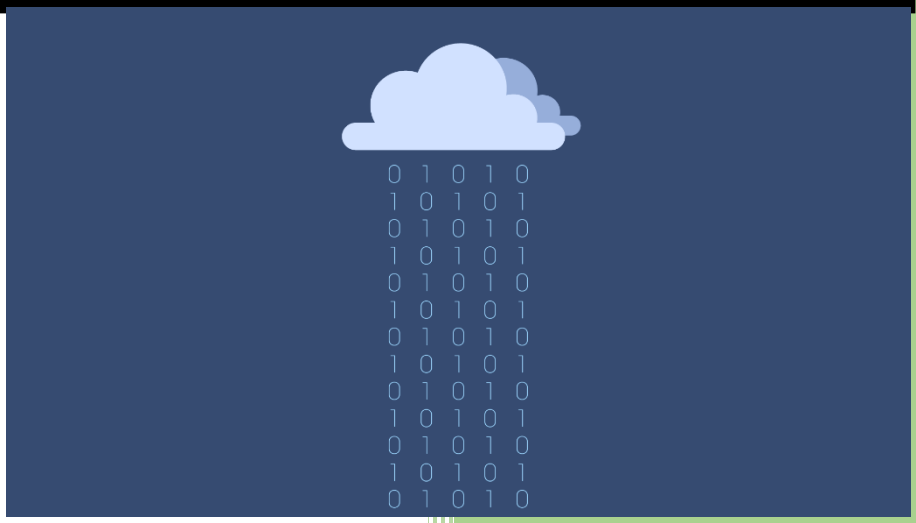
Análisis de algoritmos



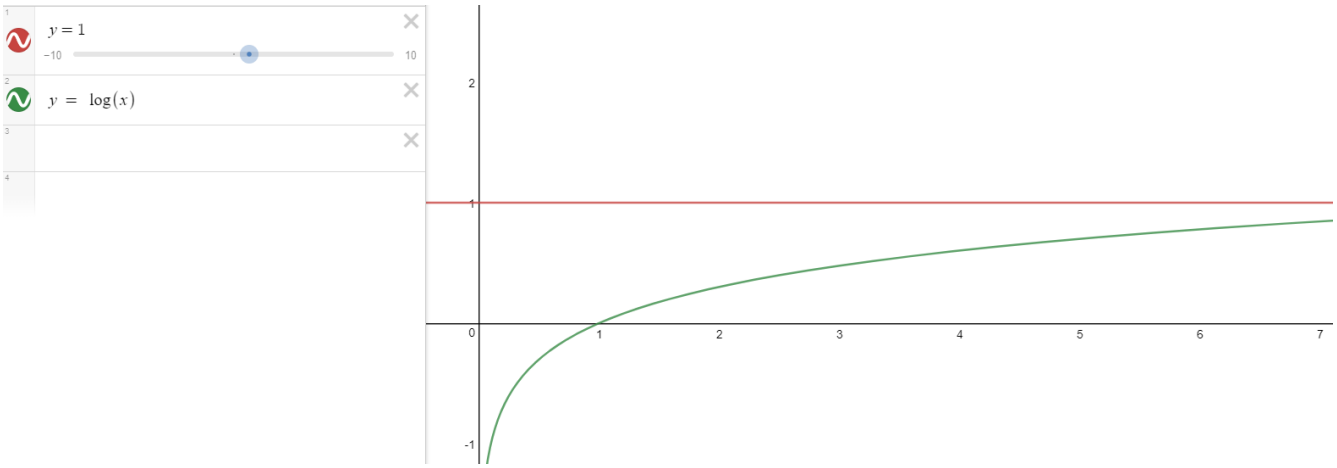
Vargas Romero Erick Efraín

Grupo 3CM2

Tarea 02: Complejidad temporal y análisis de casos

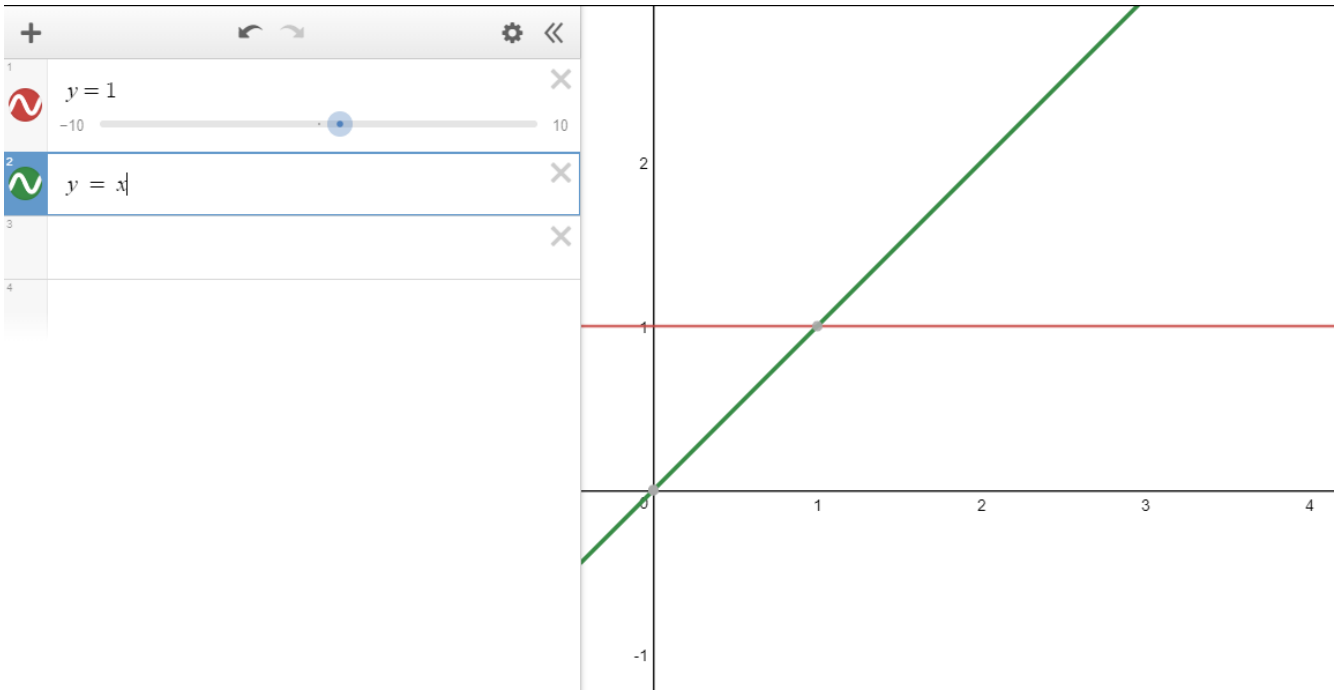
Fecha de entrega 06/09/2018

1) constante vs log(n)



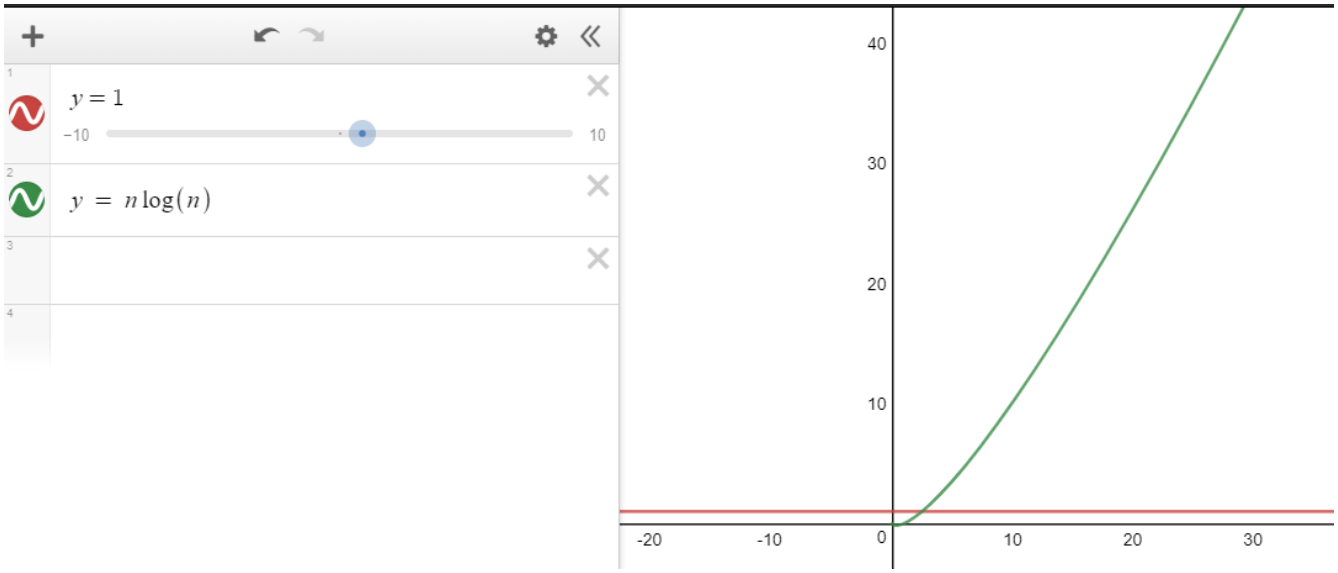
Como podemos observar la función complejidad constante es mejor cuando el tamaño de problema es demasiado grande, ya que para cualquier tamaño de problema siempre tardará lo mismo. Por otra parte, con un tamaño de problema pequeño la función complejidad log(n) es mejor ya que la solución la obtiene bastante rápido.

