Tema 1: Algoritmia

ZAVALETA BARRALES NESTOR OSIEL

LOS PRINCIPALES
TEMAS DE HOY

Contenido

¿Qué es un algoritmo?
Características de un algoritmo
Análisis y complejidad

¿Qué es un algoritmo?

Es una secuencia de pasos que implica la descripción precisa de los pasos para llegar a la solución de un problema dado.

Características de un algoritmo

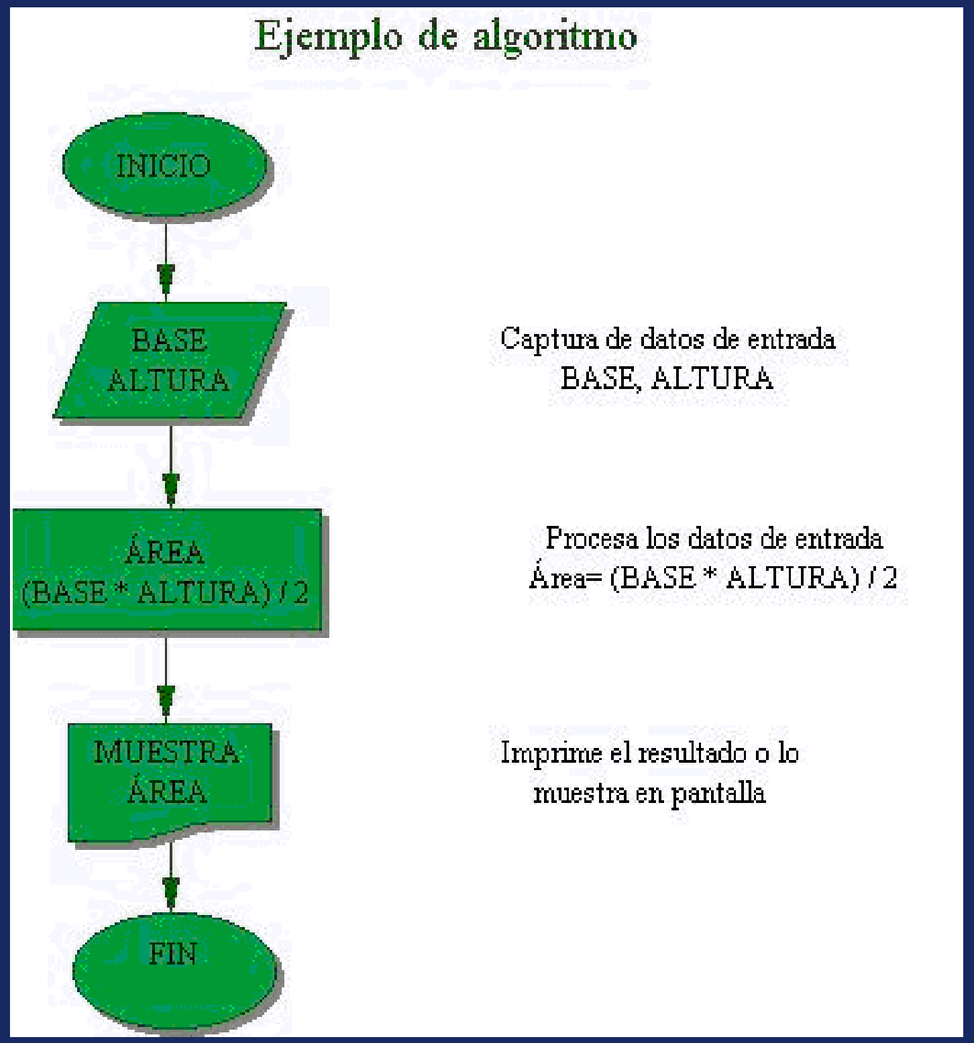
PRECISIÓN: UN ALGORITMO DESBE EXPRESARSE SIN AMBIGÜEDAD

DETERMINISMO: TODO EL ALGORITMO DEBE RESPONDER DEL MISMO MODO ANTE LAS MISMAS CONDICIONES

FINITO: LA DESCRICIÓN DE UN ALGORIMTO DEBE SER FINITO

ALGORITMIA

Representación de un algoritmo en un diagrama de flujo



Análisis y complejidad de algoritmos

Notación de Landau:

