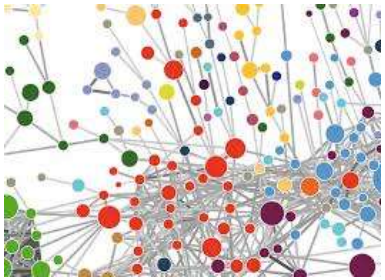




2.2 Complejidad computacional

2.2.1 Introducción a la complejidad

- El propósito de la complejidad computacional es analizar el comportamiento de los algoritmos.
- El objeto de estudio de esta rama:
  - Medir/estimar la cantidad de recursos necesarios para resolver problemas (computacionales).
  - Clasificar algoritmos de acuerdo a su dificultad computacional.

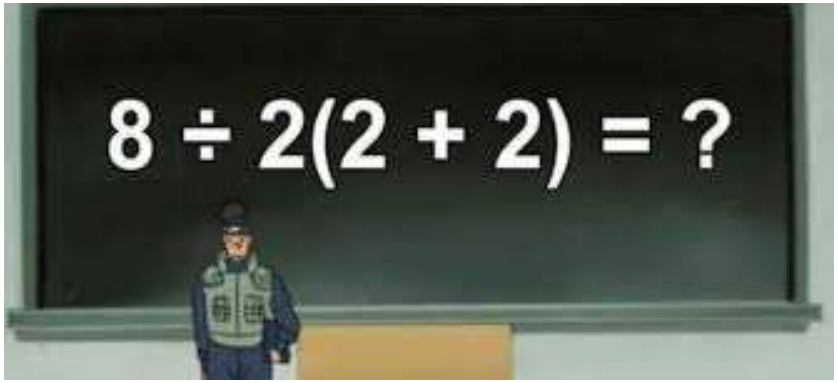
2.2.1 Introducción a la complejidad

Cuando se tiene más de un algoritmo para resolver un problema, mediante el análisis de su complejidad permite saber…

- ¿Cuál se debe elegir?
- ¿Cuál conviene más?
- ¿Cuál es mejor?

2.2.1 Introducción a la complejidad

- Su principal objetivo es dividir problemas entre aquellos tratables y no tratables de forma práctica.
- La teoría de la complejidad ha mostrado que existen problemas computacionales para los cuales, la energía del universo no sería suficiente para resolverlos.



2.2.2 Aplicaciones y medidas de complejidad

- **¿Qué se analiza?**

El análisis de complejidad se centra en el tiempo y en la cantidad de memoria que consume un algoritmo al ejecutarse

2.2.2 Aplicaciones y medidas de complejidad

- El factor de tiempo por lo general es más importante que la memoria; las consideraciones de eficiencia normalmente se basan en la cantidad de tiempo transcurrido cuando se resuelven problemas.
- Para comparar 2 o más algoritmos se tendrían que probar en las mismas condiciones (maquina, lenguaje, memoria disponible)

2.2.2 Aplicaciones y medidas de complejidad

- Dentro de un algoritmo se precisa elegir qué instrucciones se contabilizarán para la medida de la complejidad.
- Se busca elegir las operaciones consideradas como significativas.
  - Operaciones de comparación
  - Operaciones aritméticas

