**Actividad 1.2**

**- PARTE 1 -**

**Análisis de la complejidad algorítmica**

**Objetivo**

El objetivo de esta actividad es comprender la diferencia en el crecimiento de las funciones polinomiales y exponenciales, así como la importancia de la complejidad algorítmica al abordar problemas de Big Data.

**Introducción**

En el ámbito de la informática y la ciencia de datos, es crucial entender cómo el crecimiento de las funciones impacta la eficiencia de los algoritmos, especialmente cuando se trabaja con grandes volúmenes de datos.

**Conceptos clave**

* **Complejidad algorítmica**: se refiere a cómo varía el **rendimiento** de un algoritmo (en términos de tiempo de ejecución o uso de recursos) al resolver un problema, en función del **tamaño de la entrada**. La notación **Big O** se utiliza para **clasificar algoritmos según el número de operaciones que realizan en relación con el tamaño de la entrada**. Describe **cómo el número de operaciones crece a medida que aumenta el tamaño de la entrada**, centrándose en el **peor de los casos**. Al considerar el peor de los casos, los analistas pueden asegurarse de que el algoritmo funcione adecuadamente incluso en las situaciones más exigentes.

Esta notación se expresa generalmente en la forma **O(f(n))**, donde f(n) es una función que describe el **comportamiento del número de operaciones en función del tamaño de la entrada n**. Sin embargo, el rendimiento real de un algoritmo también depende de una **constante k** que puede variar según la implementación y el contexto (eficiencia del código, naturaleza de los datos de entrada, hardware de la máquina, etc.). Así, podemos expresar la relación de operaciones como k\*f(n), donde k es un factor que ajusta el número de operaciones a un valor real.

**Ejemplo de notación en Big O**:

* **O(1)**: Tiempo constante. El algoritmo tarda el mismo tiempo sin importar el tamaño de la entrada.
* **O(n)**: Tiempo lineal. El tiempo de ejecución crece de manera proporcional al tamaño de la entrada.
* **O(n2)**: Tiempo polinomial cuadrático. El tiempo de ejecución crece con el cuadrado del tamaño de la entrada. Este tipo de complejidad suele aparecer en algoritmos que requieren comparar cada elemento con cada otro (como en el caso de algunos algoritmos de ordenación).
* **O(2n)**: Tiempo exponencial. El tiempo de ejecución se duplica con cada elemento adicional en la entrada, lo que lo hace impracticable para tamaños de entrada grandes.
* **Funciones f*A*​(n)**: representa la **complejidad específica** de un algoritmo *A*. Es una función que describe el **número exacto de operaciones** que realiza el algoritmo para diferentes tamaños de entrada n.

**Ejemplo:**

Tenemos un algoritmo de complejidad **O(n2)**, como es el caso de **ordenar un vector por el método de la burbuja**. Si el tamaño del vector es de **5 elementos**, no significa que el algoritmo realice exactamente 52=25 operaciones. En cambio, lo que implica es que:

* **A medida que el tamaño de la entrada aumenta**, el número de operaciones se comportará como si se aproximara a k\*n2, donde k es alguna constante que depende del algoritmo específico.

Para calcula el número de operaciones que realizará el algoritmo, tenemos que calcular **f*A*​(5)**. Como **f*A*​(n) = (n\*(n-1))/2**, en este caso **f*A*​(5) = 10**, por lo que el algoritmo se completará en 10 operaciones.

**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 f*A*(n) vale f*A*(n)=2\*n+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 2\*n operaciones. Además, hay una operación adicional para inicializar la suma y el contador al inicio.

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

Si tenemos una funcion de 2\*n+1 y k = 1 el resultado será en un algoritmo de complejidad de tiempo lineal será:

f(15) = 2\*15+1 = 31

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

Indica que cada operación tomará más tiempo o que hay operaciones adicionales por cada iteración sobre el algoritmo. Esto podría reflejar, por ejemplo, una ejecución más lenta debido a factores como una carga adicional de procesamiento o una complejidad mayor en cada operación individual.

**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(2n).

La función f*A*(n) vale f*A*(n)=2n-1-1. Esto se debe a que cada llamada recursiva genera dos llamadas adicionales, lo que resulta en un crecimiento exponencial.

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

En esta situación el número de operaciones dependerá exclusivamente de n, por lo que para valores pequeños de n, el número de operaciones será bajo, ya que se trata de un complejo algoritmico exponencial. Para valores altos puede crecer enormemente. Dependerá del tamaño de entrada que introduzcamos.

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

Tendríamos entonces que:

f*A*(6)=26-1-1

f*A*(6)= 31

Tendremos que el algoritmo realiza 31 operaciones.