2) Constante vs n



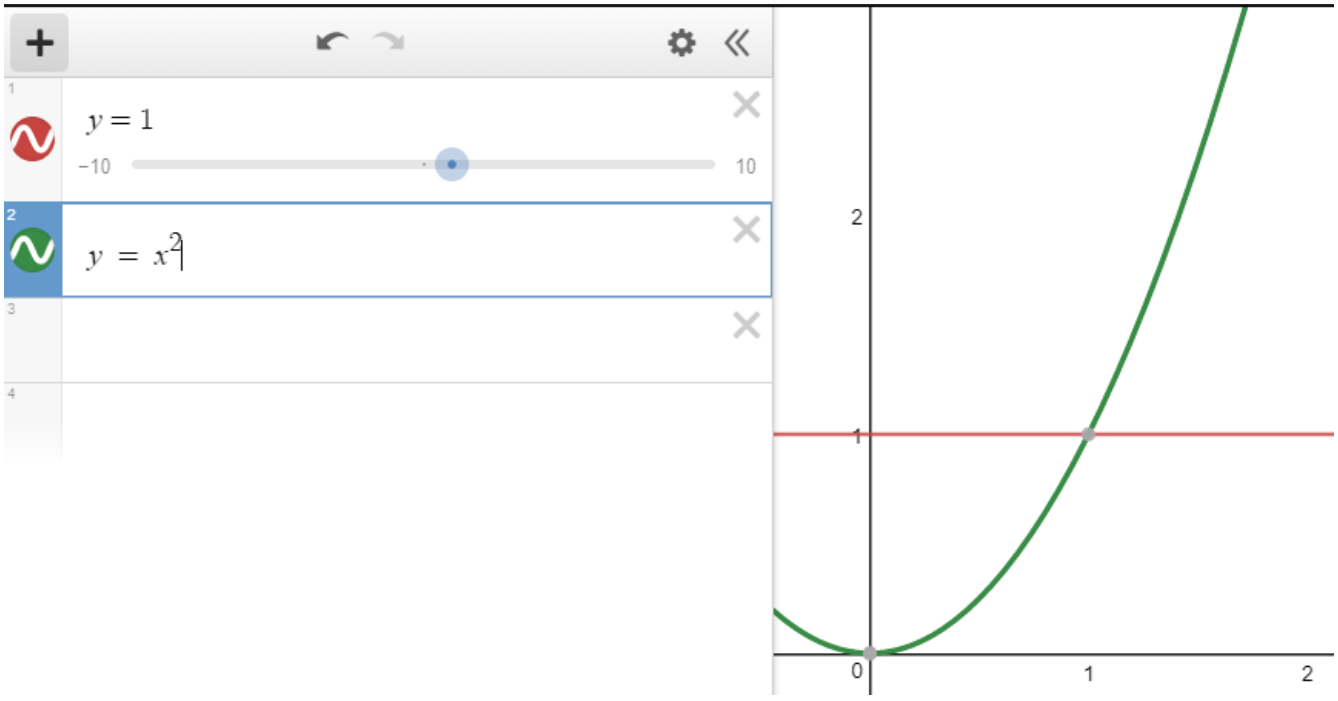
En esta gráfica podemos observar que ocurre lo mismo que en el caso anterior. La función complejidad que es constante, sin importar el tamaño del problema, llega a la solución siempre en el mismo número de pasos. Por otra parte la función complejidad n crece muchísimo más rápido que la función constante.

3) Constante vs n log(n)



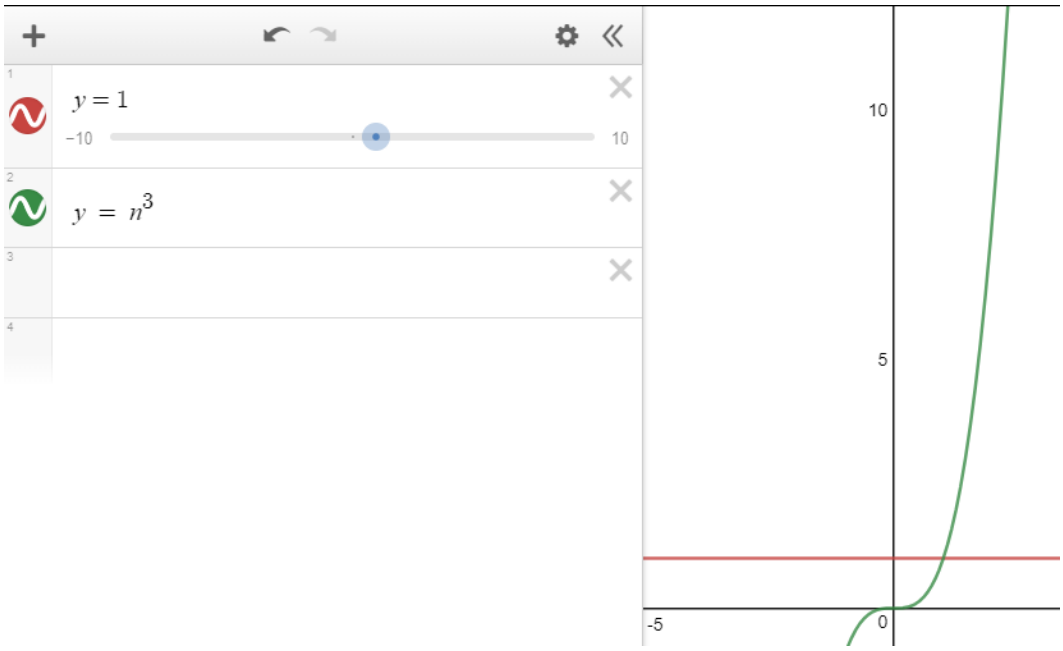
En esta gráfica nuevamente la función con la que estamos comparando nuestra función complejidad constante, crece muy, muy rápido, más aún que las comparaciones anteriores. Por tanto, se repite exactamente lo mismo, la función constante no sufre cambios aún que el tamaño del problema sea demasiado grande.

4) Constante vs n^2



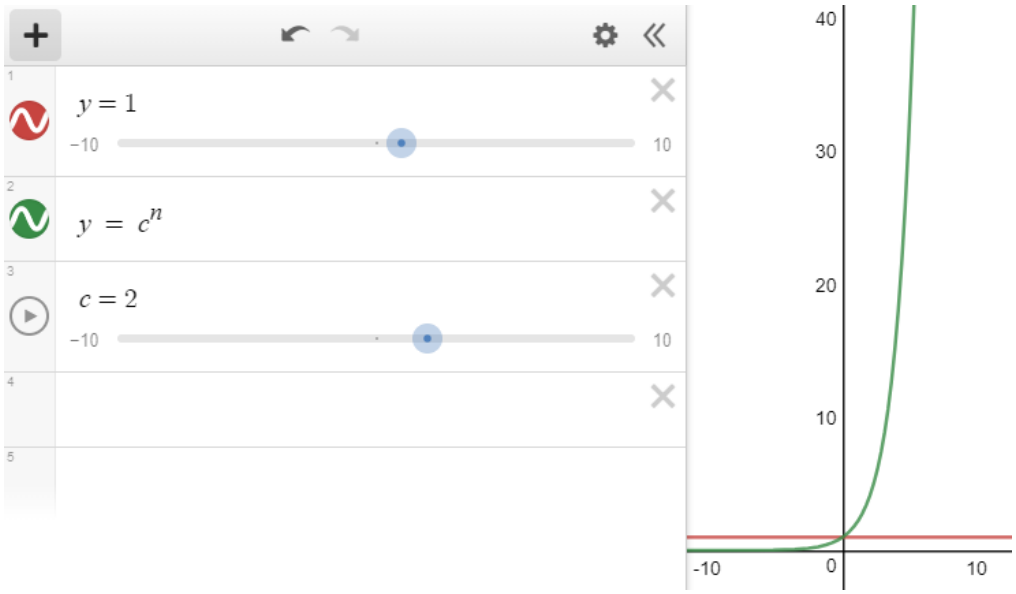
Nuevamente la función constante se mantiene igual que los casos anteriores, tiene un comportamiento mucho mejor que la función orden complejidad cuadrática ya que la función complejidad cuadrática para un tamaño de problema muy grande, el algoritmo no es eficiente en comparación con la función complejidad constante.

5) Constante vs n^3



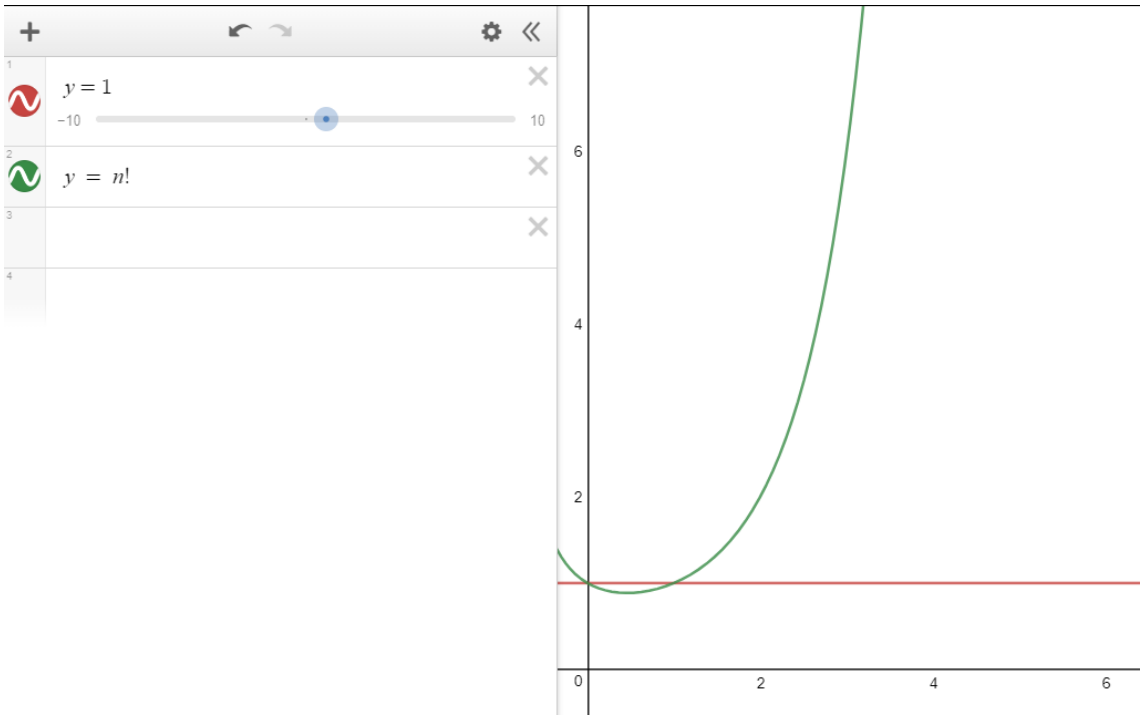
Nuevamente se repite lo mismo que en los casos anteriores, la función constante se comporta muchísimo mejor que la función complejidad cúbica. Si el tamaño de problema es muy grande, la función complejidad cúbica es en exceso ineficiente, aún más que la función complejidad cuadrática.

