Contenido

[Conclusiones Sobre Polinomios. 1](#_Toc493543103)

[Introducción 1](#_Toc493543104)

[Pruebas Realizadas 1](#_Toc493543105)

[Codificación 1](#_Toc493543106)

[Complejidad Computacional 2](#_Toc493543107)

[Tiempos de ejecución 2](#_Toc493543108)

[Observaciones finales 3](#_Toc493543109)

# Conclusiones Sobre Polinomios.

## Introducción

Ante la necesidad de poder evaluar polinomios de grado n, se han realizado diferentes métodos con el fin de poder obtener el mismo resultado con diferentes implementaciones. En dichas implementaciones, se tiene cuenta que la complejidad computacional varia, siendo algunos mas beneficiosos que otros si se desean resolver polinomios de grado muy grande. En este documento se podrá observar la evaluación de cada método, las pruebas realizadas para cada uno, y el tiempo de respuesta que tuvo.

## Pruebas Realizadas

Se realiza un Lote de Prueba el cual es utilizado para todos los métodos. Las pruebas se llevan a cabo mediante la utilización de JUnit 4.0.



De las pruebas realizadas, se contabilizo el tiempo que tarda en resolver los polinomios de grado 20, 50 y 100 a fin de lograr una comparación.

## 

## Codificación

Se llevó a cabo la creación de los siguientes métodos.

doubleevaluarMSucesivas(double x ) { . . . }

doubleevaluarRecursiva(double x ) { . . . }

doubleevaluarRecursiva P a r (double x ) { . . . }

doubleevaluarProgDinamica (double x ) { . . . }

doubleevaluarMejorada (double x ) { . . . }

doubleevaluarPow (double x ) { . . . }

doubleevaluarHorner( double x ) { . . . }

Cada uno cuenta con diferentes formas de resolver el problema. Para cada uno se llevó a cabo los correspondientes test a fin de evaluar su correcto funcionamiento y visualizar tiempos.

## Desarrollo de las pruebas.

Se llevo a cabo el monitoreo de las pruebas, utilizando el siguiente grafico como muestra.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | |
| **Polinomios** | | | | | | | |
|  | **Msucesiva** | **Recursiva** | **RecursivaPar** | **Horner** | **ProgDinam** | **Mejorado** | **Pow** |
| **Grado 0** | OK | OK | OK | OK | OK | OK | OK |
| **Grado 1** | OK | OK | OK | OK | OK | OK | OK |
| **Grado 2** | OK | OK | OK | OK | OK | OK | OK |
| **Grado 20** | OK | OK | OK | OK | OK | OK | OK |
| **Grado 50** | OK | OK | OK | OK | OK | OK | OK |
| **Grado 100** | OK | OK | OK | OK | OK | OK | OK |
| **Numero Con Coma** | OK | OK | OK | OK | OK | OK | OK |

## 

## Complejidad Computacional

El siguiente cuadro muestra como son las complejidades computacionales final (incluyendo la resolución y el cálculo de las potencias).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Msucesiva | Recursiva | RecursivaPar | Horner | ProgDinam | Mejorado | Pow |
| O | n2 | n2 | 2n | n | n | n | n2 |

## Tiempos de ejecución

Para el siguiente cálculo se utiliza una función que te devuelve el tiempo en nano segundos. De esta manera, se pudo apreciar como difiere con un buen margen los tiempos de ejecución. Los resultados son los siguientes.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Grado / Algoritmo | **Msucesiva** | **Recursiva** | **RecursivaPar** | **Horner** | **ProgDinam** | **Mejorado** | **Pow** |
| **Grado 20** | 25263 | 34341 | 70132 | 20921 | 19341 | 17763 | 46973 |
| **Grado 50** | 35526 | 104208 | 95791 | 6710 | 5132 | 5921 | 32368 |
| **Grado 100** | 105392 | 48157 | 595270 | 20131 | 19737 | 16184 | 29605 |

Con estos valores se genera un gráfico para una mejor visualización

De esta manera, se puede apreciar como la Recursiva par crece exponencialmente, asi como también se observa que tanto las funciones Horner, Programación dinámica y Mejorado crecen a otro ritmo.

## Observaciones finales

* Las funciones realizadas con el método de Horner, asi como también la de Programación dinámica y Mejorada tienen una complejidad O(n), con lo cual son los métodos más optimizados.
* La función recursiva Par se puede mejorar utilizando Programación dinámica. Se podría utilizar un algoritmo que vaya guardando las potencias calculadas para luego utilizarlas sin tener que calcular cada vez que se busque un valor.
* La función mejorada se realiza utilizando un atributo para almacenar el valor por el cual se debe multiplicar los coeficientes, similar a lo utilizado en Programación dinámica.

# Conclusiones Sobre Binomios.

## Introducción

Para la evaluación de binomios, se utilizan las mismas formas de resolver potencias que en los polinomios. También se realiza el algoritmo que retorna los coeficientes correspondientes al grado n, haciendo uso del triángulo de Tartaglia.

## Pruebas Realizadas

Se llevaron a cabo pruebas tanto con el desarrollo del binomio, armando primero el polinomio, asi como también resolviendo de forma directa el binomio. De esta manera, se pudo encontrar una diferencia muy grande entre los tiempos de ejecución de uno y de otro. Las pruebas realizadas fueron las siguientes:



## Complejidad Computacional

La complejidad se muestra en la siguiente tabla

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Msucesiva | Recursiva | RecursivaPar | Horner | ProgDinam | Mejorado | Pow |
| O | n2 | n2 | n2 | n2 | n | n/2 | n2 |

## Tiempos de ejecución

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Grado / Algoritmo | **Msucesiva** | **Recursiva** | **RecursivaPar** | **Horner** | **ProgDinam** | **Mejorado** | **Pow** |
| **Grado 20** | 2444 | 7331 | 11869 | 51096 | 5249 | 4619 | 2094 |
| **Grado 50** | 2444 | 20946 | 29324 | 652688 | 17148 | 5132 | 1746 |
| **Grado 100** | 12567 | 504787 | 352932 | 1028901 | 43745 | 21553 | 15360 |

Se puede observar que la resolución realiza con el desarrollo del polinomio es muchísimo más lenta que las que se llevan a cabo en las resoluciones directas.

## Observaciones finales

* Las funciones realizadas con programación dinámica tiene una complejidad O(n), con lo cual es el método más optimizado.
* La función recursiva Par se puede mejorar utilizando Programación dinámica. Se podría utilizar un algoritmo que vaya guardando las potencias calculadas para luego utilizarlas sin tener que calcular cada vez que se busque un valor.