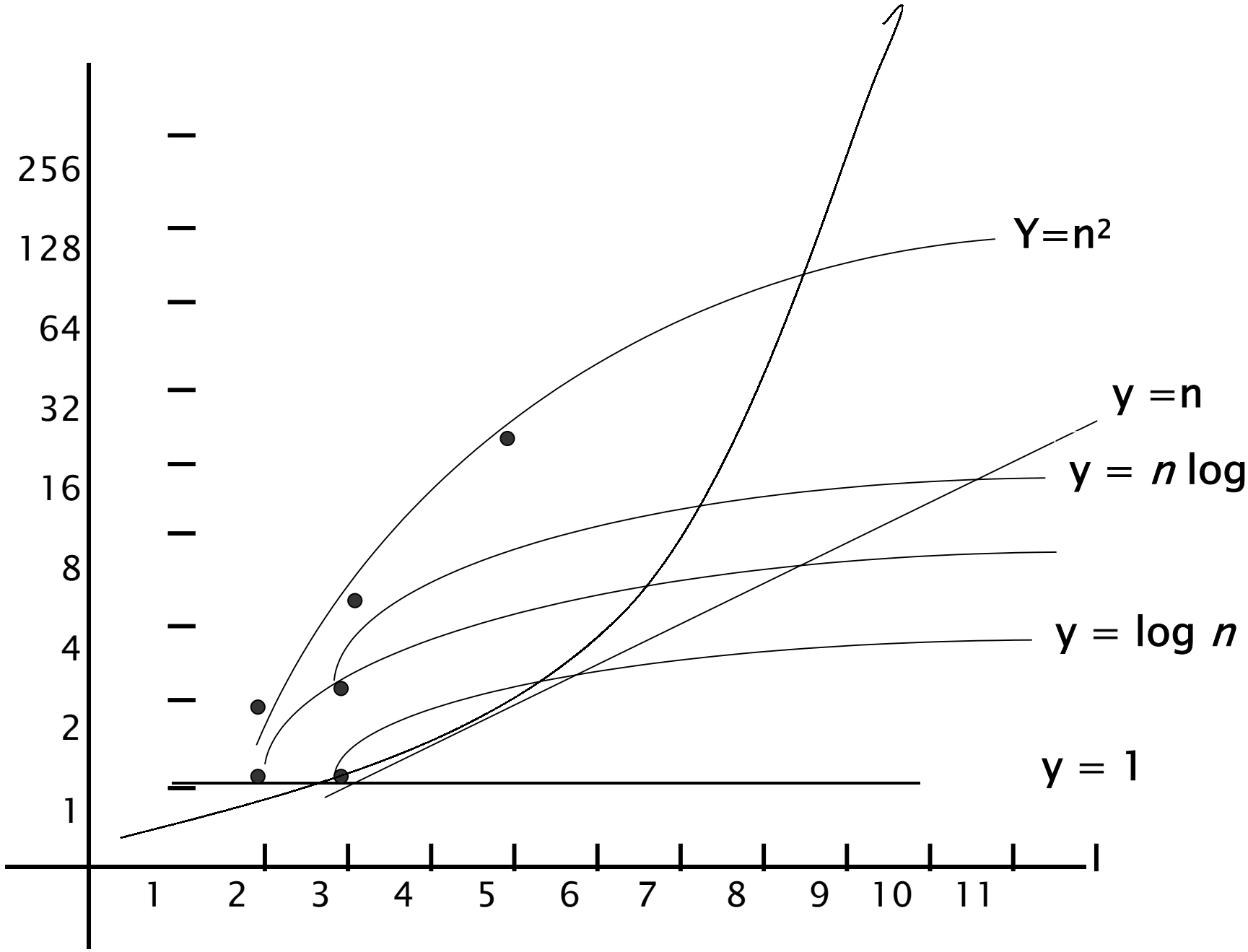
n	t(n) = $60n^2+5n+1$	$60n^2$
10	6,051	6,000
100	600,501	600,000
1,000	60,005,001	60,000,000
10,000	6,000,050,001	6,000,000,000