6) Constante vs C^n



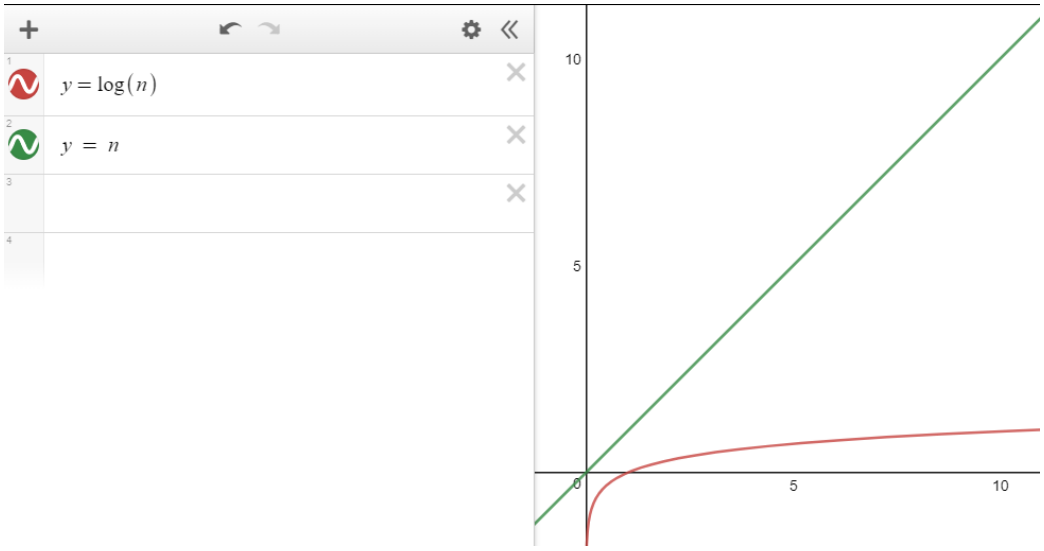
En esta gráfica podemos ver que la función complejidad exponencial es aún más ineficiente que la cúbica y cuadrática, el crecimiento de esta función es mucho mayor que las anteriores, y nuevamente la función complejidad constante, resuelve el problema en pocos pasos aún que el tamaño de problema sea muy grande.

7) Constante vs n!



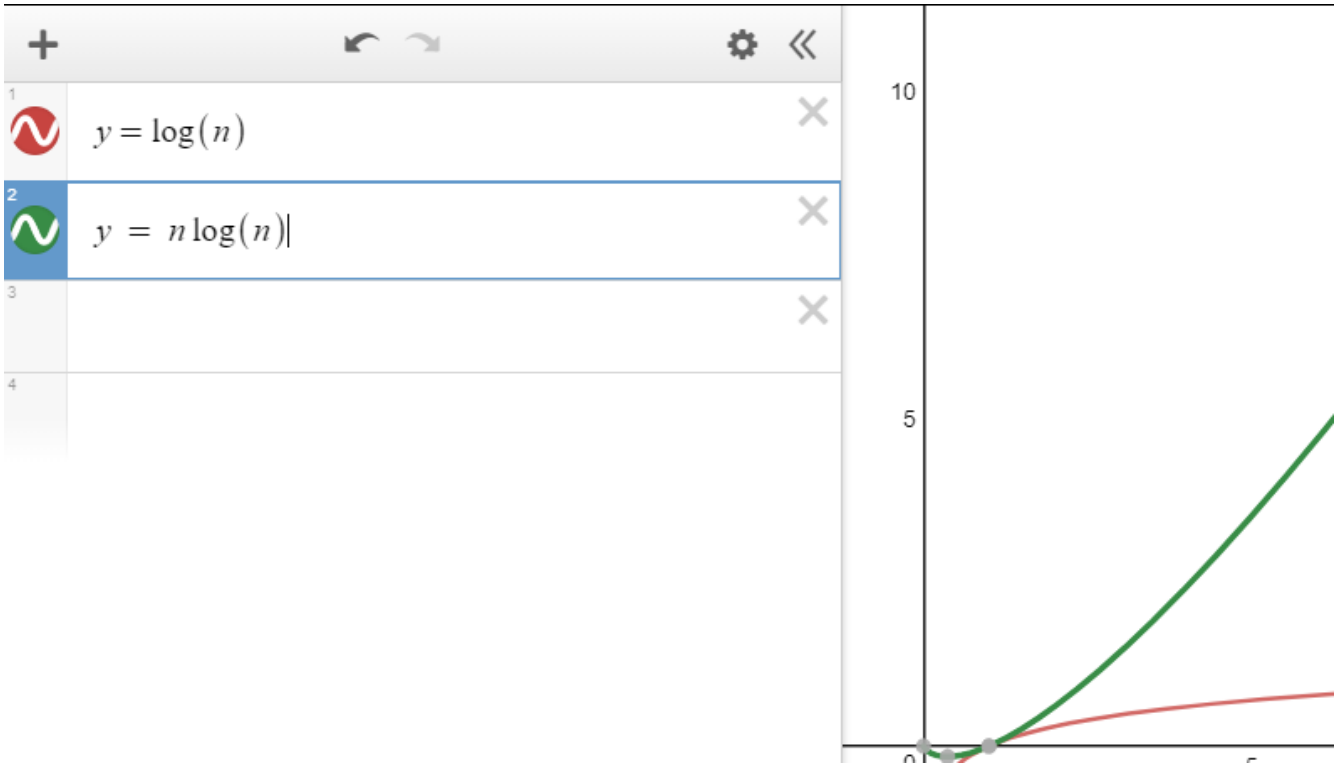
En esta gráfica, es bastante claro que sale ganando por muchísimo más que la función complejidad factorial. La función factorial, crece excesivamente más rápido, tanto que aún con una n pequeña resuelve el problema de una forma mucho más ineficiente, aún más que si fuese exponencial, cuadrática o cúbica.

8) log(n) vs n



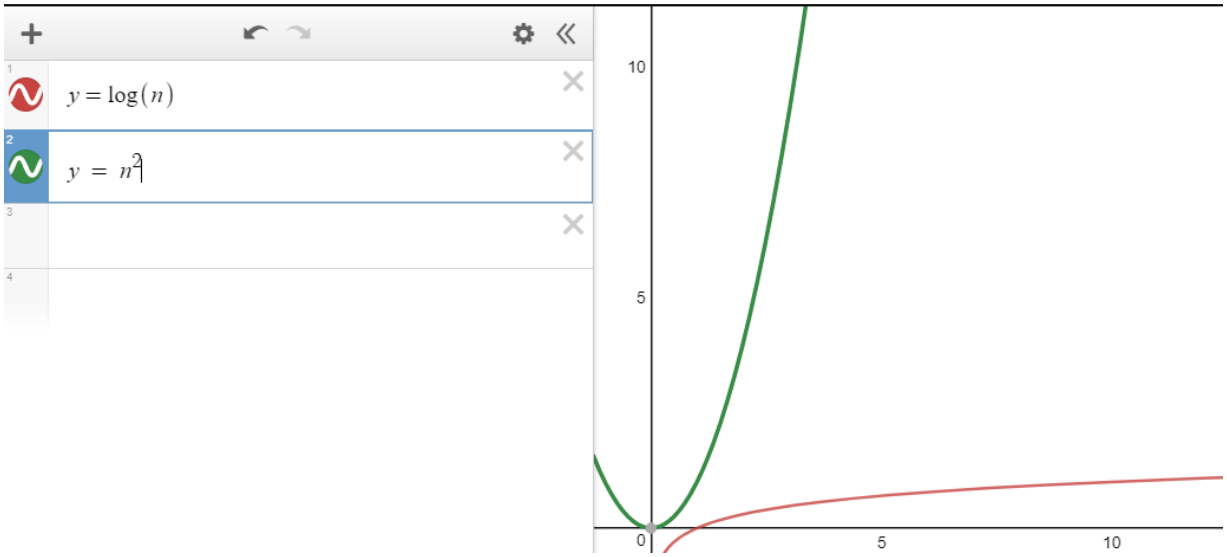
En esta comparativa podemos observar que la función logarítmica es considerablemente mejor que la función lineal, tanto para tamaños de problema pequeños, como unos muy grandes.

