

Complejidad Algorítmica

Computer Science

CS1100 - Introducción a Ciencia de la Computación

¿Qué tan eficiente es un algoritmo? - Tiempo

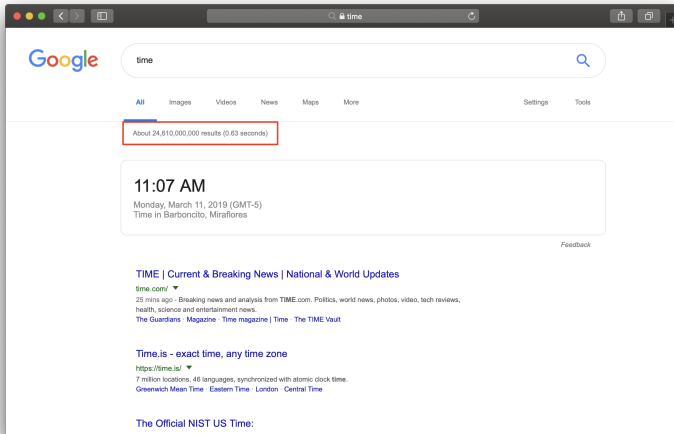


Figure: Google demora 0.63 segundos en obtener 24,610,000,000 resultados

¿Qué tan eficiente es un algoritmo? - Espacio

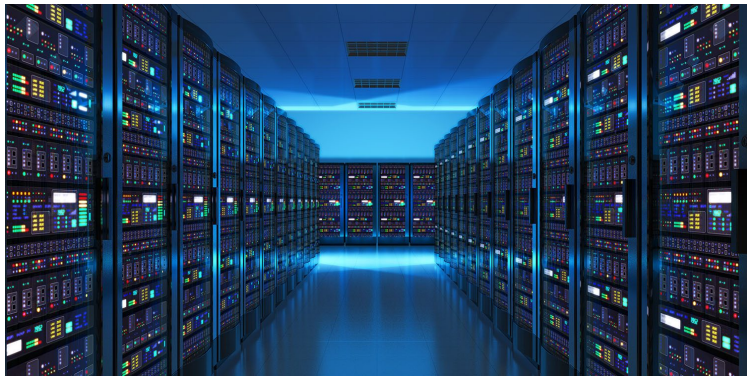


Figure: Google almacena $\sim 10\text{-}15$ exabytes de información. Si estimamos que 1 computadora tiene 500GB, 1 exabyte equivaldría a 2 millones de computadoras

Si tu algoritmo demora 1 segundo para procesar una entrada de 1000 elementos, ¿cómo se comportará si duplicamos el número de elementos de entrada?

- Se demorará la misma cantidad de tiempo.
- Será el doble de rápido.
- Tomará 4 veces más tiempo.

¿ Por qué esto es importante ?

¿ De qué depende el análisis de un algoritmo ? ¿ Por qué ?

¿Cómo se mide el tiempo en Python?

```
1  import time # Librería para acceder a las variables de tiempo
2
3  start = time.time()
4
5  """
6  Aquí van las líneas de código que van a ser medidas para
7  saber cuánto demoran en ejecutarse
8
9  """
10
11  end = time.time()
12  elapsed_time = end - start
13
14  print("It took %f seconds" % (elapsed_time))%
```

¿Cómo se mide el tiempo en Python?

```
1
2 import time # Librería para acceder a las variables de tiempo
3
4 start = time.time()
5
6 n = 10
7 total = 0
8 for counter in range(1,n+1):
9     total = total + counter
10
11 end = time.time()
12 elapsed_time = end - start
13
14 print("Sum of 1 until %d is %d and took %f seconds" % (n, total,
    elapsed_time))
```

Arreglos en 1 dimensión

10	7	3	8	5	9
0	1	2	3	4	5

```
1
2 # Crear un arreglo como
   lista
3 A = [10, 7, 3, 8, 5, 9]
4
5 # Crear un arreglo con
   NumPy
6 A = array([10, 7, 3, 8,
   5, 9])
7
8 # Iterar usando el índice
9 for i in range(len(A)):
10     print("Valor {}: {}".
11           format(i + 1, A[i]
12                 ))
11 % \end{minted}
```

Arreglos en 1 dimensión - Ejercicio

Dado un arreglo de n números enteros consecutivos. ¿ Cuánto tiempo demora el algoritmo para obtener la sumatoria de los n números? Para:

1	2	3	4	5	6	...
0	1	2	3	4	5	...

· $n = 10^2$

· $n = 10^4$

· $n = 10^6$

· $n = 10^8$

Para verificar recuerda que la sumatoria de los n primeros números naturales está dada por $\frac{n(n+1)}{2}$

Arreglos en 2 dimensiones

	0	1	2	3	4
0	255	7	0	0	69
1	100	0	56	100	109
2	101	254	23	11	0
3	2	7	255	107	96
4	178	250	0	102	45

```
1
2 # Crear una matriz usando
   listas
3 A = [[1, 4, 5],
4       [-5, 8, 9]]
5
6 # Crear una matriz usando
   arreglos NumPy
7 A = np.array([[1, 2, 3],
8               [3, 4, 5]])
9
10 # Iterar una matrix
11 for j in range(columns):
12     for i in range(rows):
13         print A[i][j]
14         %\end{minted}
```

Figure: Matriz

Arreglos en 2 dimensiones

	0	1	2	3	4
0	255	7	0	0	69
1	100	0	56	100	109
2	101	254	23	11	0
3	2	7	255	107	96
4	178	250	0	102	45

Figure: Matriz

- Resolver sistema de ecuaciones
- Representar imágenes
- Renderizar en pantallas
- Inteligencia Artificial

