Instituto Politecnico Nacional

ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”

ANÁLISIS DE ALGORITMOS

EJERCICIO 3: GRAFICACIÓN DE ORDENES DE COMPLEJIDAD

PROFE: Edgardo Adrián Franco Martínez

ALUMMNO: Rojas Alvarado Luis Enrique

GRUPO: 3CM4

INTRODUCCIÓN

En el siguiente trabajo se graficaron los ordenes de complejidad en el lenguaje de programación python usando la librería de matplotlib. En donde es más fácil visualizar los rangos solicidatos (0<n<100,000).

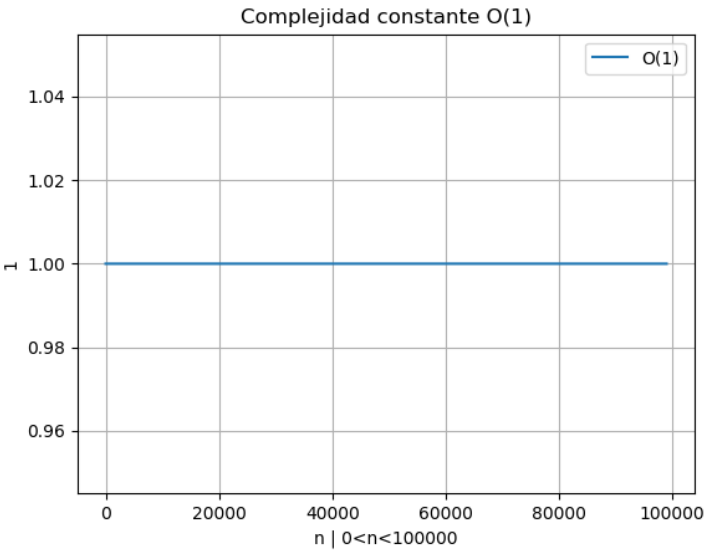
PARTE I: GRAFICACIÓN POR SEPARADO

Para éste insciso se grafican las diferentes complejidades de forma individual analizando la gráfica para una funcion con entradas n demasiado grandes para conocer el comportamiento de cada una.

• $\boldsymbol{O}$ (**1**) Complejidad constante

Quiere decir que el orden de complejidad más grande de todo el algoritmo analizado es de orden constante, quiere decir que solo tiene instrucciones de asignación, comparación, operación aritmética o la utilización de un arreglo unidimensional.

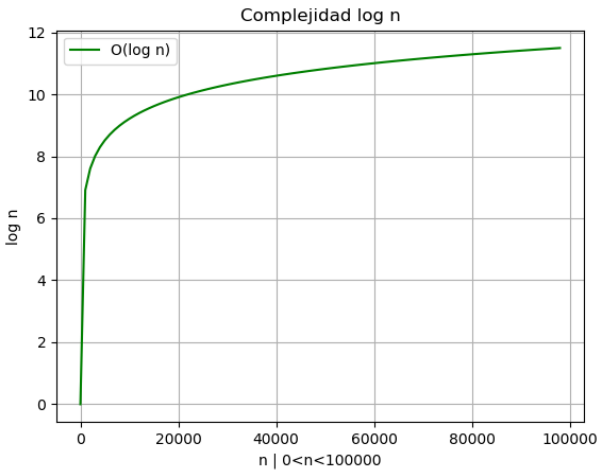
```python
#Se declara las librerías para graficar
from matplotlib import pyplot
#funcion que retorna un valor constante
def f1(x):
    return 1
#se establece un rango cuando 0<n<100,000
x=range(1,99000)
#se manda a graficar en el rango establecido
pyplot.plot(x,[f1(i) for i in x], label="O(1)")
#se le da un titulo a la gráfica
pyplot.title("Complejidad constante O(1)")
#una etiqueta para el eje de las x
pyplot.xlabel("0<n<100,000")
#Se añade una leyenda en la parte superior derecha
pyplot.legend(loc="upper right")
#se añade una cuadrícula
pyplot.grid()
#se guarda la grafica como imagen .png
pyplot.savefig("Complejidad-constante.png")
#se muestra la gráfica
pyplot.show()
```



• $\boldsymbol{O}$ (log $\boldsymbol{n}$) Complejidad logarítmica

Quiere decir que el orden de complejidad más grande es de log n, suponiendo que en el algoritmo hay una variable que se incrementa multiplicándose o se reduce dividiéndose por un valor constante.