9) log(n) vs n log(n)



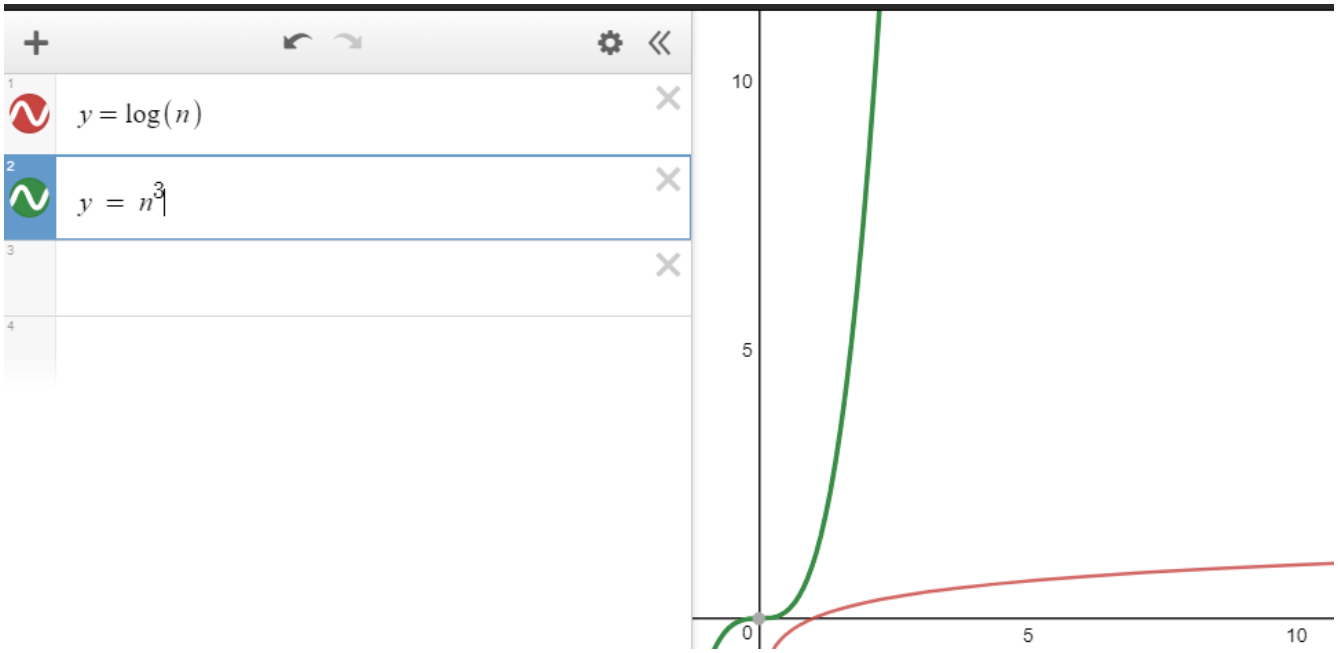
Nuevamente la función complejidad logarítmica sale ganando respecto a la función complejidad n log(n), y al igual que ocurría con la función lineal, tanto para casos donde el problema es muy pequeño como muy grande la función nlog(n) sale perdiendo.

10) log(n) vs n^2



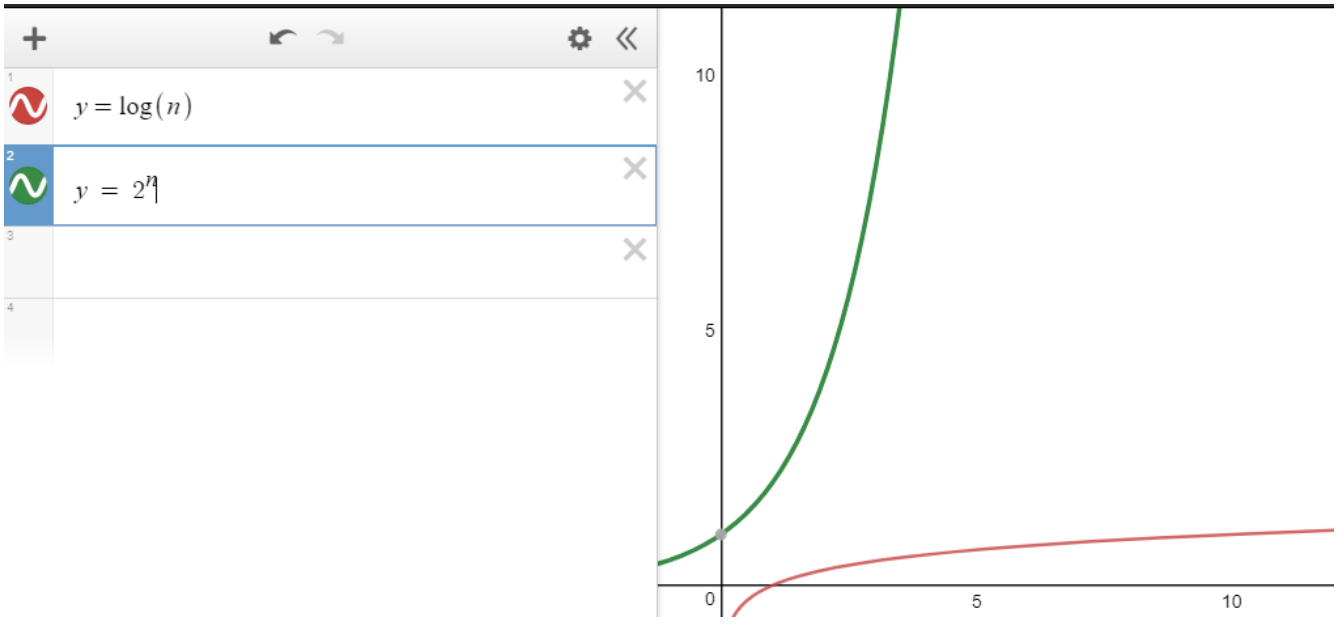
En esta comparativa, la diferencia en eficiencia puede notarse de una forma muy, muy clara, que la función cuadrática sake perdiendo, ya que su crecimiento es la multiplicación del tamaño del problema por si mismo.

11) log(n) vs n^3



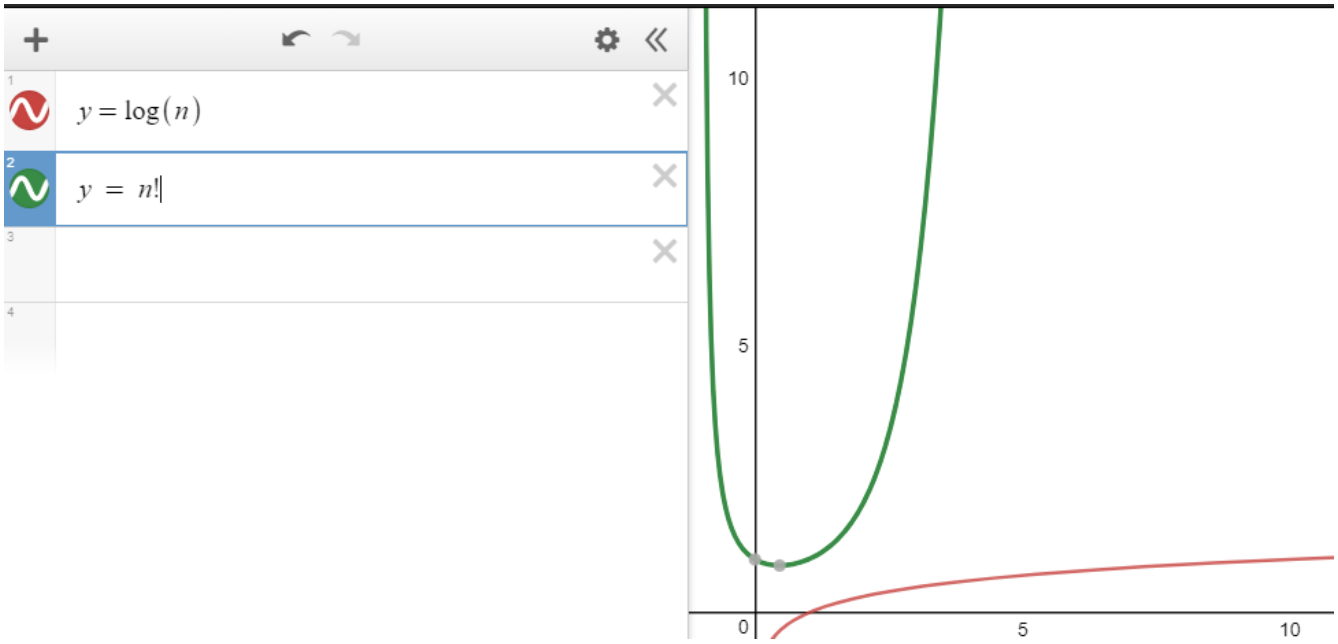
En este caso la diferencia en eficiencia entre la función complejidad logarítmica y la función complejidad cúbica es muy notable que la función complejidad logarítmica es más eficiente, al resolver problemas con un tamaño muy grande.

12) log(n) vs c^n



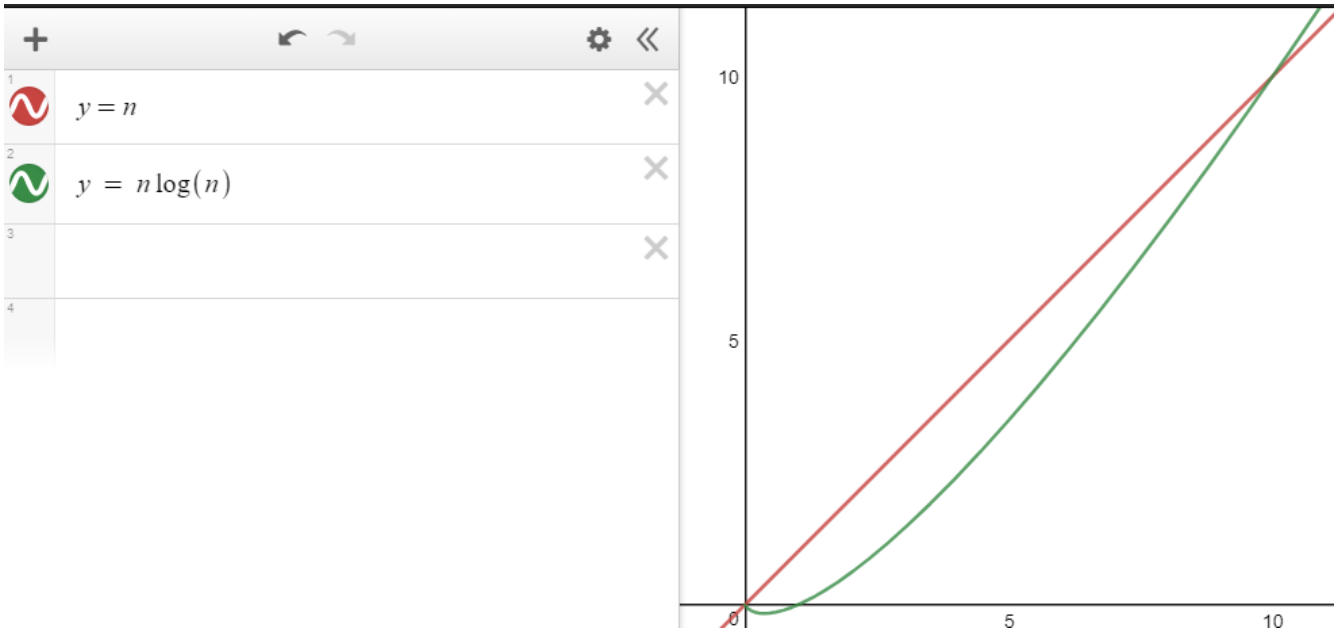
Nuevamente si comparamos estas dos funciones la función logarítmica sale ganando y ahora por una diferencia mucho mayor, la eficiencia entre una y otra es demasiada, siendo la función logarítmica considerablemente más eficiente para cualquier tamaño de problema n

13) log(n) vs n!