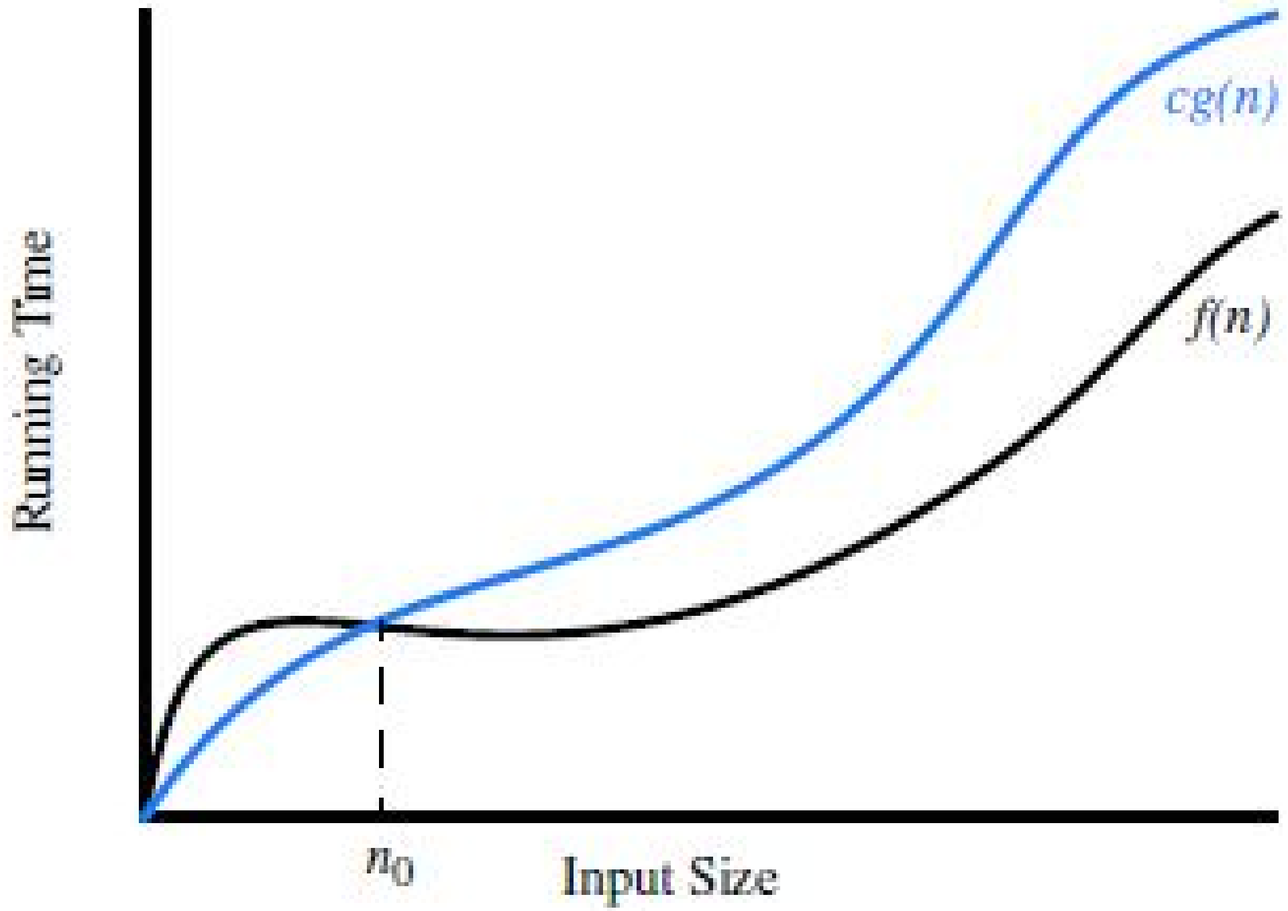
También llamada Notación Big O es una notación de comparación asintótica que permite definir cota superior e inferior de una función. Se usa en informática para definir el rendimiiento de un programa.

Ejemplos:

- $3n^2+5n-7 \in O(n^2)$.
- $O(2g(n)) = O(g(n))$
- $O(\log n) = O(\ln n)$
- $O(3n^2+5n-7) = O(n^2)$
- $O(n^2) \subset O(n^3)$
- $O(2^n)$

ALGORITMIA

Representación de Big O



Principales ordenes de complejidad

Orden	Nombre
O(1)	constante
O(log n)	logarítmica
O(n)	lineal
O(n log n)	casi lineal
O(n^2)	cuadrática
O(n^3)	cúbica
O(a^n)	exponencial

ALGORITMIA

Definiciones

Constante: Se dice que un algoritmo es constante cuando no depende de datos de entrada (n).

Logarítmico: El algoritmo reduce los datos de entrada en cada step. (Ej. Binary Search)