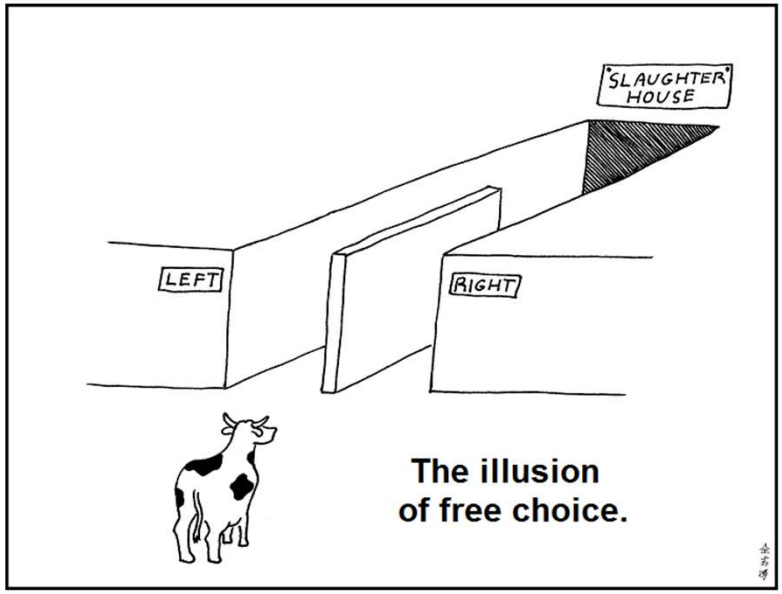
2.2.2 Aplicaciones y medidas de complejidad

- Las operaciones de comparación se consideran equivalentes entre sí
- Otra operación común es la de intercambiar valores de posiciones distintas en una colección
- Ambas aparecen principalmente en algoritmos de búsqueda y ordenamiento.

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor que o igual a
<	Menor que
<=	Menor que o igual a
<> ó !=	Diferente de

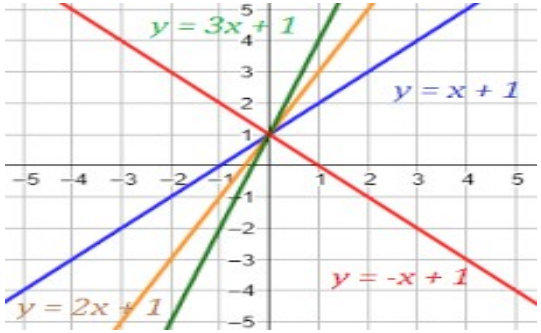
2.2.2 Aplicaciones y medidas de complejidad

- Los operadores aritméticos se contabilizan en dos grupos:
  - Aditivos: (suma, resta, incremento y decremento)
  - Multiplicativos (multiplicación, división y módulo)



2.2.2 Aplicaciones y medidas de complejidad

- Una vez que se ha seleccionado qué se medirá para determinar la complejidad, se debe obtener una función del tamaño de la entrada que caracteriza el comportamiento del algoritmo
- Al igual que en otras disciplinas científicas, el conocimiento previo sirve para predecir el comportamiento de diversas situaciones y procesos de interés.

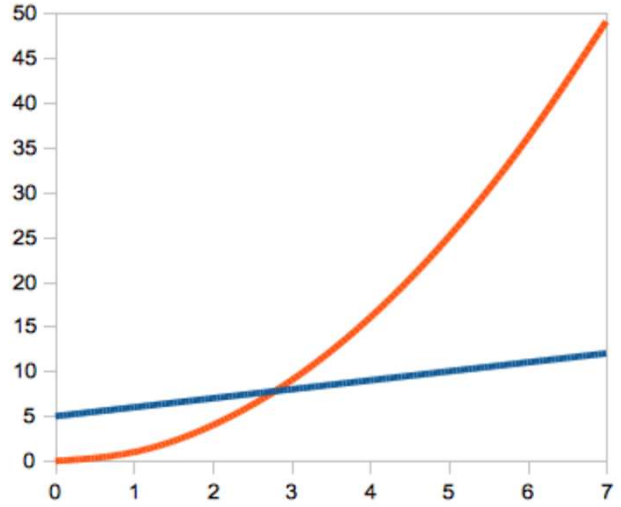


2.2.2 Aplicaciones y medidas de complejidad

- Si se logra determinar la complejidad temporal de un algoritmo entonces se puede predecir de forma confiable el tiempo de ejecución para diferentes instancias sin tener que ejecutarlo

