

Universidad Nacional de La Matanza

Departamento de Ingeniería e Investigaciones Tecnológicas

Año 2019 - Segundo Cuatrimestre

Programación Avanzada

Trabajo Práctico N° 2

Complejidad Computacional

Fecha de entrega: 19/09/2019

**Integrantes:**

Arzola, Lucas| 39.166.800

Krasuk, Joaquín 40.745.090

Stanko, Diego 39.372.117

Tourn, Facundo 39.212.117

**Docentes:**

Dra. Verónica Aubin

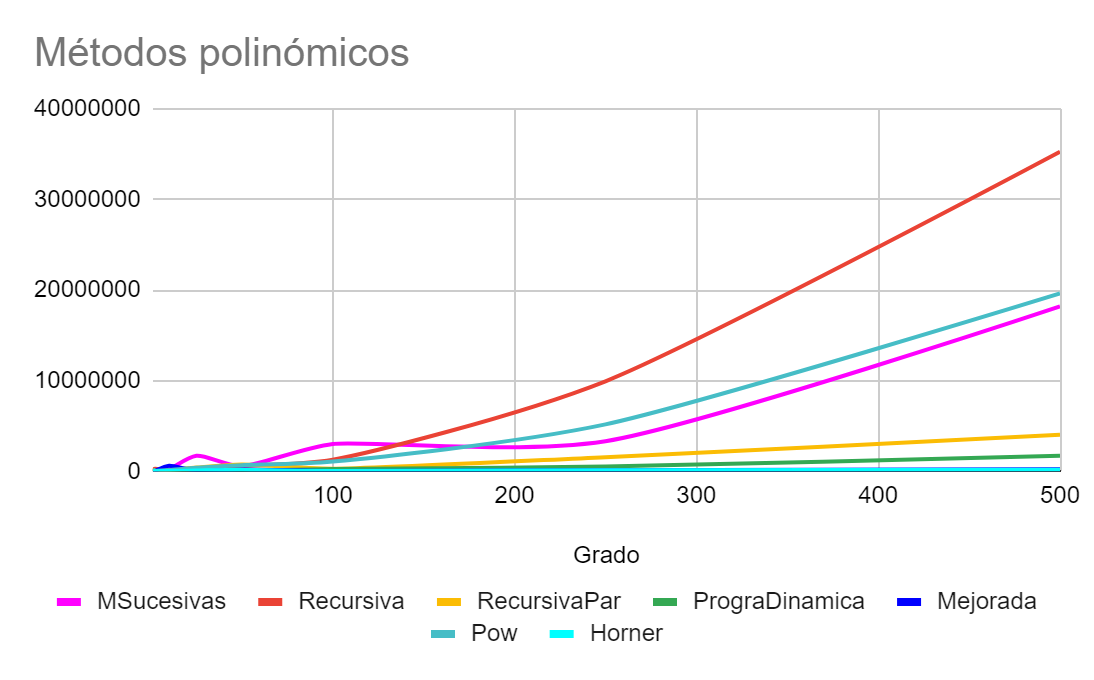
Ing. Lucas Videla

Ing. Lucas Ponce de León

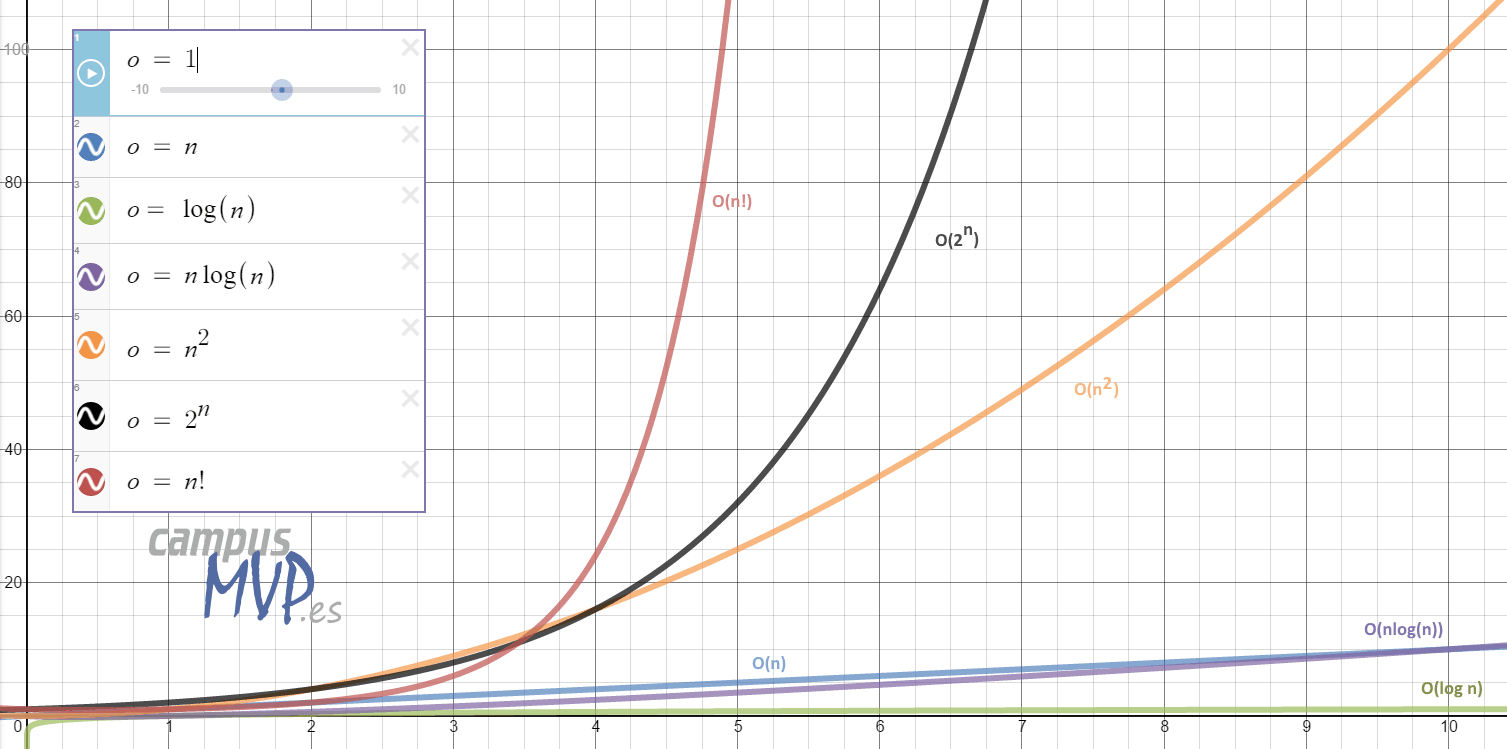
Ing. Hernán Lanzillota

**Conclusiones**:

Una vez hecho el desarrollo de los diferentes algoritmos solicitados y su posterior toma de mediciones de tiempo, llegamos a las siguientes conclusiones:



* Los algoritmos con complejidad O (n!) poseen la peor eficiencia, el tiempo de ejecución se ve afectado de forma significativa con pocas cantidades de operaciones, tales como por ejemplo el algoritmo recursivo “PotenciaRecursiva”.
* Luego de esto los algoritmos que poseen una complejidad O(n^2) son los próximos en mala eficiencia ya que el tiempo de ejecución crece de forma exponencial dependiendo de la cantidad de operaciones, como, por ejemplo “evaluarMSucesivas” y “EvaluarRecursividad”.
* Los algoritmos con complejidad O (n log n) y O (log n) son aquellos que tienen una mayor eficiencia con respecto de los anteriores algoritmos presentados. Como por ejemplo “evaluarRecursivaPar”, “evaluarProgDinamica” y “evaluarPow”

Este análisis tiene correspondencia con el siguiente gráfico en el que se compara los tiempos de ejecución de los algoritmos presentados.

* En el siguiente gráfico se puede observar como se ve afectado el tiempo en realizar una tarea dependiendo de la cantidad de datos a procesar dependiendo de cada algoritmo, esto nos permitirá saber cuál sería una complejidad adecuada para un algoritmo que estemos por desarrollar.

***Conclusión Final***

En conclusión, la complejidad computacional de cada algoritmo nos va a determinar que nos conviene seleccionar a la hora de codificarlos ya que es conveniente saber la complejidad computacional antes de codificarlos y hallarlos después, excluyendo por completo a algoritmos con complejidad computacional O(n!) para mayores de 10 elementos. Y seleccionando por amplia diferencia a algoritmo del tipo O(n\*log(n)) por sobre algoritmos de O(n^2).