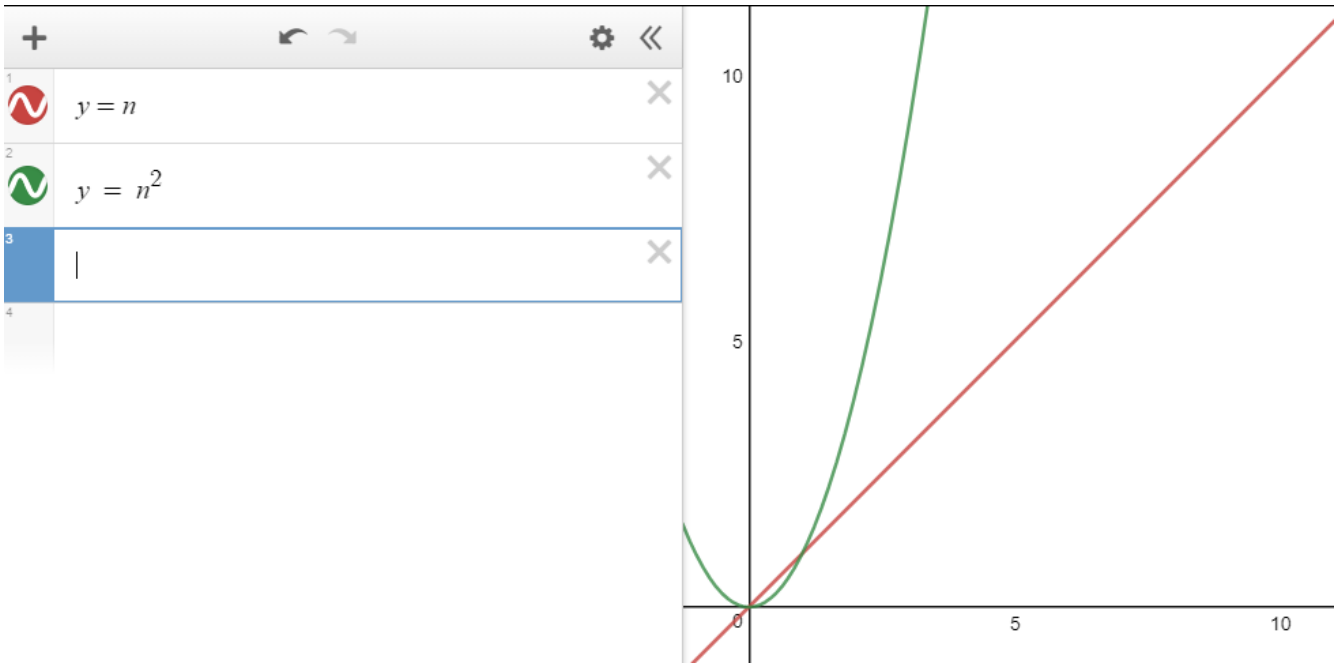
Y finalmente si debemos comparar la función logarítmica con la función factorial, la diferencia es aún mayor, siendo la función factorial en exceso ineficiente para cualquier tamaño de problema. Por tanto, la función logaritmo sale ganando también si es comparada con la función factorial.

14) n vs n log(n)



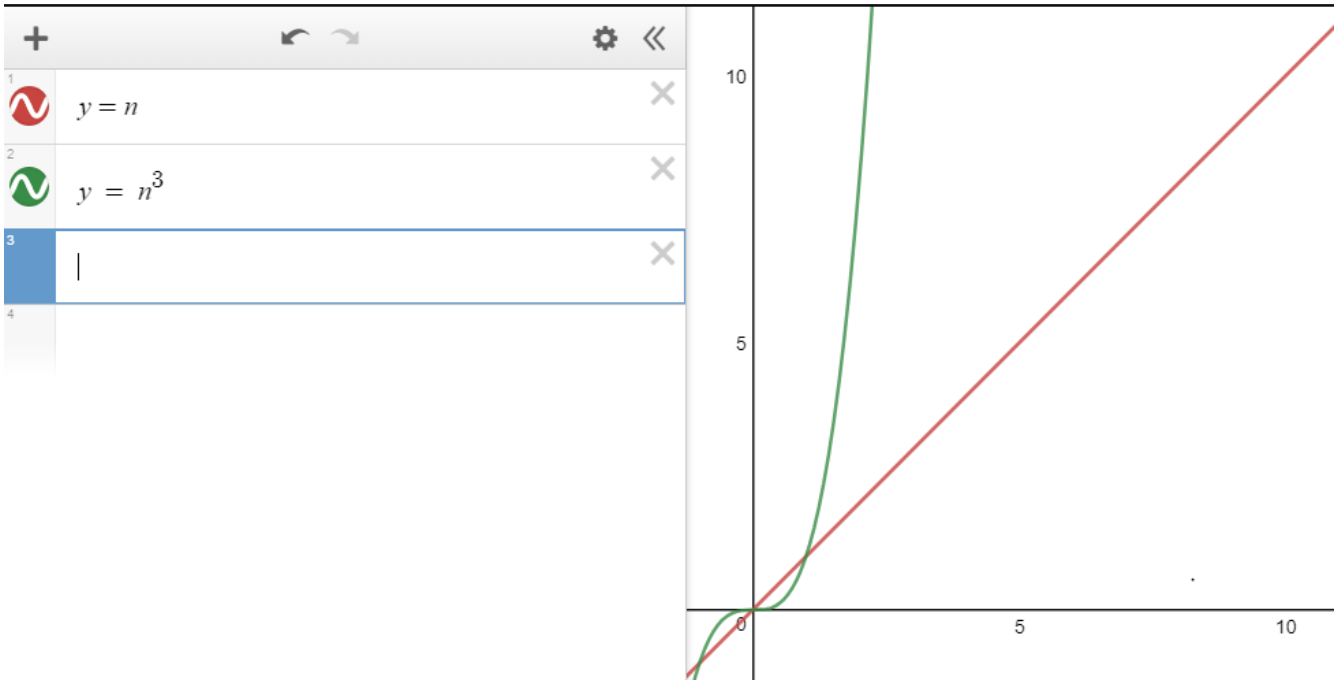
El análisis de estas dos funciones a pesar de que es un poco similar, podemos apreciar que la función lineal para tamaños de problema muy grandes es más eficiente que la función n log(n) ya que la función n log(n) podríamos decir que es lineal, pero multiplicada por un factor log(n), por tanto es obvio que será más eficiente la función lineal.

15) n vs n^2



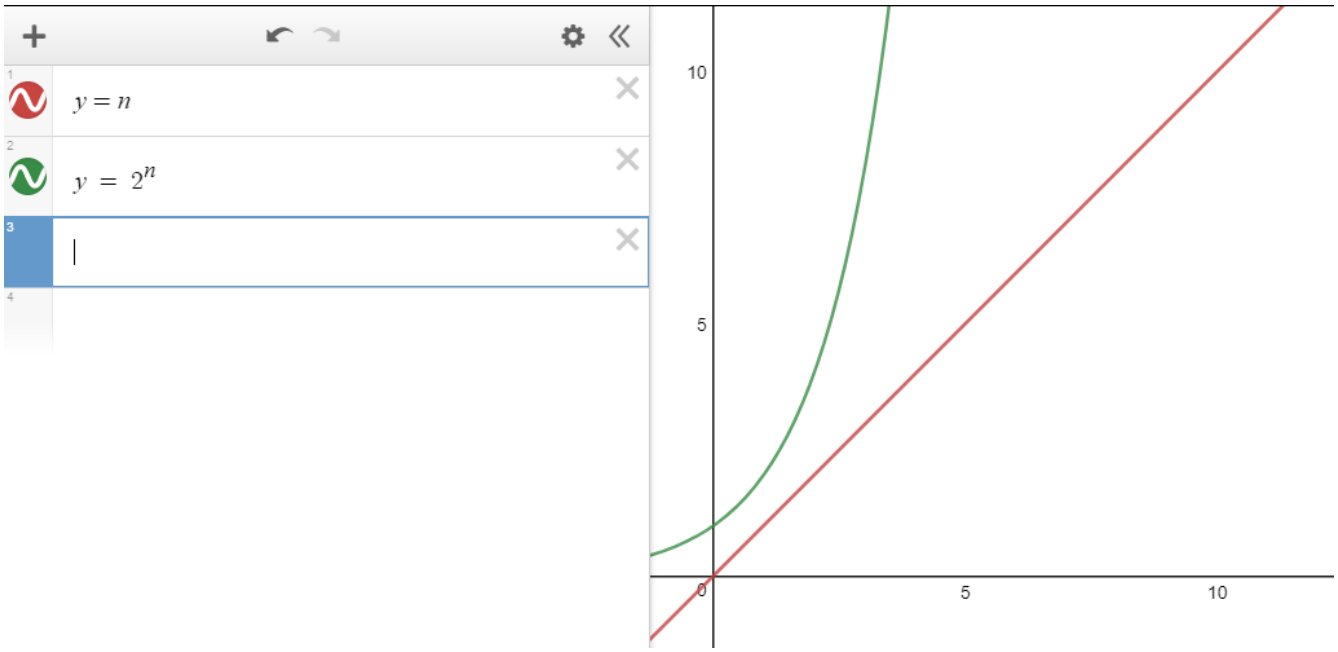
En esta comparación podemos apreciar que la función lineal, es mucho más eficiente que la función cuadrática, aún que el tamaño de problema sea muy grande, cosa que es bastante evidente de ver ya que la función cuadrática es el producto de la función lineal por sí misma.