2.2.2 Aplicaciones y medidas de complejidad

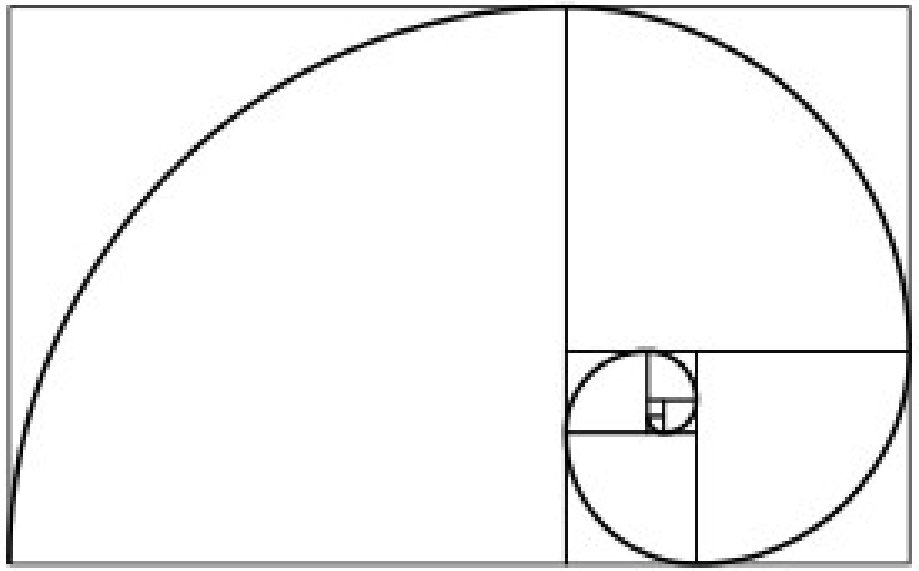
- Se puede comparar la eficiencia de dos o más algoritmos que resuelven el mismo problema, analizando sus funciones de complejidad.



EJEMPLO

- Elabora el pseudocódigo de la sucesión de Fibonacci

0 1 1 2 3 5 8 13 21
34 55 89 144 233 377
610 987 1597 2584....



EJEMPLO

- FIBONACCI 1

```
Programa: FibIterativo
    Algoritmo:
    1 leer n
    2 i = 0; j = 1
    3 para k = 1 hasta n hacer
    4   t=i+j
    5   i=j
    6   j=t
    7 devolver j
```

EJEMPLO

- FIBONACCI 2

```
Programa: FibRec
  Algoritmo:
  1 leer n
  2 si n< 2
  3   devolver n
  4 si no
  5   devolver FibRec(n-1)+Fibrec(n-2)
```

EJEMPLO – Tiempo de ejecución

n (entrada)	**10**	**20**	**30**	**50**	**100**
FibRec	1 seg	1seg	2min	21 dias	10^9 años
FibIterativo	1/6 mseg	1/3 mseg	½ mseg	¾ mseg	1.5mseg

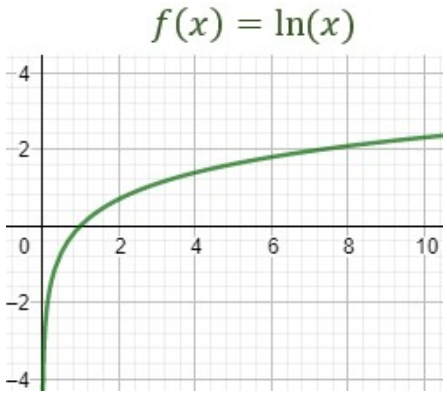
2.2.3 Funciones de complejidad

- Una vez que se ha podido determinar la complejidad de un algoritmo mediante una función, a grandes rasgos se identifican 3 tipos de comportamiento.

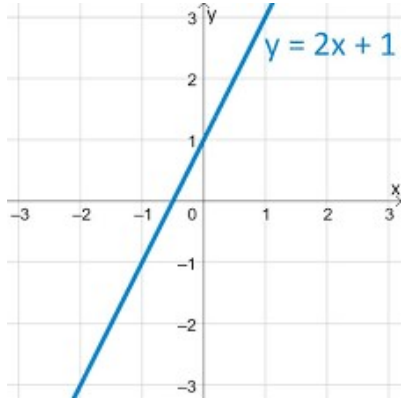
  - ✓Lineal y logarítmico
  - ✓Polinomial
  - ✓Exponenciales

2.2.3.1 Funciones lineales y logarítmicas

- Lo algoritmos cuya función característica es logarítmica permiten atacar problemas muy grandes, ya que una entrada del doble de tamaño solo requiere un poco mas de tiempo.



