**ANALISIS DE COMPLEJIDAD METODO DE NEWTON**

Teniendo en cuenta que math.Pow es O(1) re realizó el siguiente análisis de complejidad

|  |  |  |
| --- | --- | --- |
| **public** **double** realEvaluation(**double** number){  **double** value = 0;  **for** (**int**i=0;i<constants.length;i++) { value += constants[i] \* Math.*pow*(number,i);}  **return** value;} |  | # |
| C1 | 1 |
| C2 | n |
| C3 | n-1 |
| C4 | 1 |

**T(n) = 1+n+n-1+1 = 2n+1 O(n)**

|  |  |  |
| --- | --- | --- |
| **public** ComplexNumber complexEvaluation(ComplexNumber number){ |  | # |
| ComplexNumber value = **new** CNumber(0,0);  **for** (**int** i = 0; i < constants.length;i++){  value.sum(number.pow(i).multiplication(constants[i]));  }**return** value;} | C1 | 1 |
| C2 | n+1 |
| C3 | n |
| C4 | 1 |

**T(n) = 1+n+1+n+1= 2n+3 O(n)**

|  |  |  |
| --- | --- | --- |
| **public** ComplexNumber complexDerivatedEvaluation(ComplexNumber number){ |  | # |
| ComplexNumber value = **new** CNumber(0,0);  **for** (**int** i = 1; i < constants.length;i++){  value.sum(number.pow(i-1).multiplication(constants[i]\*i));}  **return** value; } | C1 | 1 |
| C2 | n |
| C3 | n-1 |
| C4 | 1 |

**T(n) = 1+n+n-1+1= 2n+1 O(n)**

|  |  |  |
| --- | --- | --- |
| **public** **double** RealDerivatedEvaluation(**double** number){ |  | # |
| **double** value = 0;  **for** (**int** i = 1; i < constants.length;i++){  value += (Math.*pow*(number,i-1) \* constants[i]\*i);}  **return** value;} | C1 | 1 |
| C2 | n |
| C3 | n-1 |
| C4 | 1 |

**T(n) = 1+n+n-1+1 = 2n+1 O(n)**

|  |  |  |
| --- | --- | --- |
| **public** ComplexNumber newtonComplexMethod(){ |  | # |
| ComplexNumber root = **new** CNumber(Math.*random*()\*10000,Math.*random*()\*10000);  **for** (**int** i = 0; i < 70; i++) {  root = root.subtraction(complexEvaluation(root).divide(complexDerivatedEvaluation(root)));}  **return** root;} | C1 | 1 |
| C2 | 1 |
| C3 | n+1 |
| C4 | 1 |

**T(n) = 1+1+n+1+1= n+4 + 2n+1 + 2n+1 = n+4 + 2(2n+1) O(n)**

|  |  |  |
| --- | --- | --- |
| **public** String toString() { |  | # |
| String polynomial = "";  **for** (**int** i = grade; i > 0; i--) {  **if**(constants[i] != 0)  polynomial += (Math.*abs*(constants[i]) != 1?(Math.*floor*(constants[i]\*1000)/1000):"") + " X^" + i + " " + (constants[i-1]<=0? (constants[i-1] == -1?" - ":""):" + ");  }  polynomial += (Math.*floor*(constants[0]\*1000)/1000);  **return** polynomial;} | C1 | 1 |
| C2 | n+1 |
| C3 | n |
| C4 | n-1 |
| C5 | 1 |

**T(n) = 1+n+1+n+n-1+1 = 3n+2 O(n)**

**ANALISIS DE COMPLEJIDAD METODO DIVISION SINTETICA O RUFFINI**

|  |  |  |
| --- | --- | --- |
| **public** String getRoots(){ |  | **#** |
| String roots = "";  **int**[] independent = exactDivisors(constants[0]);  **int**[] majorTerm = exactDivisors(constants[constants.length-1]);  **for** (**int** i = 0; i < independent.length; i++) {  **for** (**int** j = 0; j < majorTerm.length; j++) {  **if** (sinteticDivision((**double**)independent[i]/(**double**)majorTerm[j]) < 0.00000000001){  roots += "x = " + independent[i] + "/" + majorTerm[j]+"\n";  }}}  **return** roots;} | C1 | 1 |
| C2 | 1 |
| C3 | 1 |
| C4 | n+1 |
| C5 | n^2+2n+1 |
| C6 |  |
| C7 |  |
| C8 | 1 |

|  |  |  |
| --- | --- | --- |
| **private** **double** sinteticDivision(**double** num){ |  |  |
| **double** temp = constants[constants.length-1];  **for** (**int** i = constants.length-2; i >= 0; i--) {  temp = num \* temp + constants[i];}  **return** temp;} | C1 | 1 |
| C2 | n-1 |
| C3 | n |
| C4 | 1 |

**T(n) = 1+n-1+n+1 = 2n+1 O(n)**

|  |  |  |
| --- | --- | --- |
| **public** **int**[] exactDivisors(**double** num){ |  |  |
| String divisors = "";  **for** (**int** i = -((**int**)num+1); i < num+1; i++) {  **if** (num % i == 0){  divisors +=i;  **if** (i<num){  divisors+=",";}}}  String[] divisorsCut = divisors.split(",");  **int**[] divisorsReturn = **new** **int**[divisorsCut.length];  **for** (**int** i = 0; i < divisorsCut.length; i++) {  divisorsReturn[i] = Integer.*parseInt*(divisorsCut[i]);}  **return** divisorsReturn; | C1 | 1 |
| C2 | 2n+1 |
| C3 | 2n |
| C4 |  |
| C5 |  |
| C6 |  |
| C7 |  |
| C8 |  |
| C9 |  |
| C10 |  |

**T(n) =**

COMPLEJIDAD NEWTHON RAPHSON

Z es el número máximo de iteraciones para hallar la raíz.