16) n vs n^3



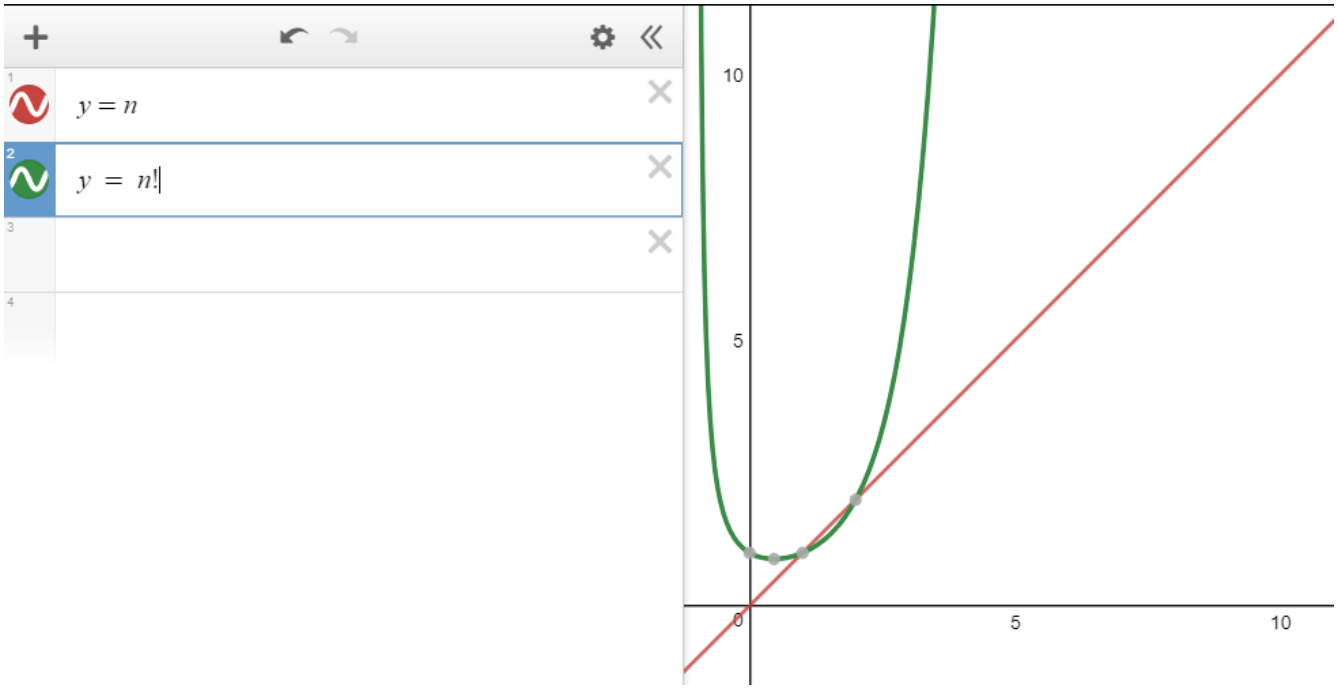
Esta comparación igual que las anteriores la función lineal es considerablemente mejor que la función cúbica, sobre todo con tamaños de problema demasiado grandes.

17) n vs c^n



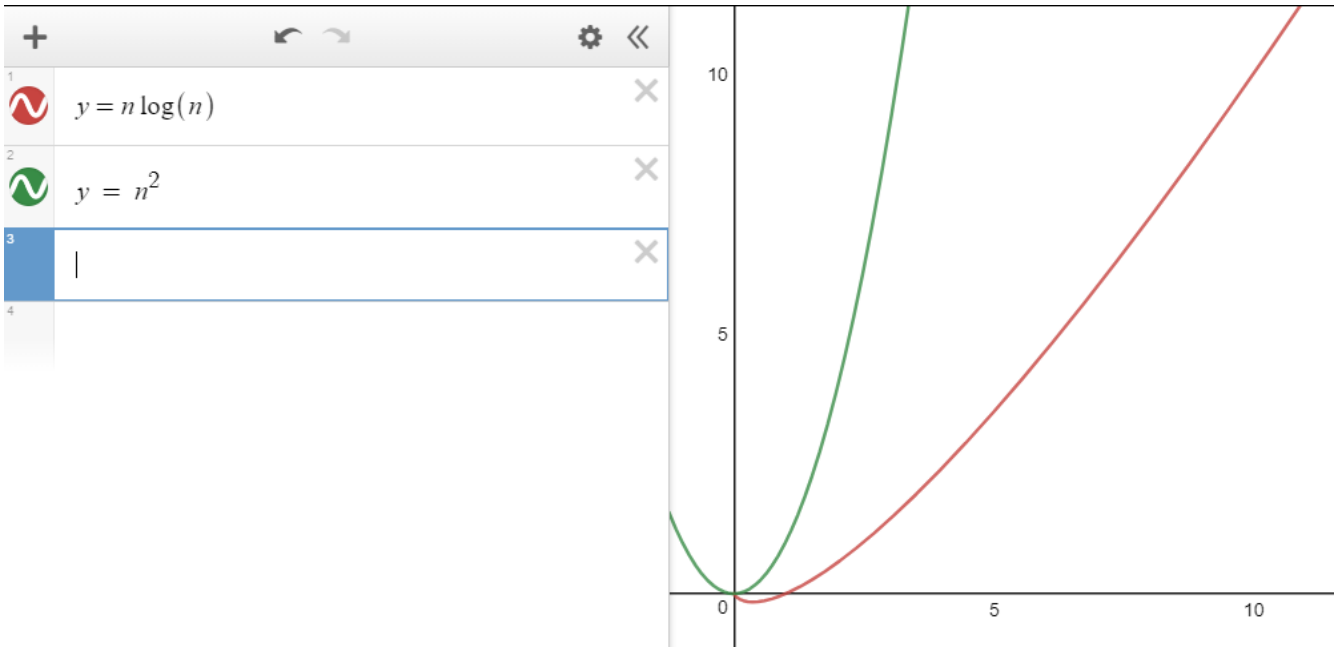
Nuevamente se repite lo mismo que los casos anteriores, la función lineal es considerablemente mejor que la función exponencial.

18) n vs n!



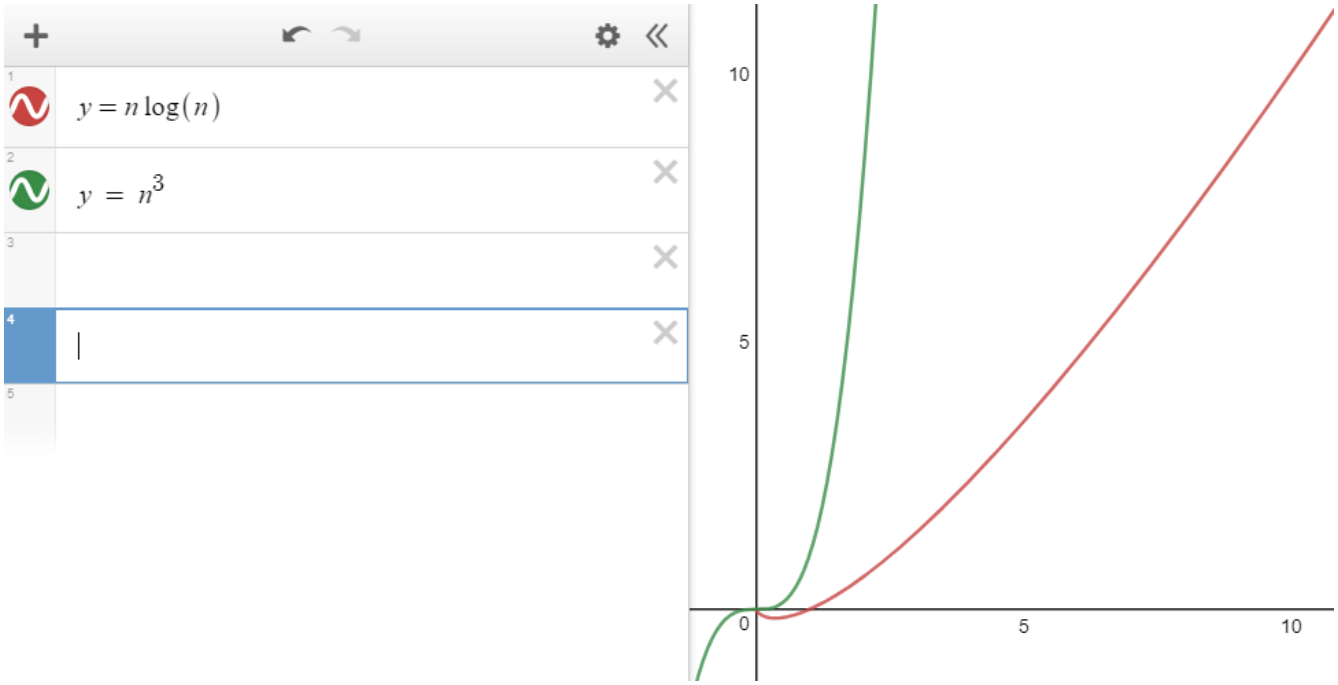
Finalmente comparamos la función lineal contra la función factorial, y es bastante claro que la función lineal es mucho más eficiente que la función factorial, siendo que el problema se resuelve mucho más eficientemente aún que el tamaño de problema sea muy grande

19) nlog(n) vs n^2



En esta comparación podemos observar que la función nlog(n) es más eficiente que la función cuadrática, aún que el tamaño de problema sea muy grande.