- Los algoritmos de complejidad lineal muestran un comportamiento “predecible”: si se duplica el tamaño de la entrada, se requerirá el doble de tiempo para resolverlos.

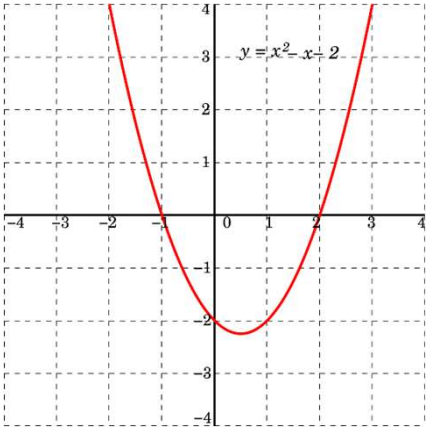


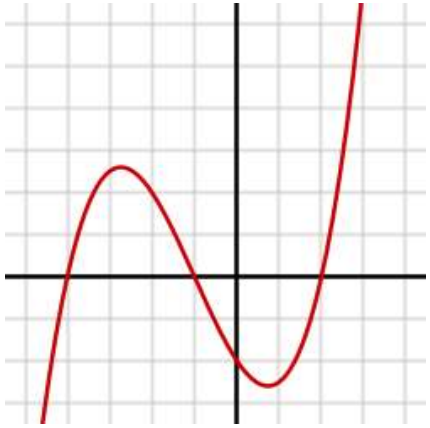
2.2.3.2 Funciones polinomiales

- Los algoritmos con este tipo de comportamiento representan un punto intermedio.
- La mayor parte de los casos se podrán resolver satisfactoriamente para entradas suficientemente grandes

2.2.3.2 Funciones polinomiales

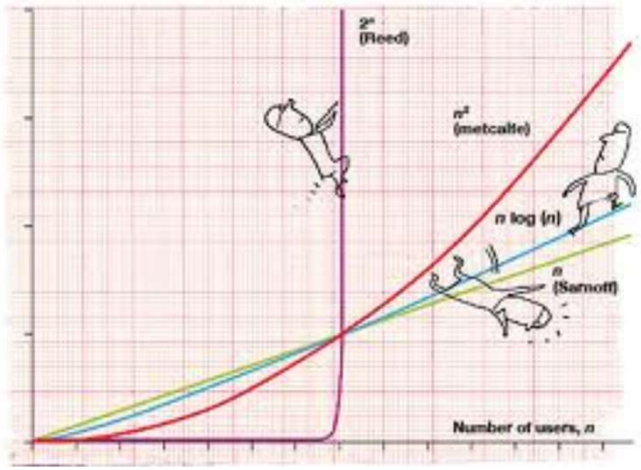
- Algunos autores separan la complejidad cuadrática y la complejidad cúbica como 2 clases de complejidad distintas.
- Esto es porque existen algoritmos que en el caso de complejidad cuadrática pueden resolver problemas con entradas aceptablemente grandes.





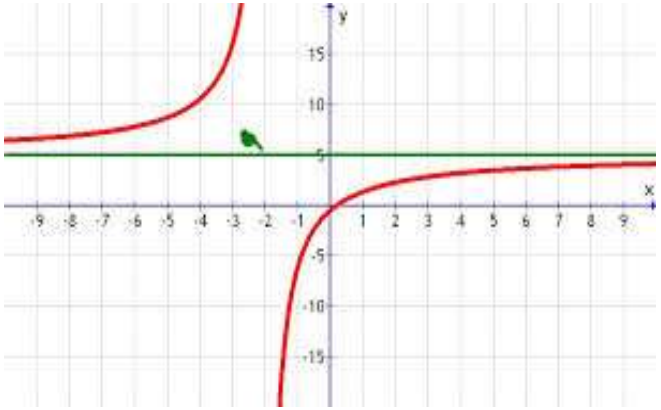
2.2.3.3 Funciones exponenciales

- Los algoritmos de comportamiento exponencial o factorial, solamente funcionarían para entradas muy pequeñas.
- En esos casos es mejor buscar otro algoritmo para solucionar el problema asociado.



