<!DOCTYPE html>

Jorge Calvo

Complejidad Algorítmica

Big O

Sígueme en - LinkedIn Twitter Blog

Complejidad Algorítmica

Definición

La **complejidad algorítmica** se refiere a la medida de la cantidad de recursos computacionales necesarios para ejecutar un algoritmo. Estos recursos pueden ser tiempo de ejecución o espacio de memoria requerido. La complejidad algorítmica se utiliza para evaluar el rendimiento de los algoritmos y determinar su eficiencia en términos de consumo de recursos.

La complejidad algorítmica se expresa generalmente en función del tamaño de entrada del algoritmo, denotado como **"n"**. La notación **Big O** se utiliza para describir la complejidad algorítmica de manera asintótica, lo que significa que se enfoca en el comportamiento del algoritmo a medida que el tamaño de entrada tiende hacia infinito.

Ejemplo

Un algoritmo con complejidad **O(n)** indica que su tiempo de ejecución aumenta linealmente con el tamaño de entrada. Por otro lado, un algoritmo con complejidad $O(n^2)$ indica que su tiempo de ejecución aumenta cuadráticamente con el tamaño de entrada. La complejidad algorítmica proporciona una herramienta importante para el análisis y diseño de algoritmos eficientes, permitiendo a los desarrolladores tomar decisiones informadas sobre qué enfoque utilizar para resolver un problema.

Ejemplos tipos de complejidad

O(1): Constante

- Un programa que me diga si un número es par o impar

In [1]:
```
num=45
if num%2==0:
  print("Par")
else: print("Impar")
```

```
Impar
```

O(n): Lineal

- Un programa que me diga el número más alto de una lista

In [2]:
```
import random

#Creamos una lista de 25 elementos con números aleatorios entre 0 y 100
lista=[random.randint(0,100) for x in range(0,25)]

print(lista)
max=lista[0]

for x in range (0,len(lista)):
  if lista[x]>max:
    max=lista[x]
print("El número más alto es ",max)
```

```
[15, 65, 8, 57, 16, 55, 81, 52, 56, 91, 17, 26, 67, 30, 15, 7, 51, 95, 96, 12, 42, 8
7, 89, 88, 51]
El número más alto es  96
```

$O(n^2)$: Cuadrático

- Un programa que ordena una lista dada

In [3]:
```
import random

lista=[random.randint(0,1000) for x in range(0,20)]
lista2=[]
print(lista)

while len(lista)>0:
    max=lista[0]
    for x in range (0,len(lista)):
      if lista[x]>max:
        max=lista[x]
    lista2.append(max)
    lista.remove(max)
print(lista2)
```

```
[481, 329, 745, 683, 615, 318, 431, 748, 897, 945, 943, 369, 101, 61, 880, 155, 167,
677, 720, 914]
[945, 943, 914, 897, 880, 748, 745, 720, 683, 677, 615, 481, 431, 369, 329, 318, 167,
155, 101, 61]
```

Mejorar algortimo: $O(n)$ Vs $O(long(n))$

Indicar la posición de un elemento en una lista ordenada

El objetivo de este ejercicio es mejorar el tiempo de computación, es decir la complejidad **(O(n))** de un algortimo para la búsqueda de elementos en una lista ordenada.

Busqueda Binaria

La complejidad de este algoritmo de búsqueda binaria es O(log n) debido a que en cada iteración, el tamaño del rango de búsqueda se reduce a la mitad. Sin embargo, si se realiza la búsqueda en una lista que ya está ordenada, la complejidad final sería O(n log n) porque se deben realizar n comparaciones (número máximo de iteraciones) en total para encontrar el elemento objetivo en la lista ordenada.

In [4]:
```
import random
import time

#Utilizamos random.sample para generar una lista de números únicos dentro de un rango
my_list=random.sample(range(0,1000000),900000)
#Utilizamos la función sort para ordenar la lista
my_list.sort()

#print("Mi lista ordenada es: ")
```

```
#print(my_list)
search = int(input("Busqueda: "))


#Algortimo tradicional

iniciar=time.time()

for x in range (0,len(my_list)):
  if my_list[x]==search:
    print("Su posición es ", x)
    break

final=time.time()

print("El tiempo del algortimo tradicional ", final-iniciar)


#Algoritmo Binario
primero=0
ultimo=len(my_list)
mitad=0

iniciar=time.time()
while primero < ultimo:
  mitad = (primero + ultimo) // 2
  if my_list[mitad] > search:
    ultimo = mitad -1
  elif my_list[mitad] < search:
    primero = mitad + 1
  else:
    print("La posición es ", mitad)
    break
final=time.time()

print("El tiempo del algortimo binario ", final-iniciar)
```

```
Su posición es  41
El tiempo del algortimo tradicional  0.0005044937133789062
El tiempo del algortimo binario  0.00042510032653808594
```