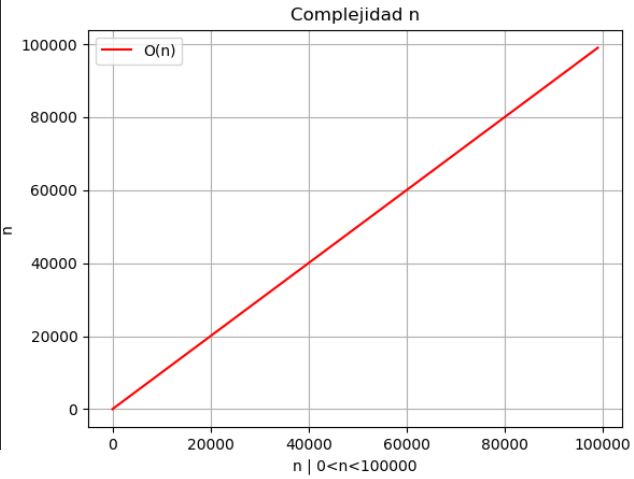
```python
#Se importa la librería para poder extraer funciones
#Entre las que se encuentran sen(),cos() y log()
import numpy
#Se importa la librería de graficación para python
from matplotlib import pyplot
#se establece un dominio de la funcion donde 0<x<100,000
#con incrementos de 1,000 en 1,000
x=numpy.arange(1,99000,1000)
#Se evalua la funcion de la lista de puntos x en la funcion y
y=numpy.log(x)
#Se grafica la funcion con un color de linea verde y una etiqueta
pyplot.plot(x,y,'g',label="O(log n)")
#Título de la gráfica
pyplot.title("Complejidad log n")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-logn.png")
#Se muestra la grafica
pyplot.show()
```



• $\boldsymbol{O}$ ($\boldsymbol{n}$) Complejidad lineal

Quiere decir que si existe una variable que se incrementa o decrementa en un valor constante el orden que predomina en el algoritmo es el lineal.

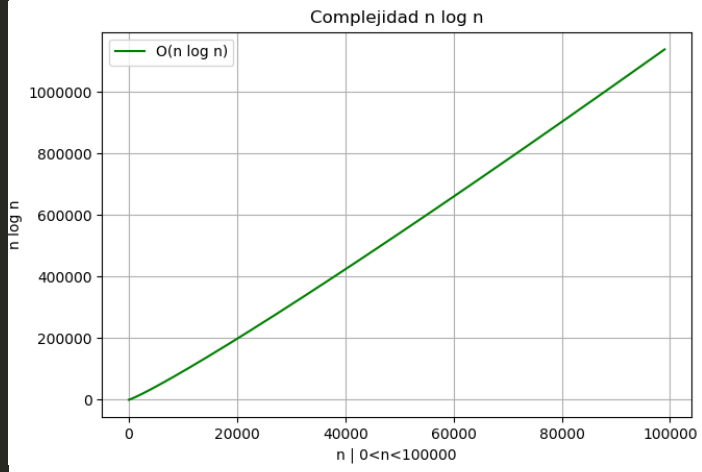
```python
#Se importa las librerías para graficar
from matplotlib import pyplot
#se establece un dominio de la funcion donde 0<x<100,000
x=range(1,99000)
#Se evalua la funcion de la lista de puntos x en la funcion y
y=x
#Se grafica la funcion con un color de linea verde y una etiqueta
pyplot.plot(x,y,'r',label="O(n)")
#Título de la gráfica
pyplot.title("Complejidad n")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-lineal-n.png")
#Se muestra la grafica
pyplot.show()
```



• $\boldsymbol{O}$ ($\boldsymbol{n}$ log $\boldsymbol{n}$) Complejidad “n log n”

Quiere decir que en éste orden de complejidad hay un ciclo en el cual entra n veces y que dentro de éste hay una variable de control que se multiplica o se divide por un valor constante.