$\therefore$ **Complejidad : n^2**

(Se dice que t(n) es del orden de n^2)

Notación con O	*Nombre*

$O(1)$	Constante
$O(\log \log (n))$	Log log
$O(\log (n))$	Logarítmica
$O(n)$	Lineal
$O(n \log (n))$	$n \log n$
$O(n^2)$	Cuadrática
$O(n^3)$	Cúbica
$O(n^m)$	Polinomial
$O(m^n)$, $m \geq 2$	Exponencial
$O(n!)$	Factorial


256
128
64
32
16
8
4
2
1
1 2 3 4 5 6 7 8 9 10 11
Y=n²
y =n
y = n log
y = log n
y = 1

BUENOS Y MALOS ALGORITMOS

Convención 1: Todos los algoritmos, desde constantes hasta polinomiales, son polinomiales

Convención 2 : Todos los algoritmos exponenciales y factoriales, son exponenciales

*Convención 3 :*Los algoritmos polinomiales son "buenos" algoritmos

Los algoritmos exponenciales son "malos" algoritmos

CRECIMIENTO DE FUNCIONES

Función	*Valores Aproximados*		
n	10	100	1000
$n \log n$	33	664	6640
n^3	1,000	1,000,000	1,000,000,000
$10^6\ n^8$	1,014	10^{22}	10^{30}
2^n	1,024	1.27×10^{30}	--
$n^{\log n}$	2,099	1.93×10^{13}	--
$n!$	3,628,800	10^{158}	--

EFECTO DE LA TECNOLOGÍA

Función	*Tamaño de la Instancia Solucionada en 1 Día*	*Tamaño de Instancia Solucionada en un Día en una Computadora 10 Veces Más Rápida*
n	10^{12}	10^{13}
$n \log$	0.948×10^{11}	0.87×10^{12}
n^2	10^6	3.16×10^6
n^3	10^4	2.15×10^4
$10^8 n^4$	10	18
2^n	40	43
10^n	12	13
$N^{\log n}$	79	95
$n!$	14	15

ALGORITMOS DETERMINÍSTICOS Y NO-DETERMINÍSTICOS

Algoritmo Determinístico

Tiene la propiedad de que el resultado de cada operación, se define en forma única

Algoritmo No-Determinístico

Tiene la propiedad de que el resultado de una o varias operaciones, se determina dentro de un conjunto especificado de posibilidades

PROBLEMAS POLINOMIALES Y NO-POLINOMIALES

Problema Polinomial

Existe un algoritmo determinístico, capaz de resolverlo en tiempo polinomial

Problema No-Polinomial

No existe un algoritmo determinístico, capaz de resolverlo en tiempo polinomial

PROBLEMAS DECIDIBLES:

REQUIEREN UNA RESPUESTA (SI/NO).

PROBLEMAS INDECIDIBLES: NO EXISTEN ALGORITMOS CAPACES DE RESOLVERLOS -> PROBLEMAS INTRATABLES

EJEMPLO: ES IMPOSIBLE ESPECIFICAR UN ALGORITMO QUE SEA CAPAZ DE DECIDIR SI/NO DADO UN PROGRAMA ARBITRARIO, ESTE TERMINE.

PROBLEMAS NP

Problemas NP Completos

Subconjunto de problemas NP (tipo decisión), que son LOS más difíciles de resolver. Todo problemas NPC se transforma a SAT (U OTRO), EXISTE un algoritmo no determínistico polinomial

Problemas NP Hard

Conjunto de problemas (versión optimización de NP,..etc), que son más difíciles de resolver. Un problema que no está en NP se transforma a NPC , no se sabe de la existencia de un algoritmo DETERMINISTICO polinomial.