Lineal: El tiempo incrementa linealmente con los datos de entrada

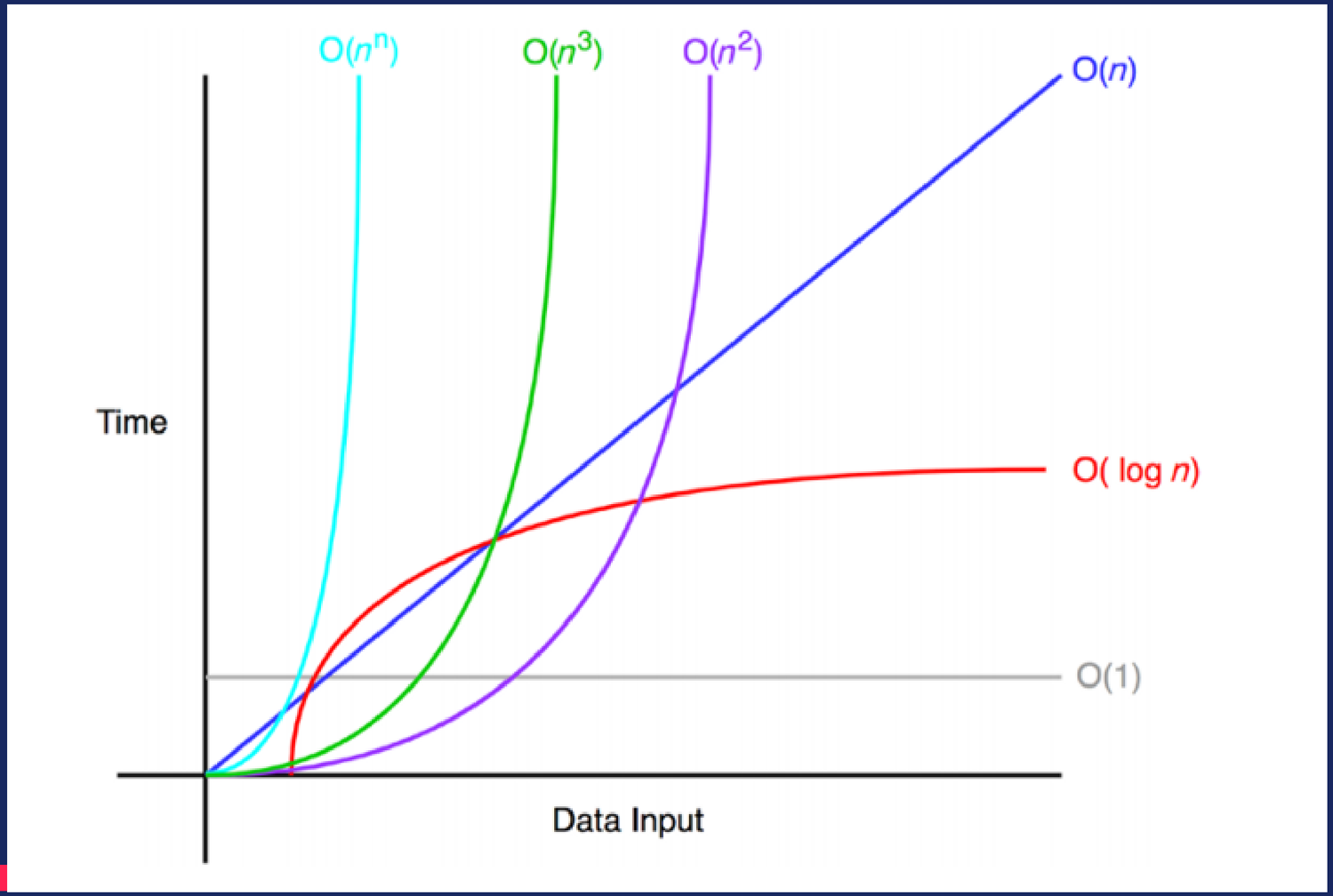
Quasilinear: Cuando cada operación en los datos de entrada tiene un tiempo de complejidad logaritmica. (Ej. Merge sort, Heap Sort).

Cuadrático: Cuando en cada operación que se realiza se desempeña una operación linear.

Definiciones

Exponencial: Se le llama algoritmo de crecimeinto exponencial cuandoel tiempo de complejidad aumenta al doble en cada adición de datos de entrada. (Ej, Brute-Force algorithms)

Ralación de los datos de entrada con el tiempo de ejecución



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

ALGORITMIA

Ejemplo de determinación de tiempo de ejecución y notación Big O

Analicemos la siguinete función:

$T(n) = 1 + n$

Donde n es el tamaño de datos entrantes

¿Cuál sería el témino dominante y por qué?

Ejemplo de determinación de tiempo de ejecución y notación Big O

Ahora analicemos el siguiente fragmento de código:

```
a=5
b=6
c=10
for i in range(n):
    for j in range(n):
        x = i * i
        y = j * j
        z = i * j
for k in range(n):
    w = a*k + 45
    v = b*b
d = 33
```

Efecto de las duplicaciones

A continuación se muestran las siguientes tablas que nos permiten observar de manera como es el efecto de el tamaño de datos en nuestros algoritmos, también el efecto del tiempo en cuanto a las operaciones que se realizan.

Better

Worst

T(n)	n = 100	n = 200
log(n)	1 h.	1.15 h.
n	1 h.	2 h.
nlog(n)	1 h.	2.30 h.
n^2	1 h.	4 h.
n^3	1 h.	8 h.
2^n	1 h.	$1.27*10^{30}$ h.

T(n)	t = 1h	t = 2h
log(n)	n = 100	n = 10000
n	n = 100	n = 200
nlog(n)	n = 100	n = 178
n^2	n = 100	n = 141
n^3	n = 100	n = 126
2^n	n = 100	n = 101

ALGORITMIA

Ejercicios de términos no dominantes

¿Cuál es el término dominante de las siguientes expresiones?

O(N +N^2) =

O(N+LOG(N)) =

O((5*2^N)+ N^100) =

VAMONOS A COLAB