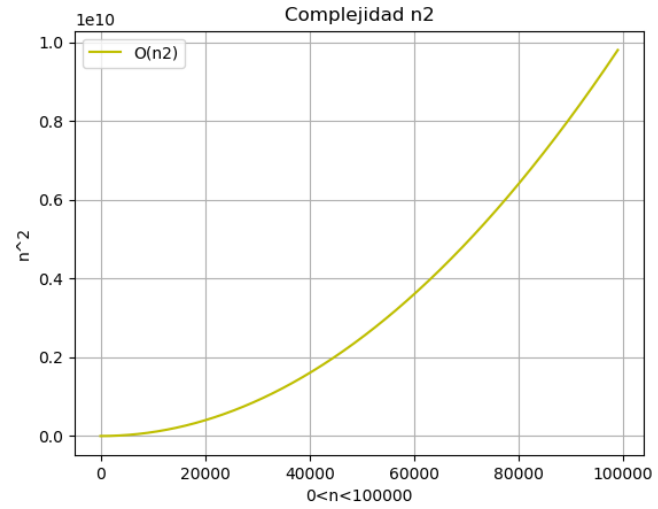
```python
#Se importa la librería para poder extraer funciones
#Entre las que se encuentran sen(),cos() y log()
import numpy
#Se importa la librería de graficación para python
from matplotlib import pyplot
#se establece un dominio de la funcion donde 0<x<100,000
#con incrementos de 1,000 en 1,000
x=range(1,99000)
#Se evalua la funcion de la lista de puntos x en la funcion y
r=numpy.log(x)
z=r*x
#Se grafica la funcion con un color de linea verde y una etiqueta
pyplot.plot(x,z,'g',label="O(n log n)")
#Título de la gráfica
pyplot.title("Complejidad n log n")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-nlogn.png")
#Se muestra la grafica
pyplot.show()
```



• $\boldsymbol{O}$ ($\boldsymbol{n}^2$) Complejidad cuadrática

Quiere decir que pueden existir dos ciclos anidados que entran n veces uno dentro del otro y que por tanto es el orden más alto de complejidad dentro del algoritmo.

```python
#Se importa la librería de graficación para python
from matplotlib import pyplot
#Funcion cuadratica recive el parametro x evaluado de 0<x<100,000
def f(x):
    y=x**2
    return y
#se establece un dominio de la funcion donde 0<x<100,000
x=range(1,99000)
#Se grafica la funcion con un color de linea amarillo y una etiqueta
#el ciclo funciona para cada valor de x en el rango establecido
pyplot.plot(x,[f(i) for i in x],'y',label="O(n2)")
#Título de la gráfica
pyplot.title("Complejidad n2")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-n2.png")
#Se muestra la grafica
pyplot.show()
```

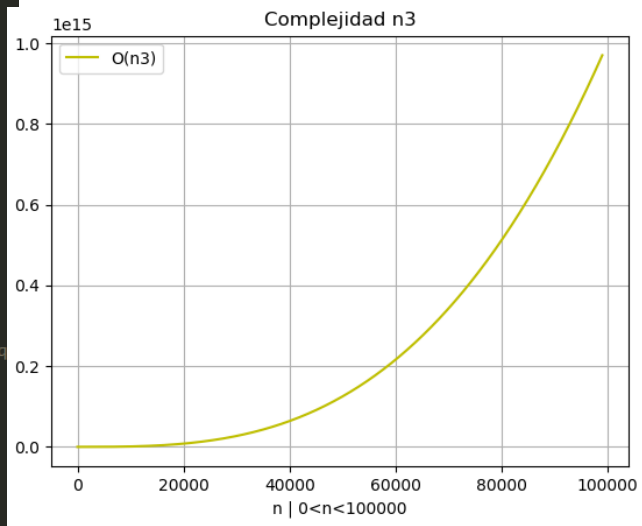


El número 1e10 significa que el eje de las “y” está en notación científica o exponencial y se refiere a que el $(99000)^2 \approx 0.9 \times 10^{10}$

• $\boldsymbol{O}$ ($\boldsymbol{n}^3$) Complejidad cubica

Quiere decir que pueden existir tres ciclos anidados que entran n veces uno dentro de otro y que por tanto es el orden más alto de complejidad dentro del algoritmo.

