Ejercicio 2 – Daniel Marín López
Sistemas de Big Data

Índice

Parte 1: Análisis de la complejidad algorítmica

Ejercicio 1

Considera un algoritmo de complejidad O(n) donde se recorre un vector para calcular la suma de sus elementos y contar cuántos elementos tiene.

La función fA(n) vale $f_A(n)$ = 2n + 1. Esto se debe a que para cada elemento del vector se realiza una operación de suma y una operación de conteo, lo que da 2n operaciones. Además, hay una operación adicional para inicializar la suma y el contador al inicio.

a) ***¿Cuántas operaciones realizará el algoritmo para un vector de 15 elementos si k vale 1?***

Pues si n = 15 eso equivale en la función a $f_A(15) = 2 \cdot 15 + 1$ que es igual a 31 operaciones.

b) ***¿Qué significaría que k fuera mayor que 1 y qué ocurriría en tal caso?***

Como k=1 eso significa que las operaciones se realizarían en un solo ordenador, si se aumentara k entonces tendríamos que distribuir el n.º de operaciones entre las k máquinas disponibles.

Ejercicio 2

Considera un algoritmo que calcula el n-ésimo número de la serie de Fibonacci de forma recursiva. Este algoritmo tiene una complejidad de $O(2^n)$.

La función $f_A(n)$ vale $f_A(n) = 2^{n-1} - 1$. Esto se debe a que cada llamada recursiva genera dos llamadas adicionales, lo que resulta en un crecimiento exponencial.

a) ***¿Cuántas operaciones aproximadas realizaría el algoritmo sin considerar el valor de k?***

Si tenemos valores muy grandes, el – 1 se desprecia. Por lo tanto el n.º de operaciones aproximadas que realizaría el algoritmo es de 2^{n-1}.

b) ¿Cómo cambiaría el número total de operaciones si el tamaño de la entrada fuera 6?

Si n = 6 entonces tendríamos lo siguiente $f_A(6) = 2^{6-1} - 1$ que equivale a 31 operaciones en total.

Parte 2: Análisis de la complejidad algorítmica a través de representaciones gráficas

Ejercicio 3

Carga en Google Colab este código.

```
# IMPORTACIÓN DE LIBRERÍAS
from matplotlib import pyplot as plt # para representación gráfica
import numpy as np # cálculo matemático
import math # funciones matemáticas

# CUERPO DEL PROGRAMA
m=20. # creación de una variable de tipo float (por el .)
n=np.arange(m) # creación de vector: n = [0, 1, 2, 3, ..., 19]
y=np.zeros(len(n)) # inicialización del vector a 0
for i in  range(len(n)):
  # A continuación se debe crear la función y[i] a representar. Comentar las que no se quiera representar dejando
  #y[i]=i**2 # función i al cuadrado (función polinomial): tiene crecimiento polinomial
  y[i]=3**i # función i al cuadrado (función polinomial): tiene crecimiento polinomial
  #y[i]=float(math.factorial(i)) # función factorial (función superexponencial o suprageométrica): tiene crecimien
  #print(y[i])
  print(f"y[{i}] = {y[i]}")
plt.plot(n,y,color="c") # crea la gráfica con los valores del vector n en el eje x y los valores del vector y en e
plt.yscale("log") # cambia la escala a representación logarítmica en el eje y
plt.show() # muestra la gráfica creada
```

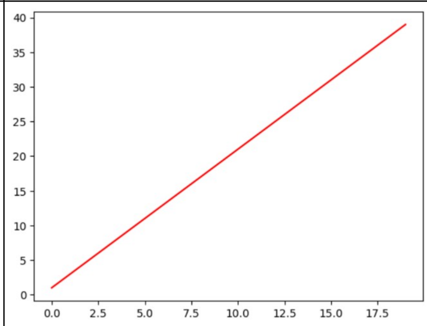
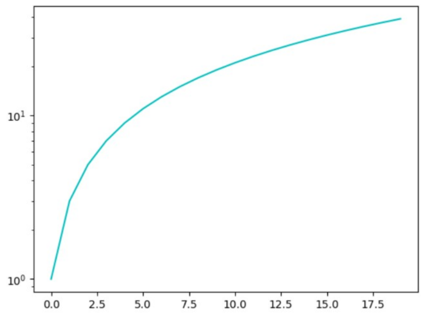
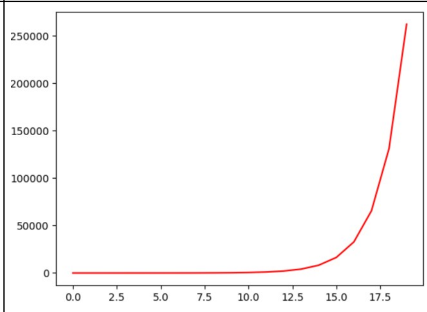
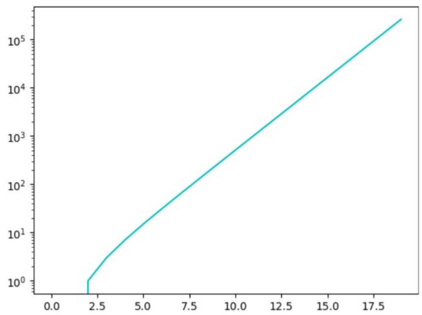
Figura 1: Código que genera gráficas

Entiéndelo. Ejecútalo. Luego, crea y representa estas dos nuevas funciones:

- $f_A(n) = 2n + 1$
- $f_A(n) = 2^{n-1} - 1$

a) Para cada una de ellas, ejecuta el código, copia y pega las gráficas que has obtenido en cada escala y analiza los resultados, explicando si la función es polinómica o exponencial y discutiendo cómo se comportan las gráficas obtenidas.

Las funciones representadas en el programa son las siguientes:

<table>
<tr><th>Función</th><th>Escala Lineal</th><th>Escala Logarítmica</th></tr>
<tr><td>$f_A(n) = 2n + 1$</td><td>
</td><td>
</td></tr>
<tr><td>$f_A(n) = 2^{n-1} - 1$</td><td>
</td><td>
</td></tr>
</table>

La primera función es polinómica ya que se ve como crece en forma de recta mientras que su escala logarítmica muestra como se curva poco a poco.

La segunda función es exponencial ya que crece muy rápido con valores de n muy alto, mientras que su escala logarítmica crece en forma de línea recta.

b) ***Explica por qué es importante visualizar los resultados en las distintas escalas.***

Visualizar la función en distintas escalas nos viene bien para ver el comportamiento de aquellas funciones de tipo exponencial ya que crecen muy rápido con valores más altos.

c) ***Reflexiona sobre la importancia de entender la complejidad algorítmica en el contexto de Big Data.***

Si trasladamos la complejidad algorítmica al Big Data nos daremos cuenta que es fundamenta saber manejar este tipo de conceptos ya que nos ayuda a evaluar la eficiencia y escalabilidad de algoritmos que trabajan con grandes volúmenes de datos. La complejidad algorítmica nos ayuda a prever el comportamiento de un algoritmo, sobre todo en relación a su tiempo de ejecución y uso de memoria, aspectos muy críticos para el Big Data.