2.2.3.4 Funciones asintóticas

- Los algoritmos de comportamiento asintótico son aquellos que se encuentran en los “extremos” de la clasificación de complejidad.
- Se les denomina de esta forma porque la función que los representa es de tipo asintótica.



2.2.4 Notación para medir la complejidad

- Sea la función:

$$f(n) = n^2 + 100n + \log_{10} n + 1000$$

n	$f(n)$	n^2		$100n$		$\log_{10} n$		1,000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1,000	90.83
10	2,101	100	4.76	1,000	47.6	1	0.05	1,000	47.60
100	21,002	10,000	47.6	10,000	47.6	2	0.001	1,000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1,000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1,000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1,000	0.00

2.2.4 Notación para medir la complejidad

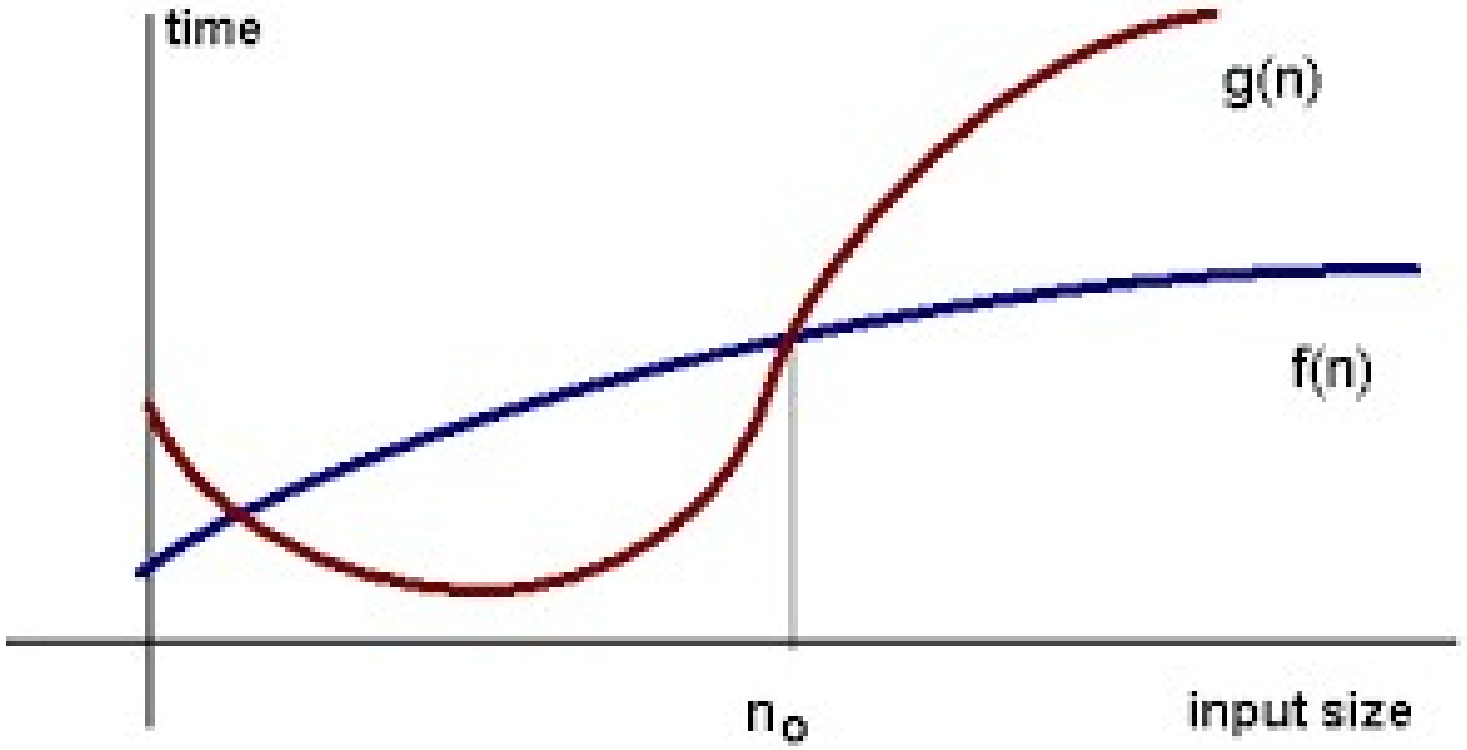
- La razón de crecimiento de un algoritmo (de una función matemática) es dominada por el término más grande de la función, eso implica que los otros términos pueden ser despreciados.