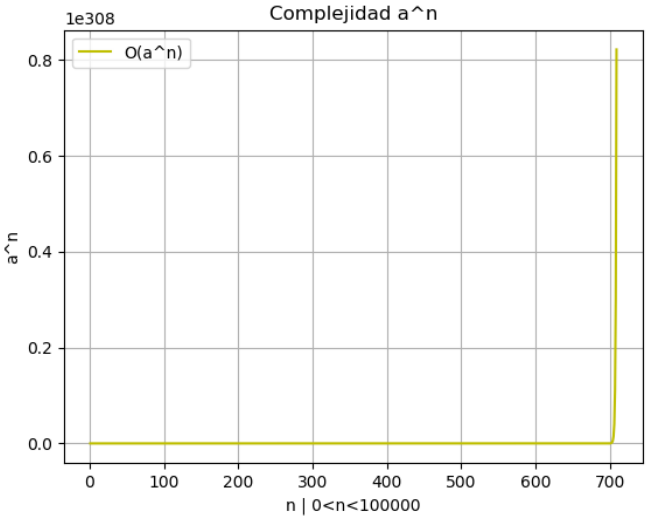
```python
#Se importa la librería de graficación para python
from matplotlib import pyplot
#Funcion cuadratica recive el parametro x evaluado de 0<x<100,000
def f(x):
    y=x**3
    return y
#se establece un dominio de la funcion donde 0<x<100,000
x=range(1,99000)
#Se grafica la funcion con un color de linea amarillo y una etiqueta
#el ciclo funciona para cada valor de x en el rango establecido
pyplot.plot(x,[f(i) for i in x],'y',label="O(n3)")
#Título de la gráfica
pyplot.title("Complejidad n3")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-n3.png")
#Se muestra la grafica
pyplot.show()
```



• $\boldsymbol{O}\ \boldsymbol{c}^{n}$; $\boldsymbol{c} > \mathbf{1}$ Complejidad exponencial

Quiere decir que hay una variable de control que la variación en el tiempo es proporcional a su valor, lo que implica que su valor sea más y más grande y crezca más rápido en el tiempo, por lo anterior se puede decir que el orden de complejidad más alto del algoritmo es exponencial.

```python
#Se importa la librería para sacar la exponencial
import numpy
#Se importa la librería de graficación para python
from matplotlib import pyplot
#Funcion cuadratica recive el parametro x evaluado de 0<x<100,000
def f(x):
    y=numpy.exp(x)
    return y
#se establece un dominio de la funcion donde 0<x<100,000
x=range(1,99000)
#Se grafica la funcion con un color de linea amarillo y una etiqueta
#el ciclo funciona para cada valor de x en el rango establecido
pyplot.plot(x,[f(i) for i in x],"y",label="O(a^n)")
#Título de la gráfica
pyplot.title("Complejidad a^n")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-exp.png")
#Se muestra la grafica
pyplot.show()
```

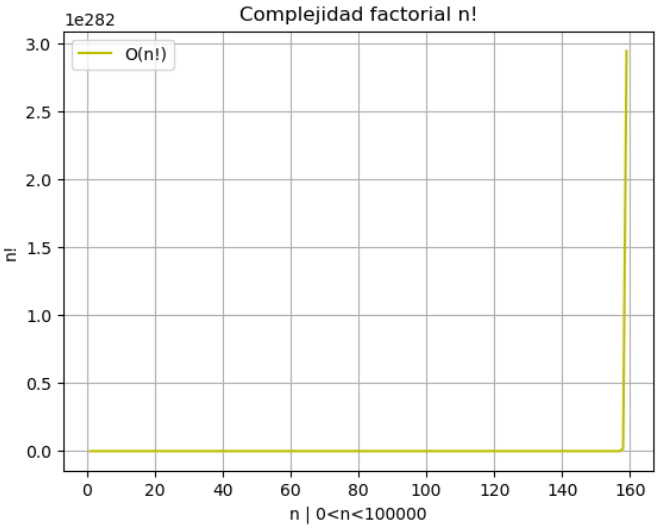


De la misma manera que la anterior gráfica, quiere decir que una constante mayor que 1 elevado a n's muy grandes llega un punto en el que se dispara.

