# Trabajo práctico número 2

Complejidad Computacional – Programación Dinámica

Conclusiones

A partir de los diferentes métodos de cálculo en ambos ejercicios, polinomios y binomios de Newton, observamos una relación directa entre la complejidad computacional de los algoritmos versus su tiempo de ejecución.

Como vemos en la planilla de Excel asociada, entre menos complejidad computacional tiene un algoritmo tarda menos tiempo en realizar tareas que son similares.

En contra partida, vemos que algoritmos con la misma complejidad, aunque al fin y al cabo más eficientes, logran mejores tiempos de ejecución. Por ejemplo:

|  |  |  |
| --- | --- | --- |
| Método | Complejidad | Tiempo de ejecución |
| terminoQueOcupaElLugarKIterativo | O(N) | 3072 nanosegundos |
| terminoQueOcupaElLugarKRecursivo | O(N) | 1920 nanosegundos |

Como vemos estos dos métodos implementan la misma complejidad computacional pero el método iterativo supera con un margen importante al método recursivo.

Todos aquellos algoritmos que implementaron programación dinámica gozan de una mejor complejidad computacional y, en consecuencia, mejores tiempos de ejecución.

Ejemplo:

|  |  |  |
| --- | --- | --- |
| Método | Complejidad | Tiempo de ejecución |
| obtenerPolinomioIterativo | O(N^2) | 37247 nanosegundos |
| obtenerPolinomioRecursivo | O(N^2) | 617834 nanosegundos |
| obtenerPolinomioPow | O(N^2) | 36095 nanosegundos |
| **obtenerPolinomioDinamica** | **O(N\*LOG(N))** | **26495 nanosegundos** |

Para concluir podemos terminar diciendo que para lograr mejores tiempos de ejecución y mayor escalabilidad en los problemas que nuestro algoritmo es capaz de resolver lo mejor que podemos hacer es bajar la complejidad computacional a valores del estilo O(N), O(Log N) o también (N Log N).