20) nlog(n) vs n^3

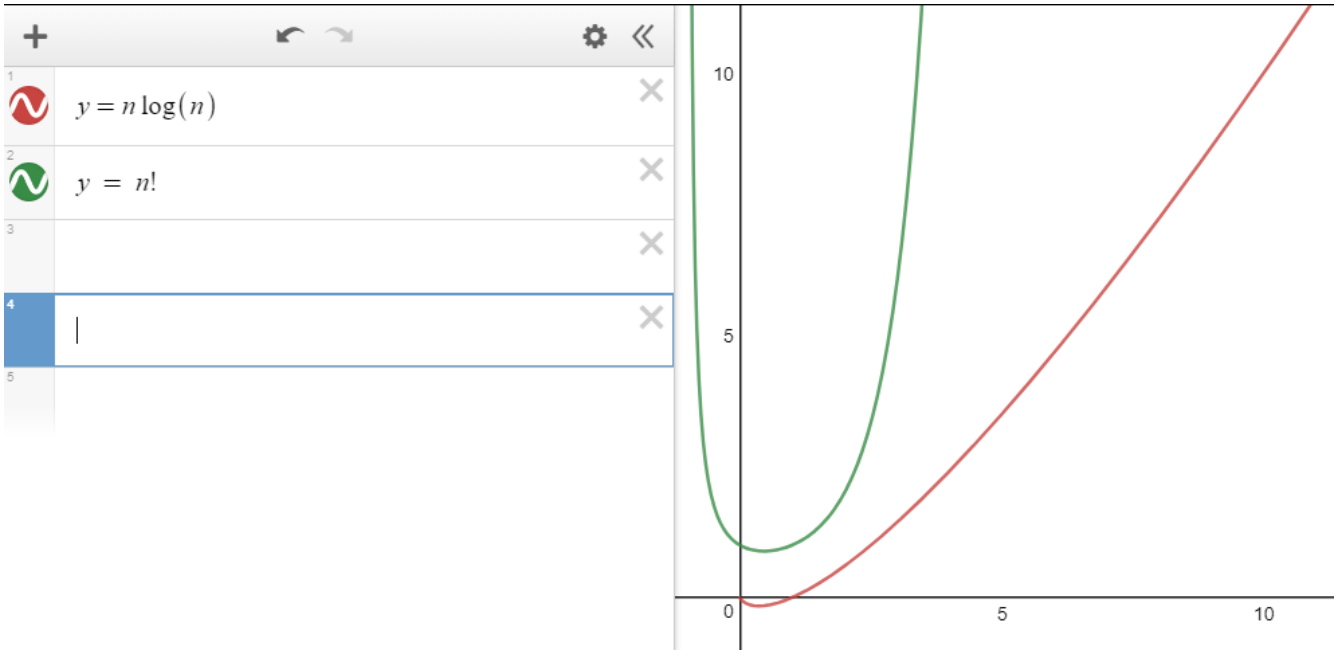


Nuevamente se repite lo mismo que con el caso anterior, pero la función cúbica crece aún más rápido que la función nlog(n) tanto que el introducir un tamaño de problema muy grande tardara demasiado en ser resuelto por este algoritmo.

21) nlog(n) vs c^n

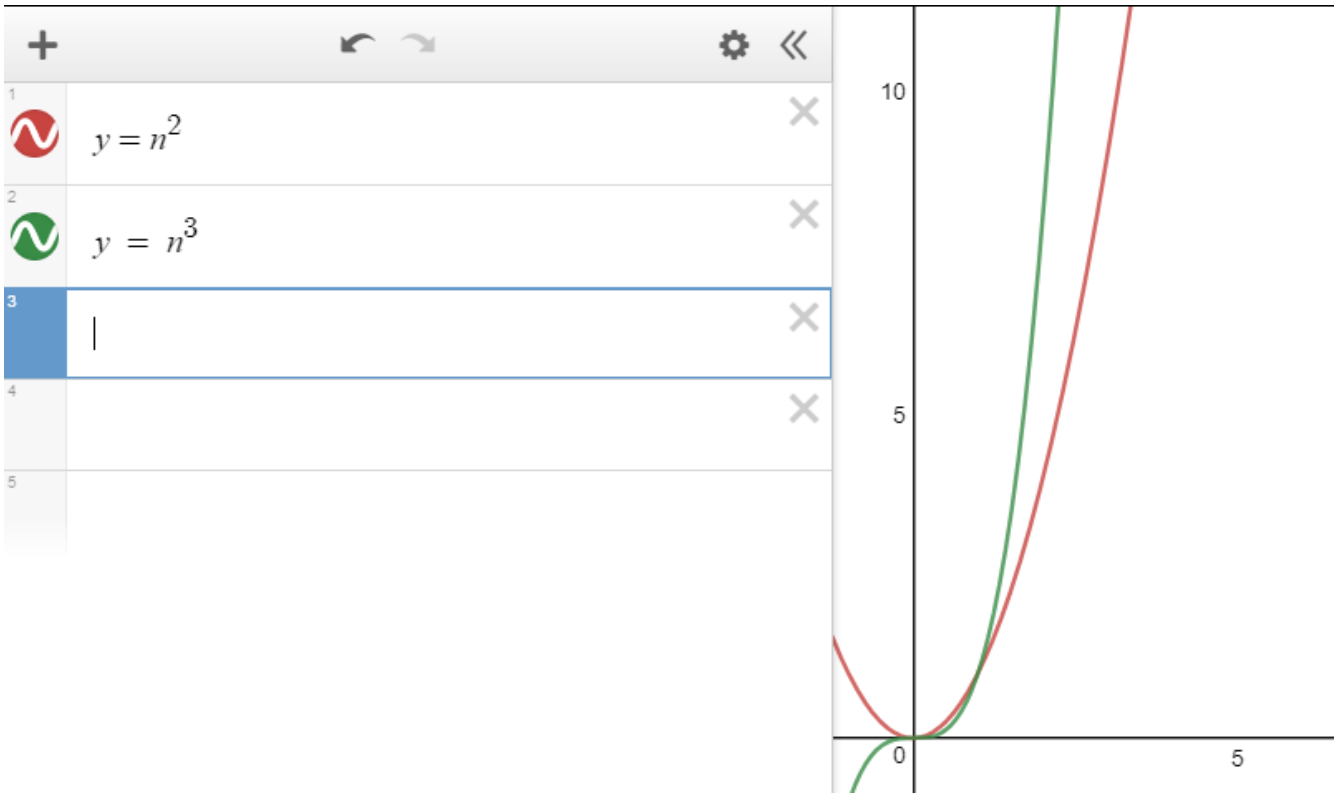
Nuevamente la función nlog(n) sale ganando contra la función exponencial, por bastante, siendo la función exponencial muy ineficiente si es comparada con la función nlog(n) tanto para tamaños de problema muy pequeños, como muy grandes.

22) nlog(n) vs n!



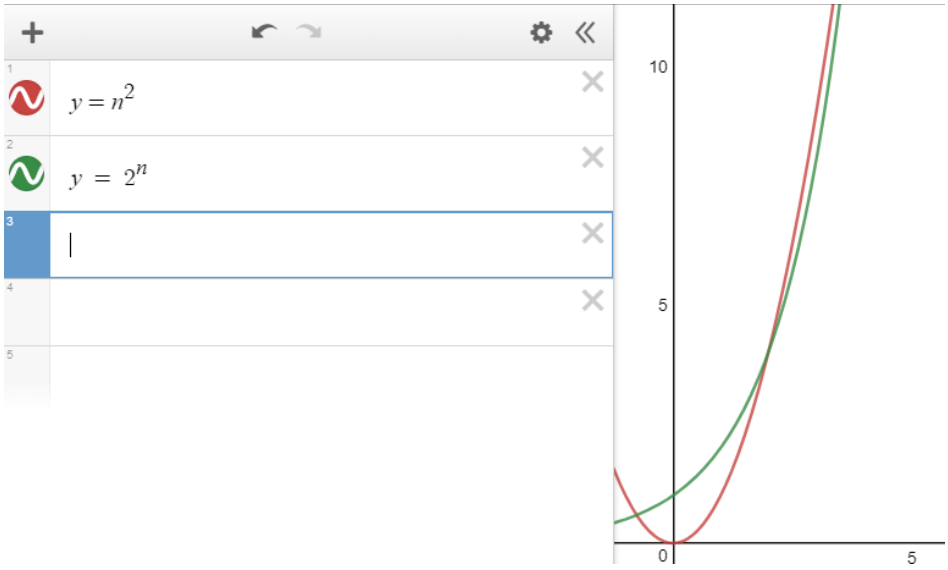
Finalmente, tenemos la comparación de nlog(n) y n! donde claramente la función nlog(n) es bastante mejor en eficiencia a diferencia de la función factorial, siendo excesivamente ineficiente la función factorial.