• $\boldsymbol{O}(\boldsymbol{n}!)$ Complejidad factorial

Quiere decir que si existe una manera dentro del algoritmo de ordenar n elementos distintos y que no estén repetidos la función crecerá factorialmente y su orden de complejidad más alto es de factorial.

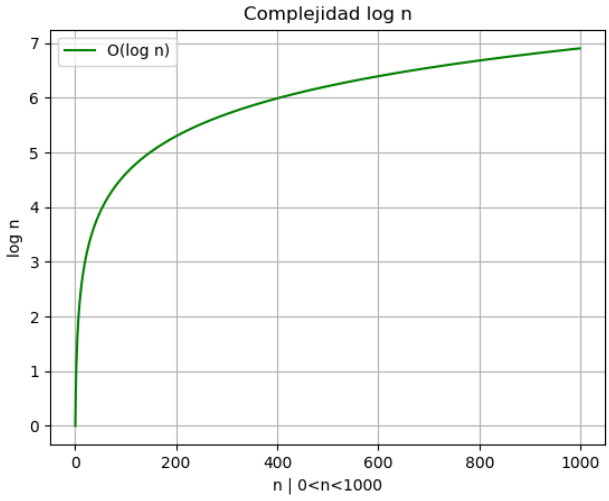
```python
#Se agrega la librería que calcula el factorial directamente
from math import factorial
#Se importa la librería de graficación para python
from matplotlib import pyplot
#Funcion cuadratica recive el parametro x evaluado de 0<x<100,000
def f(x):
    y=factorial(x)
    return y
#se establece un dominio de la funcion donde 0<x<100,000
x=range(1,160)
#Se grafica la funcion con un color de linea amarillo y una etiqueta
#el ciclo funciona para cada valor de x en el rango establecido
pyplot.plot(x,[f(i) for i in x],"y",label="O(n!)")
#Título de la gráfica
pyplot.title("Complejidad factorial n!")
#Una etiqueta en el eje de las x
pyplot.xlabel("0<n<100000")
#Se agrega una leyenda con la etiqueta de la funcion en la esquina sup izq
pyplot.legend(loc="upper left")
#Cuadriculado de la gráfica
pyplot.grid()
#Se guarda la gráfica con extensión .png en el mismo directorio
#en donde se encuentra el programa
pyplot.savefig("Complejidad-fact.png")
#Se muestra la grafica
pyplot.show()
```

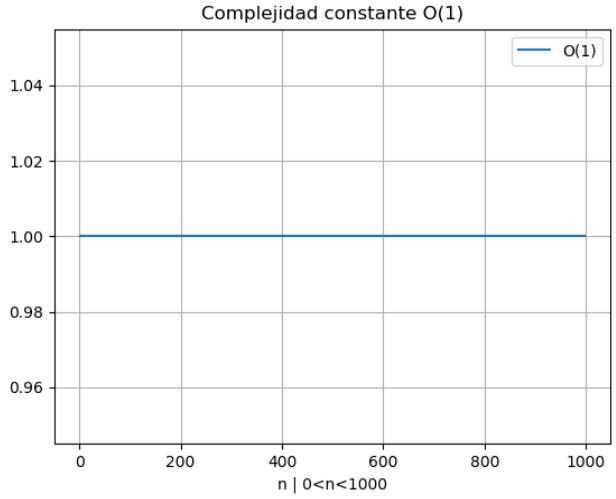


Se puede observar un comportamiento similar al de la exponencial, (99000)!≈2.9x10^{282}.

