**Universidad Nacional de La Matanza**

Depto. de Ingeniería e Investigaciones Tecnológicas



**PROGRAMACIÓN**

**AVANZADA**

TRABAJO PRÁCTICO 3

Complejidad Computacional

Integrantes:

|  |  |  |
| --- | --- | --- |
| APELLIDO, Nombre | E-mail | DNI |
| AMORUSO, Sergio Federico | [sergioamoruso91@gmail.com](mailto:sergioamoruso91@gmail.com) | 36593815 |
| BRUDE, Alejandro Gabriel | [alejandrobrude@gmail.com](mailto:alejandrobrude@gmail.com) | 33908097 |
| CABRAL, Rodrigo Ariel | [cabralrodrigoariel@gmail.com](mailto:cabralrodrigoariel@gmail.com) | 35103809 |
| TOMALINO, Carlos Alberto | [c.ctomalino@gmail.com](mailto:c.ctomalino@gmail.com) | 32942556 |

PRIMER CUATRIMESTRE – AÑO 2017

**POLINOMIO: LOTE DE PRUEBAS**

A continuación, se muestran los casos de prueba realizados sobre las distintas versiones del programa que evalúa polinomios por medio de múltiples métodos. En cada caso se presenta en primer lugar la descripción, en segundo lugar, la entrada, seguido de la salida que se espera en base a dicha entrada, y por último qué fue lo que se obtuvo al ejecutar cada método.

**01\_RaizNula**

Se evalúa el comportamiento del programa cuando la raíz es nula.

|  |
| --- |
| **Entrada** |
| 2 |
| 0 |
| 1 |
| -3 |
| 1 |

|  |
| --- |
| **Salida Esperada** |
| 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Método** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **Salida** | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **T (ns)** | 168616 | 124308 | 124308 | 115692 | 117333 | 189539 | 114872 |

**02\_RaizNegativa**

Se evalúa el comportamiento del programa cuando la raíz es negativa.

|  |
| --- |
| **Entrada** |
| 3 |
| -1 |
| 25 |
| -3 |
| 60 |
| 13 |

|  |
| --- |
| **Salida Esperada** |
| -75 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Método** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **Salida** | -75 | -75 | -75 | -75 | -75 | -75 | -75 |
| **T (ns)** | 82051 | 31589 | 31590 | 37744 | 45949 | 91077 | 46359 |

**03\_GradoDos**

Se evalúa el comportamiento del programa cuando el polinomio es de segundo grado.

|  |
| --- |
| **Entrada** |
| 2 |
| 2 |
| 1 |
| -4 |
| 4 |

|  |
| --- |
| **Salida Esperada** |
| 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Método** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **Salida** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **T (ns)** | 50872 | 37743 | 36513 | 22564 | 26257 | 62359 | 22974 |

**04\_GradoTres**

Se evalúa el comportamiento del programa cuando el polinomio es de tercer grado.

|  |
| --- |
| **Entrada** |
| 3 |
| 2 |
| 1 |
| -6 |
| 12 |
| -8 |

|  |
| --- |
| **Salida Esperada** |
| 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Método** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **Salida** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **T (ns)** | 95589 | 55385 | 71385 | 37744 | 34872 | 87795 | 34462 |

**05\_RaizIrracional**

Se evalúa el comportamiento del programa cuando la raíz es un número irracional.

|  |
| --- |
| **Entrada** |
| 3 |
| 1.41 |
| 1 |
| -3 |
| 1 |
| 7 |

|  |
| --- |
| **Salida Esperada** |
| 5.248921 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Método** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **Salida** | 5.2489209per | 5.2489209per | 5.2489209per | 5.2489209per | 5.2489209per | 5.248921 | 5.248921 |
| **T (ns)** | 73026 | 48821 | 38154 | 43487 | 35693 | 80411 | 36103 |

**\*** per = periódico

**06\_Fatiga**

Se evalúa el comportamiento del programa cuando el polinomio es de grado 99.

|  |
| --- |
| **Entrada** |
| 99 |
| 6 |
| … |

|  |
| --- |
| **Salida Esperada** |
| Desconocida |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **M** | **MSucesivas** | **Recursiva** | **RecursivaPar** | **ProgDinamica** | **Mejorada** | **Pow** | **Horner** |
| **S** | 4.5627090219052543E77 | 4.5627090219052543E77 | 4.562709021905254E77 | 4.5627090219052543E77 | 4.5627090219052553E77 | 4.5627090219052543E77 | 4.562709021905257E77 |
| **T** | 4661341 | 916924 | 192001 | 125539 | 126769 | 182975 | 190359 |

