**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**

**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 |

**BINOMIO DE NEWTON: LOTE DE PRUEBAS**

**07\_BinomioTrivial**

|  |
| --- |
| **Entrada** |
| 2 |
| 3 |
| 3 |

|  |
| --- |
| **Salida Esperada** |
| x^3\*8.0+x^2\*36.0+x^1\*54.0+x^0\*27.0 |

|  |  |  |
| --- | --- | --- |
| **Método** | **calcularBinomioCompleto** | **calcularBinomioCompletoOptimizado** |
| **Salida** | x^3\*8.0 + x^2\*36.0 + x^1\*54.0 + x^0\*27.0 | x^3\*8.0 + x^2\*36.0 + x^1\*54.0 + x^0\*27.0 |
| **T (ns)** | 75077 | 66461 |

**08\_BinomioExpCero**

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

|  |
| --- |
| **Salida Esperada** |
| x^0\*1.0 |

|  |  |  |
| --- | --- | --- |
| **Método** | **calcularBinomioCompleto** | **calcularBinomioCompletoOptimizado** |
| **Salida** | x^0\*1.0 | x^0\*1.0 |
| **T (ns)** | 68923 | 62359 |

**09\_BinomioExpUno**

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

|  |
| --- |
| **Salida Esperada** |
| x^1\*2.0 + x^0\*3.0 |

|  |  |  |
| --- | --- | --- |
| **Método** | **calcularBinomioCompleto** | **calcularBinomioCompletoOptimizado** |
| **Salida** | x^1\*2.0 + x^0\*3.0 | x^1\*2.0 + x^0\*3.0 |
| **T (ns)** | 82461 | 57026 |

**10\_BinomioFatiga**

|  |
| --- |
| **Entrada** |
| 2 |
| 3 |
| 30 |

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

|  |  |  |
| --- | --- | --- |
| **Método** | **calcularBinomioCompleto** | **calcularBinomioCompletoOptimizado** |
| **Salida** | x^30\*1.073741824E9 + x^29\*-1.610612736E9 + x^28\*0.0 + x^27\*1.4495514624E10 + x^26\*0.0 + x^25\*3.42456532992E11 + x^24\*-4.8922361856E10 + x^23\*9.172942848E10 + x^22\*-8.2556485632E10 + x^21\*4.1278242816E10 + x^20\*-6.1917364224E10 + x^19\*9.2876046336E10 + x^18\*1.39314069504E11 + x^17\*-2.08971104256E11 + x^16\*-6.26913312768E11 + x^15\*-1.880739938304E12 + x^14\*-1.410554953728E12 + x^13\*-1.057916215296E12 + x^12\*1.586874322944E12 + x^11\*2.380311484416E12 + x^10\*-3.570467226624E12 + x^9\*5.355700839936E12 + x^8\*-2.4100653779712E13 + x^7\*6.025163444928E13 + x^6\*-7.2301961339136E13 + x^5\*1.138755891091392E15 + x^4\*0.0 + x^3\*2.44019119519584E14 + x^2\*0.0 + x^1\*-1.37260754729766E14 + x^0\*2.05891132094649E14 | x^30\*1.073741824E9 + x^29\*4.831838208E10 + x^28\*1.05092481024E12 + x^27\*1.471294734336E13 + x^26\*1.4896859185152E14 + x^25\*1.161955016441856E15 + x^24\*7.2622188527616E15 + x^23\*3.73485540999168E16 + x^22\*1.610656395558912E17 + x^21\*5.905740117049344E17 + x^20\*1.86030813687054336E18 + x^19\*5.0735676460105728E18 + x^18\*1.204972315927511E19 + x^17\*2.502634810003292E19 + x^16\*4.558370546791711E19 + x^15\*7.293392874866737E19 + x^14\*1.0256333730281349E20 + x^13\*1.2669588725641667E20 + x^12\*1.3725387786111805E20 + x^11\*1.3002998955263816E20 + x^10\*1.0727474138092647E20 + x^9\*7.662481527209034E19 + x^8\*4.701977300787362E19 + x^7\*2.4532055482368844E19 + x^6\*1.0732774273536369E19 + x^5\*3.8637987384730931E18 + x^4\*1.11455732840569997E18 + x^3\*2.4767940631237776E17 + x^2\*3.9805618871632144E16 + x^1\*4.11782264189298E15 + x^0\*2.05891132094649E14 |
| **T (ns)** | 356102 | 350769 |

**POLINOMIO: ANÁLISIS DE COMPLEJIDAD COMPUTACIONAL**

|  |  |
| --- | --- |
| **MÉTODO** | **BIG O** |
| **MSucesivas** | **n \* (n-1) -----> n^2**  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: cantidad de loops para recorrer vector polinomio.  (n-1): cantidad de loops para calcular el valor de la potencia de x. |
| **RecursivaPar** | **n\* log2(n)**  n: cantidad de loops para recorrer el vector polinomio  log2(n): este método envía como parámetro la mitad de la potencia. |
| **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(Math.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. |

**BINOMIO DE NEWTON: ANÁLISIS DE COMPLEJIDAD COMPUTACIONAL**

|  |  |
| --- | --- |
| **MÉTODO** | **BIG O** |
| **obtenerTermino** | **n**  Para calcularlo, necesito saber primero la complejidad computacional del método combinatoria, que se compone de la siguiente manera:  **private** **int** combinatoria(**int** n, **int** k) {  **return** factorial(n) / (factorial(k) \* factorial(n - k));  }  La CC de **factorial** es **n**, entonces queda **n/k\*(n-k)**.  **n** dividido un número muy grande da como resultado un **n** más chico, pero se toma el peor caso (que el denominador no sea muy grande) quedando **n**.  Luego además del llamado al método **combinatoria** hay 2 llamados al método Math.pow(), cuya CC como se mencionó anteriormente es **1**. Queda entonces que la complejidad computacional del método **obtenerTermino** es **n** (siendo **n** el grado del polinomio desarrollado o el exponente del binomio que se encuentra en el archivo de entrada). |
| **calcularBinomioCompleto** | **n^2**  En este caso, tenemos un llamado al método **combinatoria** dentro de un **for** que va desde cero hasta el exponente. Por lo tanto queda O(combinatoria) \* O(n) = O(n) \* O(n) = n^2. |
| **calcularBinomioCompletoOptimizado** | **n**  En este caso tenemos primero un **for** que va desde 1 hasta el valor del exponente y luego otro **for** que va desde 0 hasta el exponente. No hay ningún llamado a otro método dentro de ninguno de los dos. Por regla de la suma se toma el peor, que es el segundo, quedando **n**. |
| **calcularXPolinomioCompleto** | **n**  Dentro de dicho método hay un **for** que decrementa la variable k desde el valor del exponente hasta 0, y dentro de ese for hay varios llamados al método Math.pow() cuya CC, como se mencionó anteriormente, es constante, por lo tanto se puede afirmar que la CC de este método es **n**. |

**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 |

**BINOMIO DE NEWTON: GRÁFICOS Y TABLAS DE RENDIMIENTO COMPARATIVO**

**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 esa situación 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.

**BINOMIO DE NEWTON: CONCLUSIONES**

Según lo analizado, los métodos correspondientes a la clase **BinomioDeNewton** son medianamente óptimos ya que su complejidad computacional es de O(n), a excepción del método **calcularBinomioCompleto**, cuya CC asciende a **n^2**. Sin embargo, el método **calcularBinomioCompletoOptimizado** logra solucionar este inconveniente, por medio de la utilización del triángulo de Tartaglia.