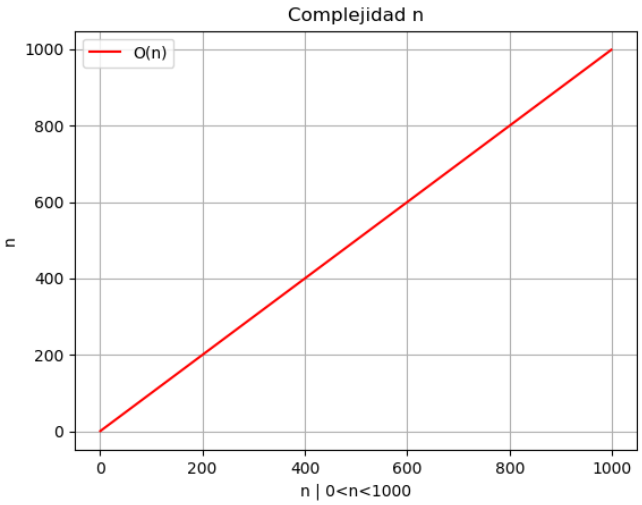
PARTE II: CONFRONTACIÓN DE ORDENES

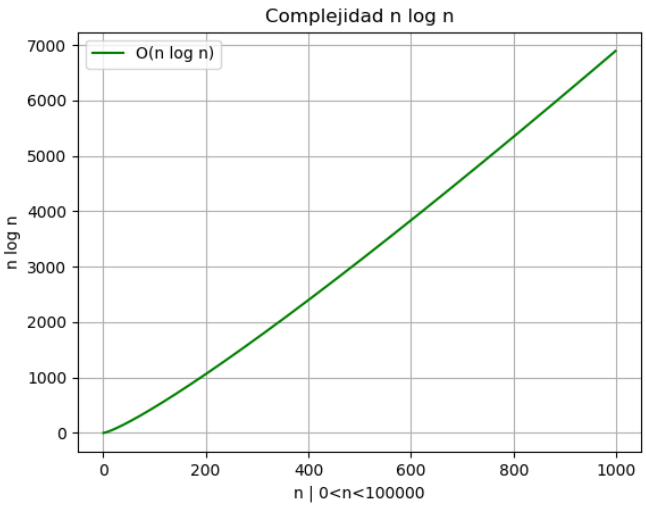
Para éste inciso se comparan en pares los ordenes de complejidad para hacer una decisión en la cual se elegirá una función por encima de la otra de acuerdo a sus características.