2.2.4.1 Big-O

- La notación de uso más común para especificar la complejidad asintótica es la notación “O Grande (Big-O)” introducida en 1894 por Paul Bachman.

Definición: $f(n)\ es\ O\big(g(n)\big)$ si existen los números positivos c y N tales que $f(n) \leq c \cdot g(n)$ para todo $n \geq N$

2.2.4.1 Big-O



2.2.4.1 Big-O

- De manera creciente, la clasificación de la categoría de complejidad de notación O-grande es la siguiente

O(1)	constante
O(log n)	Logarítmico
O(n)	Lineal
O(n log n)	Lineal-logarítmico
O(n^c)	Polinomial
O(c^n)	Exponencial
O(n!)	factorial

*Considerar que ***c*** es una constante y ***n*** el tamaño de la entrada.

2.2.4.1 Big-O

- Ejemplos de complejidades conocidos

O(1)	Tablas hash
O(log(n))	Búsqueda binaria
O(n)	Búsqueda lineal
O(n log(n))	Heap sort, Quick sort
$O(n^2)$	Ordenamiento por selección
$O(n^3)$	Multiplicación de matrices
$O(2^n)$	Torres de Hanoi

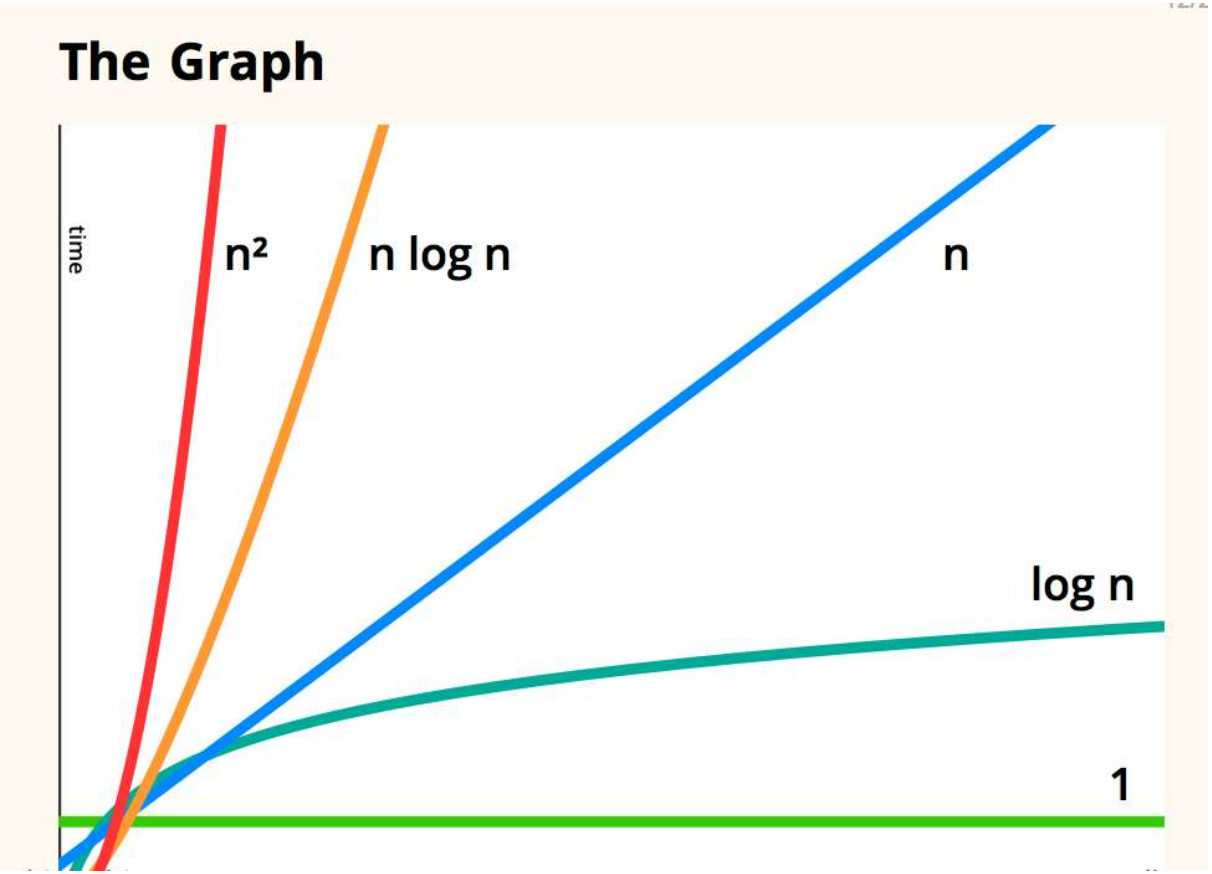
2.2.4.2 Tiempos de ejecución por categoría

n		10		10^2		10^3	
constant	$O(1)$	1	1 µsec	1	1 µsec	1	1 µsec
logarithimic	$O(\lg n)$	3.32	3 µsec	6.64	7 µsec	9.97	10 µsec