Arreglos en 2 dimensiones



Figure: Imagen en escala de grises

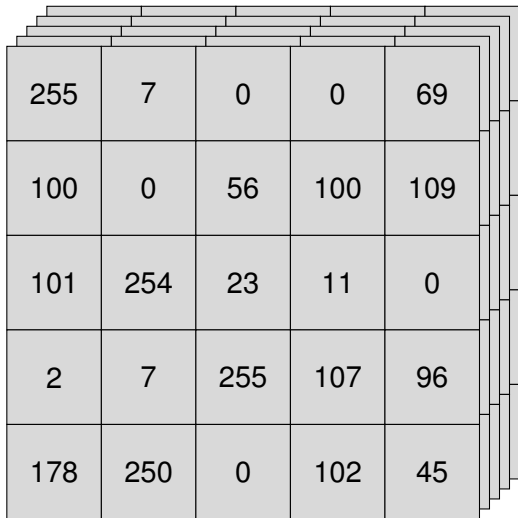
- Resolver sistema de ecuaciones
- Representar imágenes
- Renderizar en pantallas
- Inteligencia Artificial

Arreglos en 2 dimensiones - Ejercicio

Dada una matriz cuadrada M de tamaño $n \times n$, escriba el código de algunas funciones para medir el tiempo de ejecución de los siguientes problemas:

- Encontrar el elemento de **menor** valor
- Encontrar el elemento de **mayor** valor
- Encontrar la **sumatoria** de todos los valores
- Construir otra matriz con los elementos elevados al **cuadrado**.

Arreglos en 3 dimensiones



255	7	0	0	69
100	0	56	100	109
101	254	23	11	0
2	7	255	107	96
178	250	0	102	45

- Realidad virtual
- FaceID del iPhone
- Simulaciones quirúrgicas
- Gráficos de Fornite
- Física cuántica

Arreglos en 3 dimensiones

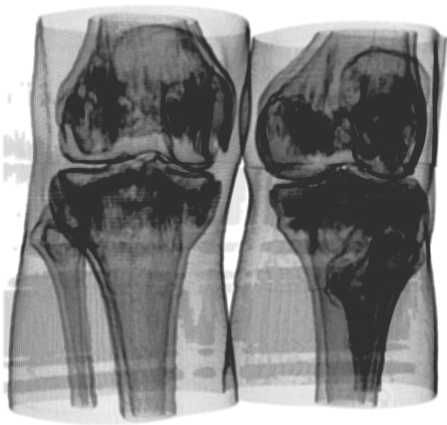


Figure: Imagen tomográfica

- Realidad virtual
- FaceID del iPhone
- Simulaciones quirúrgicas
- Gráficos de Fornite
- Física cuántica

Análisis Asintótico - Bucles anidados

```
1 sum = 0
2 for i = 1 to n do
3   for j = 1 to n do
4     sum = sum + 1
```

$$\sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n \quad (1)$$
$$= n^2$$

Análisis Asintótico - Bucles anidados

```
1 sum = 0
2 for i = 1 to n do
3     for j = i to n do
4         sum = sum + 1
```

$$\begin{aligned}\sum_{i=1}^n \sum_{j=i}^n 1 &= \sum_{i=1}^n (n - i + 1) \\ &= \sum_{i=1}^n (n + 1) - \sum_{i=1}^n i \\ &= n(n + 1) - \frac{n(n + 1)}{2} \\ &= \frac{n(n + 1)}{2} \\ &= n^2\end{aligned}$$

(2)

Análisis Asintótico

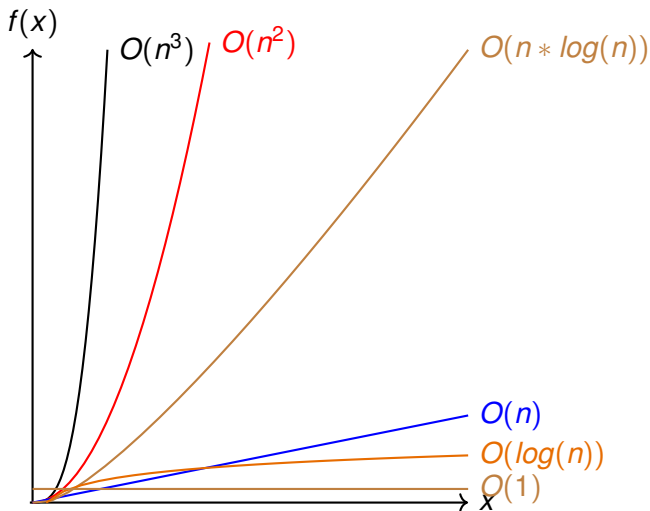


Figure: Análisis asintótico de algoritmos

Ejercicios y lectura adicional

Ejercicios



[www.hackerrank.com
sem09-sesion-b-complejidad-01](https://www.hackerrank.com/sem09-sesion-b-complejidad-01)

Conteo de Vocales



[www.hackerrank.com
sem09-sesion-b-complejidad-02](https://www.hackerrank.com/sem09-sesion-b-complejidad-02)

Rotación de matrices



[www.hackerrank.com
sem09-sesion-b-complejidad-03](https://www.hackerrank.com/sem09-sesion-b-complejidad-03)

Rotación de matrices con re



[www.hackerrank.com
sem09-sesion-b-complejidad-04](https://www.hackerrank.com/sem09-sesion-b-complejidad-04)

Suma de números

Lectura Adicional



Algorithm Analysis
@ahmedamedy/algorithm-analysis-bf0ca650619



Resumen

El análisis de un algoritmo depende de la **cantidad de elementos** de entrada.

Empíricamente se puede medir el tiempo que demora un algoritmo para ser comparado.

Existe una manera teórica que permite comparar algoritmos denominado **análisis asintótico**.