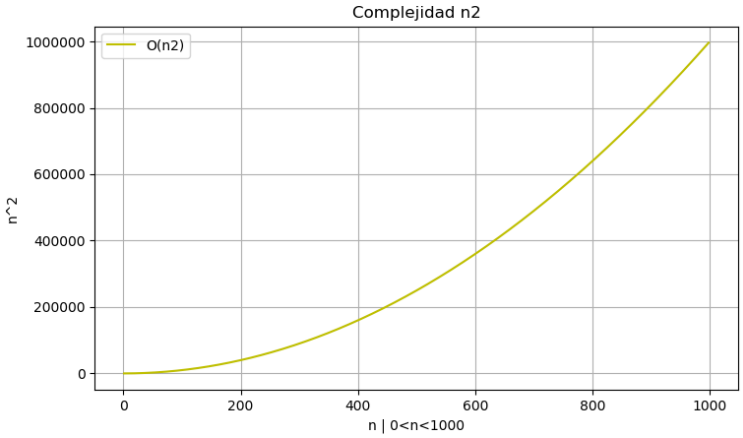


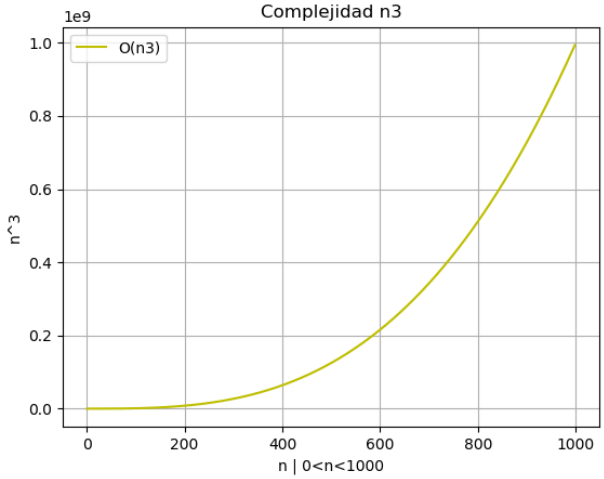
La complejidad log n domina asintóticamente a la complejidad constante, si se tuviera que elegir alguna sería la complejidad log n puesto que computacionalmente se ajusta para cada entrada n, y modela el comportamiento de la complejidad constante.



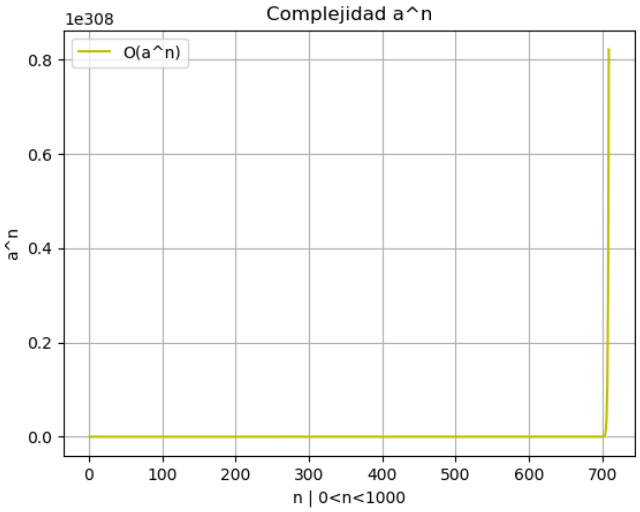


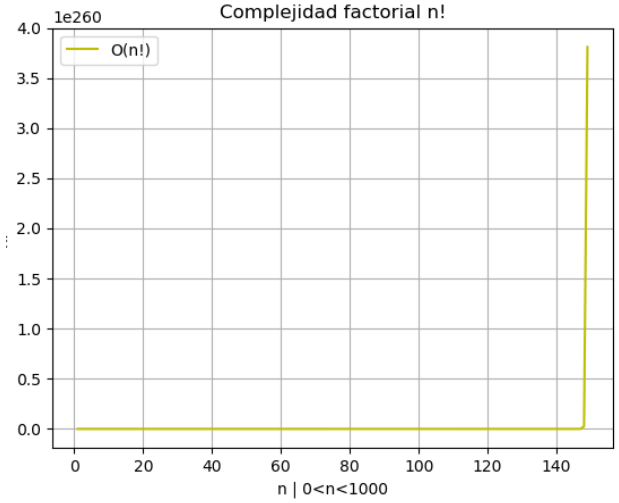
La complejidad n, como es lineal, la función de salida será la misma entrada n, y en la complejidad n log n será de manera logarítmica, se nota que en n log n es parecida a la lineal pero hace una pequeña curva. Si se tuviera que elegir a una función sería la complejidad n log n, ya que domina asintóticamente a la complejidad lineal y modela el comportamiento de ésta.





La complejidad n^2 y n^3 son bastante parecidas si a la curva se refiere, pero aun así es evidente que la n^3 domina asintóticamente a n^2 y por tanto es la mejor opción para elegir una función que modela a las demás que están por debajo de ella.





Éstas 2 funciones como lo son la exponencial y la factorial de una entrada muy grande n se aprecia bastante claro que se dispara hacia arriba pero se alcanza a apreciar que en un rango exacto en una entrada exacta la función exponencial se queda un poco después de dicha entrada y la salida de una factorial será para esa entrada exacta un crecimiento que modela a la complejidad exponencial, por lo que es elegible para usarse una función factorial por encima de todas las anteriores puesto que domina asintóticamente a la complejidad exponencial y a todas sus antecesoras.