23) n^2 vs n^3



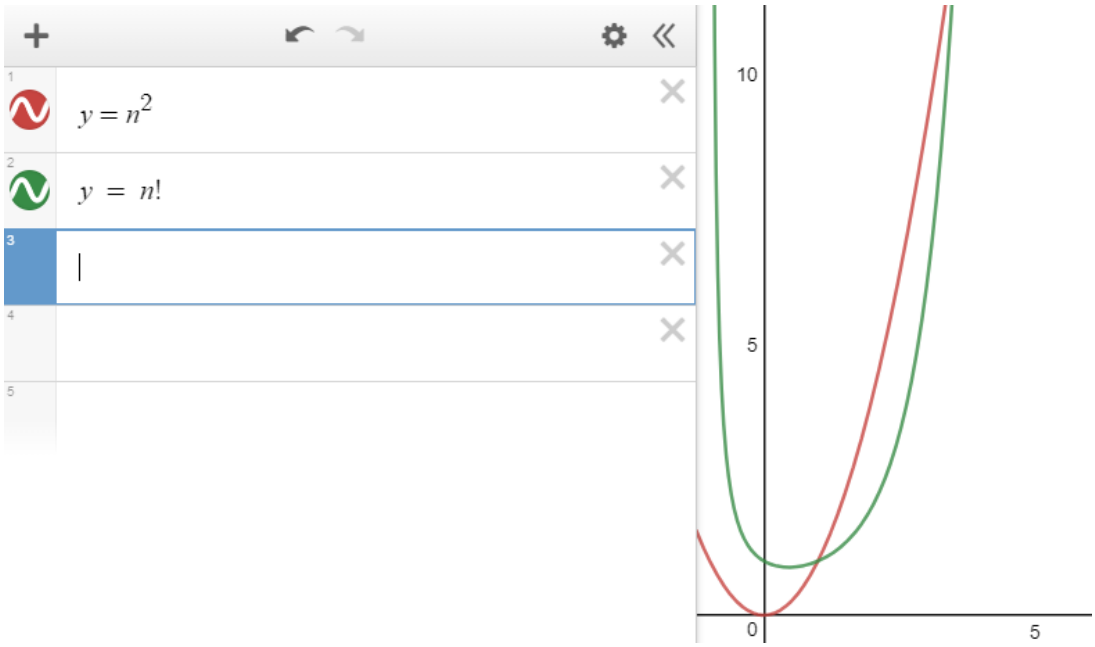
Comparando estas dos funciones podemos observar que es más eficiente un algoritmo con un orden de complejidad cuadrático a uno cúbico, la diferencia puede notarse con tamaños de problema muy grandes.

24) n^2 vs c^n



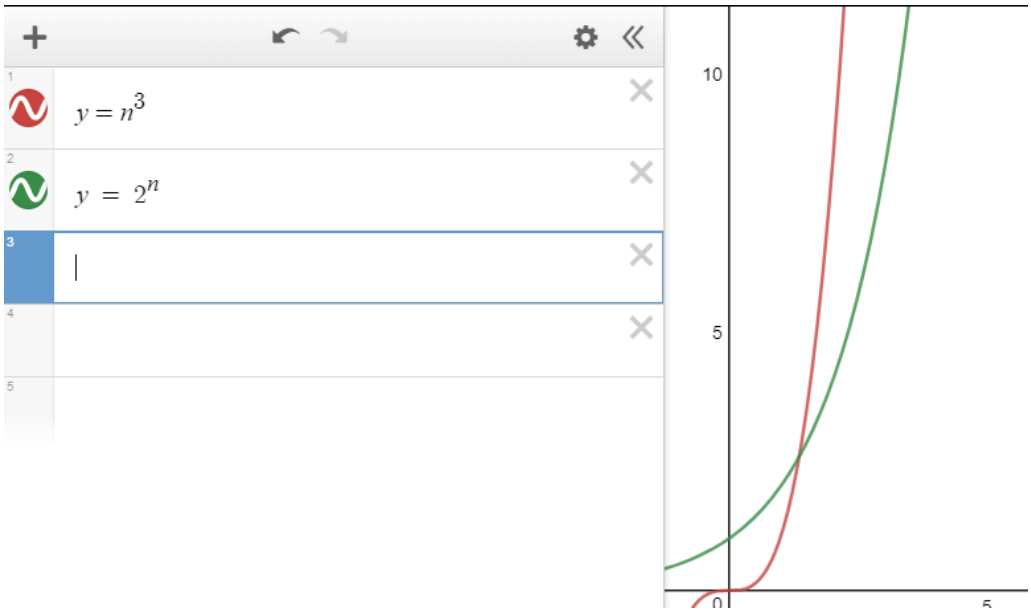
Si comparamos la función cuadrática contra la función exponencial, podemos observar que la función exponencial es muy ineficiente al lado de la función cuadrática, siendo que para tamaños de problema muy grandes, la función exponencial tarda demasiado en resolver un problema.

25) n^2 vs n!



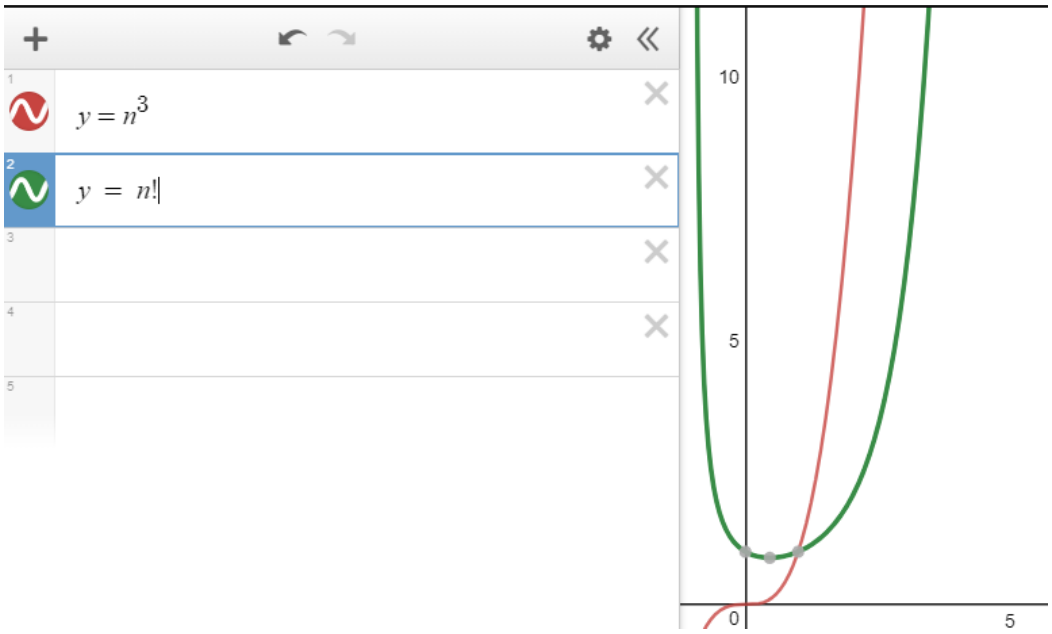
En esta comparación podemos observar que la función cuadrática es mucho más eficiente que la función complejidad factorial, siendo que para tamaños de problema muy grandes el crecimiento de la función logarítmica es bastante rápido.

26) n^3 vs c^n



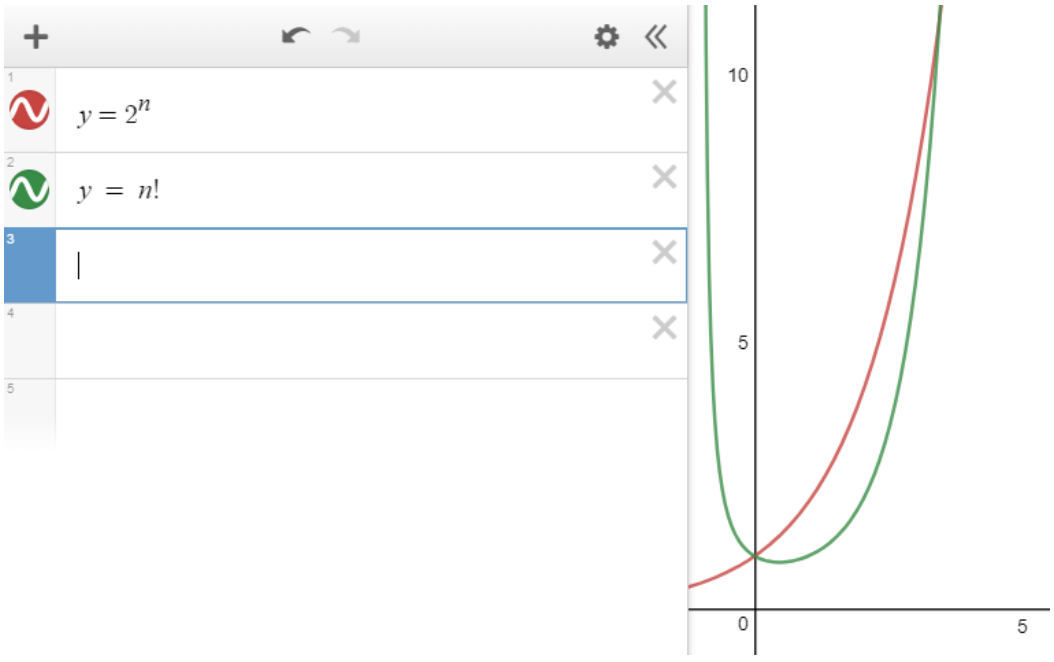
En este caso considero que ambas funciones son bastante ineficientes ya que ambas funciones de complejidad tienen un crecimiento bastante grande, pero como es evidente la función cubica es mucho más eficiente que la función exponencial tanto para tamaños de problema muy grandes como para tamaños de problema muy pequeños.

27) n^3 vs n!



Nuevamente si comparamos estas funciones podemos observar que la función cúbica es mucho más eficiente que la función factorial, sobre todo con tamaños de problema muy grandes.

28) c^n vs n!



En esta comparativa podemos observar que la función complejidad exponencial se comporta mejor que la función complejidad factorial, sobre todo con tamaños de problema muy grandes. Pero considero que ambos algoritmos son en exceso ineficientes.