**POLINOMIO: ANÁLISIS DE COMPLEJIDAD COMPUTACIONAL**

|  |  |
| --- | --- |
| **MÉTODO** | **BIG O** |
| **MSucesivas** | **n \* (n-1) = n^2 – n**  n: cantidad de loops para recorrer vector polinomio.  (n-1): cantidad de loops para calcular el valor de la potencia de x. |
| **Recursiva** | **n \* (n-1) = n^2 – n**  n: cantidad de loops para recorrer vector polinomio.  (n-1): cantidad de loops para calcular el valor de la potencia de x. |
| **RecursivaPar** | **n \* [ ½ \* log2(n-1) + ½ \* (n-1) ]**  n: cantidad de loops para recorrer el vector polinomio  ½ \* log2(n-1) + ½ \* (n-1): la mitad de las veces que se ejecuta este método recursivo se utilizará recursiva par, que envía como parámetro la mitad de la potencia (con O(log2(n-1))) y la otra mitad se utilizará recursiva (con O(n-1)). |
| **ProgDinamica** | **n**  Hay 2 loops, uno para cargar el vector de potencias, con O(n-1) y otro para recorrer el vector polinomio, con O(n). Por regla de la suma queda la segunda. |
| **Mejorada** | **n**  Es igual que el método anterior, sólo que realiza ambos cálculos en un mismo loop, por lo tanto la complejidad computacional es la misma. |
| **Pow** | **n \* O(pow) = n \* O(1) = n**  La complejidad computacional del método “pow” perteneciente al paquete de clases estándar “Math” de java es de O(1). |
| **Horner** | **n**  En este caso, sólo se utiliza recursividad, pero la cantidad de llamados al método recursivo depende de la cantidad n de elementos del polinomio. |

**POLINOMIO: GRÁFICOS Y TABLAS DE RENDIMIENTO COMPARATIVO**

**MSucesivas:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 50872 |
| 3 | 70909 |
| 4 | 92368 |
| 5 | 100765 |
| 20 | 74330 |
| 50 | 122535 |
| 99 | 399329 |

**Recursiva:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 50382 |
| 3 | 37009 |
| 4 | 53804 |
| 5 | 72153 |
| 20 | 59401 |
| 50 | 174473 |
| 99 | 453445 |

**Recursiva Par:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 32655 |
| 3 | 37009 |
| 4 | 54115 |
| 5 | 58780 |
| 20 | 61890 |
| 50 | 128444 |
| 99 | 349258 |

**ProgDinamica:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 46961 |
| 3 | 47273 |
| 4 | 51004 |
| 5 | 64999 |
| 20 | 47895 |
| 50 | 53493 |
| 99 | 136841 |

**Mejorada:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 48206 |
| 3 | 46340 |
| 4 | 49761 |
| 5 | 51316 |
| 20 | 37321 |
| 50 | 61268 |
| 99 | 153014 |

**Pow:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 125957 |
| 3 | 102942 |
| 4 | 120359 |
| 5 | 105741 |
| 20 | 180072 |
| 50 | 124401 |
| 99 | 206506 |

**Horner:**

|  |  |
| --- | --- |
| **Polinomio de grado** | **tiempo de ejecución en Ns** |
| 2 | 52871 |
| 3 | 54426 |
| 4 | 51004 |
| 5 | 62201 |
| 20 | 58158 |
| 50 | 53493 |
| 99 | 132177 |

**POLINOMIO: CONCLUSIONES**

Con la información obtenida, se llegó a las siguientes conclusiones:

* El método **MSucesivas**, en términos generales, es el menos eficiente, ya que, salvo algún caso excepcional, su tiempo de ejecución superó ampliamente el de los demás.
* La diferencia entre los tiempos de ejecución de los métodos **recursiva** y **recursivaPar** no es muy grande a excepción de cuando se desea procesar un gran número de datos de entrada. Es en esta situación cuando conviene utilizar el segundo ya que el tiempo de ejecución es reducido sustancialmente.
* En cuanto a los métodos **ProgDinamica** y **Mejorada**, la diferencia entre sus tiempos de ejecución es mínima y se mantiene constante en todos los casos de prueba realizados, por lo tanto se determina que es indistinto utilizar uno u otro a la hora de resolver este problema. Además, conforman una opción más conveniente que los métodos recursivos anteriores por demostrar un tiempo de ejecución ligeramente menor.
* El método **Pow** cuenta con la ventaja de que utiliza la función estándar Math.pow(), motivo por el cuál sirve para ahorrar tiempo de codificación y se comporta muy bien con polinomios de grado n muy alto, pero no así con polinomios de grado n chico.
* Finalmente, el método **Horner**, que utiliza el algoritmo de Horner, es el más adecuado para resolver este problema, ya que en todos los casos, su tiempo de ejecución se mantuvo en los niveles más bajos.