linear	$O(n)$	10	10 µsec	10^2	100 µsec	10^3	1 msec
$O(n \lg n)$	$O(n \lg n)$	33.2	33 µsec	664	664 µsec	9970	10 msec
quadratic	$O(n^2)$	10^2	100 µsec	10^4	10 msec	10^6	1 sec
cubic	$O(n^3)$	10^3	1 msec	10^6	1 sec	10^9	16.7 min
exponential	$O(2^n)$	1024	10 msec	10^{30}	$3.17 * 10^{17}$ yrs	10^{301}	

2.2.4.2 Tiempos de ejecución por categoría

n		10^4		10^5		10^6	
constant	$O(1)$	1	1 µsec	1	1 µsec	1	1 µsec
logarithmic	$O(\lg n)$	13.3	13 µsec	16.6	7 µsec	19.93	20 µsec
linear	$O(n)$	10^4	10 msec	10^5	0.1 sec	10^6	1 sec
$O(n \lg n)$	$O(n \lg n)$	133×10^3	133 msec	166×10^4	1.6 sec	199.3×10^5	20 sec
quadratic	$O(n^2)$	10^8	1.7 min	10^{10}	16.7 min	10^{12}	11.6 days
cubic	$O(n^3)$	10^{12}	11.6 days	10^{15}	31.7 yr	10^{18}	31,709 yr
exponential	$O(2^n)$	10^{3010}		10^{30103}		10^{301030}	

2.2.4.2 Tiempos de ejecución por categoría Cola



Operaciones Cola

- La complejidad temporal asintótica de las operaciones es:

Operación	Orden
Create	O(1)
clear()	O(n)
isEmpty()	O(1)
enqueue()	O(1)
dequeue()	O(1)
first(n)	O(1)

Listas Ligadas simples

- Complejidad Asintótica

Operación	Orden
Insertar	O(n)
Eliminar(n)	O(n)
Buscar(n)	O(n)