**- PARTE 2 -**

**Análisis de la complejidad algorítmica a través de representaciones gráficas**

**Objetivo**

El objetivo de esta actividad es que comprendas el crecimiento de funciones polinómicas y exponenciales mediante representaciones gráficas, utilizando código Python. Aprenderás a diferenciar entre escalas lineales y logarítmicas y analizarás las implicaciones de cada tipo de función en el contexto de la complejidad algorítmica.

**Introducción**

En el estudio de la complejidad algorítmica, es fundamental entender cómo crecen las funciones matemáticas que describen el rendimiento de los algoritmos. La representación gráfica de estas funciones permite visualizar el comportamiento de los algoritmos a medida que aumenta el tamaño de la entrada. A través de esta actividad tienes la oportunidad de implementar código y explorar gráficamente las diferencias entre funciones polinómicas y exponenciales.

**Conceptos Clave**

* **Escala lineal**: los intervalos entre los valores representados en el eje (ya sea el eje x o y) son **constantes**. Es decir, cada incremento en los valores del eje es uniforme.

Si representamos una función como y=n, un incremento de 1 en n siempre producirá el mismo incremento en el valor de y.

Ejemplo:

* Si tenemos los valores n=1,2,3,4,5, el gráfico en escala lineal los mostrará igualmente espaciados.



* En una gráfica de escala lineal, las funciones polinómicas (como y=n2 o y=n3) y exponenciales (como y=2n) mostrarán un crecimiento que se acelera a medida que n aumenta.



* **Escala logarítmica**: los intervalos **no son constantes**, sino que aumentan de forma exponencial. El valor en el eje es proporcional al logaritmo de los datos reales, lo que comprime valores grandes y expande valores pequeños. Esto es útil para representar funciones que crecen muy rápido, como las exponenciales y factoriales, permitiendo una visualización más clara.

Ejemplo:

* Si representamos y=10,100,1000,10000 en una escala logarítmica, el gráfico mostrará estos valores con el mismo espacio entre ellos porque la escala se basa en potencias de 10.
* En una gráfica logarítmica, las funciones exponenciales se ven como líneas rectas, y las funciones polinómicas se ven con pendientes menos pronunciadas, reflejando su menor crecimiento comparado con las exponenciales.

**RECUERDA**: ¿Cómo se calcula el logaritmo? Vamos a verlo con un ejemplo:

El logaritmo en base 10 de un número x se define como el número y que satisface la ecuación:

10y=x

Por ejemplo, para calcular log⁡10(20):

10y=20

Tienes que encontrar y, que es el valor al que debes elevar 10 para obtener 20. El valor resultante es aproximadamente:

log⁡10(20) ≈ 1,3010

Esto significa que 101.3010 ≈ 20

|  |  |  |
| --- | --- | --- |
| **EJEMPLOS** | **Escala lineal** | **Escala logarítmica** |
| **Función polinómica**  Ejemplo:  y = n3 |  |  |
| Crecen más despacio que las exponenciales, pero aún muestran una aceleración significativa para valores grandes de n. | Sigue un comportamiento lineal, pero con pendientes menores que las exponenciales, lo que indica que el crecimiento es más lento en comparación con estas. |
| **Función exponencial**  Ejemplo:  y = 2n |  |  |
| Crecen extremadamente rápido, mucho más que los polinomios. Al aumentar n, el valor de la función se dispara. | Se comportan como líneas rectas, lo que refleja un crecimiento exponencial constante. |

**EJERCICIO 3:**

Carga en Google Colab [este código](https://drive.google.com/file/d/1CbvBaRlc1QSQJTM0b5vll79RO1HjM6CB/view?usp=sharing). Entiéndelo. Ejecútalo. Luego, crea y representa estas dos nuevas funciones:

* fA(n) = 2\*n+1
* fA(n) = 2n−1−1
* 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.

Para fA(n) = 2\*n+1 :



Como podemos observar la función es polinómica en escala logarítmica, ya que tiene un crecimiento lineal, pero la curva indica un crecimiento más lento con respecto a las exponenciales.

Para fA(n) = 2n−1−1 :



Se trata de una función exponencial en escala logarítmica. Ya que al principio aunque crezca como una pequeña curva inclinada, luego crece de manera recta. Reflejando el exponente.

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

La escala lineal muestra el crecimiento real y absoluto.

La escala logarítmica es útil para apreciar el crecimiento inicial de funciones exponenciales o factoriales, que de otro modo serían difíciles de interpretar visualmente.

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

Comprender la complejidad algorítmica es crucial, donde los conjuntos de datos son enormes. Los algoritmos con crecimiento polinómico o logarítmico suelen ser manejables, mientras que los de crecimiento exponencial o factorial se vuelven impracticables a medida que crecen los datos.