# Análisis de Complejidad Computacional

## **Evaluar polinomio usando:**

## Potencia por multiplicaciones sucesivas: **O(n^2)**

La potencia es de complejidad de orden O(n). Entonces, la complejidad de evaluar el polinomio sería O(n^2)

## Potencia por recursividad: **O(n^2)**

La recursividad se considera de evolución lineal, entonces es complejidad de orden O(n). La complejidad de evaluar el polinomio sería O(n^2).

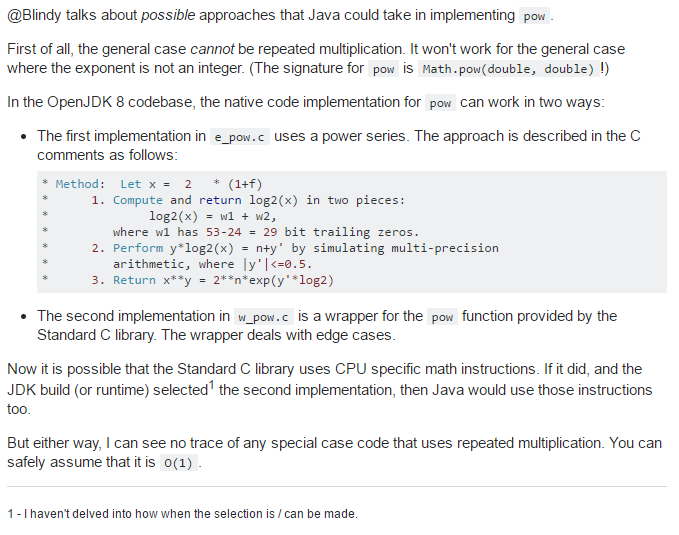
## Potencia por recursividad según potencia par/impar: **O(n log n)**

Si el exponente es par, tiene una evolución logarítmica. Si el exponente es impar, tiene una evolución lineal. El peor caso sería que se dé muchas veces un exponente impar. La complejidad de la potencia es de orden O(log n). Entonces, la complejidad de evaluar el polinomio sería O(n log n)

## Programación dinámica: **O(n)**

Las potencias se van calculando a medida que se necesitan, tomando como referencia el valor almacenado en el ciclo anterior. Como resultado, en un solo ciclo se consigue calcular las potencias y a su vez ir sumando los términos del coeficiente. Realiza n sumas y 2n multiplicaciones. Como se hace un solo recorrido, la complejidad sería O(n).

## Math.Pow: **O(n)**

Según la investigación llevada a cabo por el grupo, encontramos lo siguiente:

(fuente: <http://stackoverflow.com/questions/32418731/java-math-powa-b-time-complexity>)

De esta manera, asumiendo que Math.Pow tiene complejidad O(1). Podemos decir que la evaluación del polinomio tiene una complejidad de orden O(n).

## Horner: **O(n)**

Al evaluarse monomialmente un polinomio, se consigue evaluar al polinomio con un solo ciclo. Está demostrado que este algoritmo es óptimo, ya que minimiza el número de operaciones (realiza n sumas y n multiplicaciones). En consecuencia, la complejidad es de orden O(n)

# Conclusiones

Gráfico con las funciones de las CC involucradas en los algoritmos:

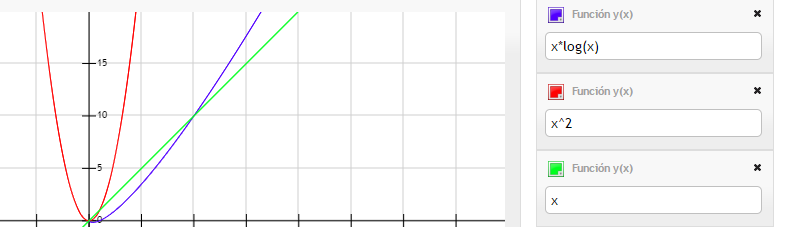


Gráfico con los tiempos de ejecución obtenidos:

Se puede notar que, si bien algunas complejidades se igualan en su orden, varían en su tiempo de ejecución.

Algoritmos con CC de igual orden:

**O(n):** Horner – Math.Pow – Programación dinámica

Analizando las implementaciones de cada uno, el algoritmo de Horner sería el más eficiente, ya que es el que realiza menor cantidad de operaciones. Sin embargo, al comparar los tiempos obtenidos, puede verse que tanto los tiempos de ejecución de los algoritmos Horner y Programación dinámica son casi iguales. Por otro lado, en los tiempos del cálculo del polinomio mediante el uso de Math.Pow se puede observar un leve incremento constante en el tiempo de ejecucióna medida que el grado del polinomio va aumentando.

Los tres son muy eficientes, destacándose Horner y Programación dinámica.

**O(n^2):** Multiplicaciones sucesivas – Recursividad

Si bien las complejidades computacionales de ambos algoritmos son iguales, es claramente visible en el gráfico de tiempos que el uso de multiplicaciones sucesivas es más eficiente. El algoritmo de recursividad implica usar la pila del sistema e ir gastando más memoria para mantener las variables utilizadas en tiempo de ejecución.

**O(n log n):** Recursividad exponente par/impar

Podemos observar en el gráfico de tiempos que el algoritmo se ejecuta en un tiempo aceptable.