**Trabalho 1 - Experimentos em Complexidade**

**4645G-04 - Algoritmos e Estruturas de Dados I - Turma 10**

**Nome**: Gustavo Lösch do Amaral

**Curso**: Ciencia de Dados e Inteligência Artificial

**Algoritmo 1:**

**Código**

public class Algoritmo1 {

    public static void main( String[] args){

        for(int i = 0; i<429; i++){

            System.out.println(i+" "+f(i));

        }

    }

    public static int f(int n) {

        int i, j, k, res = 0;

        int cont\_op = 0;

        for(i = 1; i <= n+1; i += 1) {

            for(j = 1; j <= i\*i; j += i+1) {

                for(k = i/2; k <= n+j; k += 2) {

                    res = res + n-1;

                    cont\_op++;

                }

            }

        }

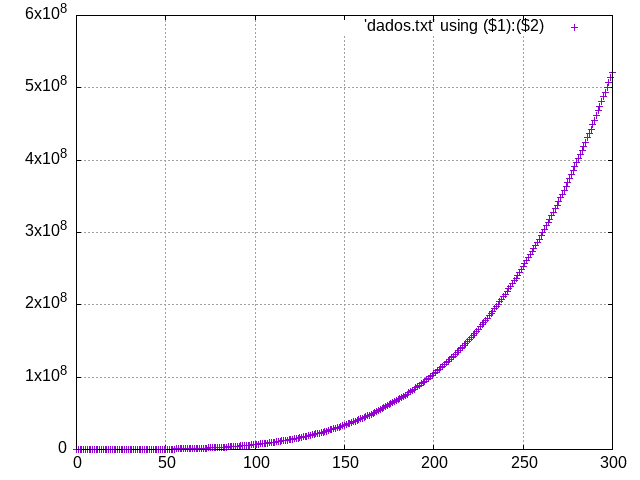
        return cont\_op;

    }

}

**Tabela**

|  |  |
| --- | --- |
| **n** | **f(n)** |
| 0 | 1 |
| 15 | 5420 |
| 30 | 66904 |
| 45 | 309304 |
| 60 | 933441 |
| 75 | 2216065 |
| 90 | 4509909 |
| 105 | 8243514 |
| 120 | 13921481 |

**Gráfico**

**Cálculo**

B = (LOG(15081)-LOG(19))/(LOG(20)-LOG(2))

B = 2,8997

**Conclusão**

O algoritmo apresenta uma função de complexidade polinomial cúbica (***n3***), tornando-se muito lento à medida que o número de ***n*** cresce. O algoritmo pode ser eficiente para entradas pequenas, mas sua eficiência cai muito rapidamente em casos maiores.

**Algoritmo 2:**

**Código**

public class Algoritmo2 {

    public static void main(String[] args){

        for(int i = 0; i<500; ++i){

            System.out.println(i+" "+f(i));

        }

    }

    public static int f( int n ) {

        int i, j, k, res = 0;

        int cont\_op = 0;

        for( i = n; i <= n; i += i/2+1 ) {

            for( j = i/2; j <= i\*i; j += i+1 ) {

                for( k = n; k <= 2\*n; k += i+1 ) {

                    res = res + n;

                    cont\_op++;

                }

            }

        }

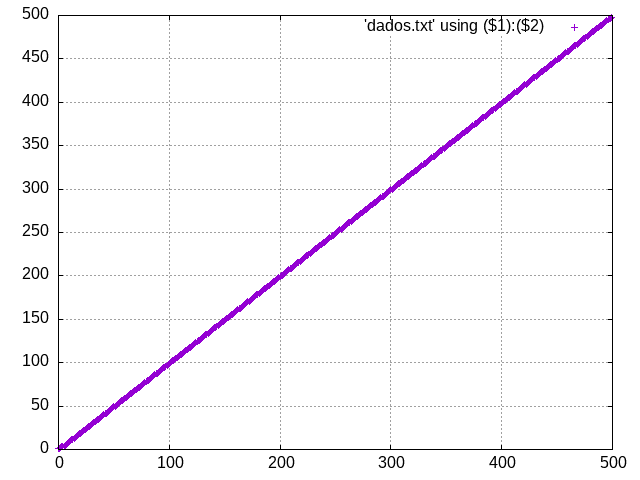
        return cont\_op;

    }

}

**Tabela**

|  |  |
| --- | --- |
| **n** | **f(n)** |
| 0 | 1 |
| 15 | 14 |
| 30 | 29 |
| 45 | 44 |
| 60 | 59 |
| 75 | 74 |
| 90 | 89 |
| 105 | 104 |
| 120 | 119 |

**Gráfico**

**Cálculo**

B = (LOG(19)-LOG(2))/(LOG(20)-LOG(2))

B = 0,9777

**Conclusão**

O algoritmo apresenta um grau de complexidade linear (***n***), tornando-se muito eficiente até mesmo para grandes entradas de dados. O algoritmo é ideal para problemas que exigem um processamento rápido, por possuir um cresimento de ***n*** proporcional a ***f(n)***.

**Algoritmo 3:**

**Código**

import java.lang.Math;

public class Algoritmo3 {

    public static void main(String[] args){

        for(int i = 0; i<1000; i = i + 15){

            System.out.println(i+" "+f(i));

        }

    }

    public static int f( int n ) {

        int i, j, k, res = 0;

        int cont\_op = 0;

        for( i = 1; i <= n\*n; i += 2 ){

            for( j = i/2; j <= 2\*i; j += i/2+1 ) {

                for( k = j+1; k <= n+j; k += k/2+1 ) {

                    res = res + Math.abs(j-i);

                    cont\_op++;

                }

            }

        }

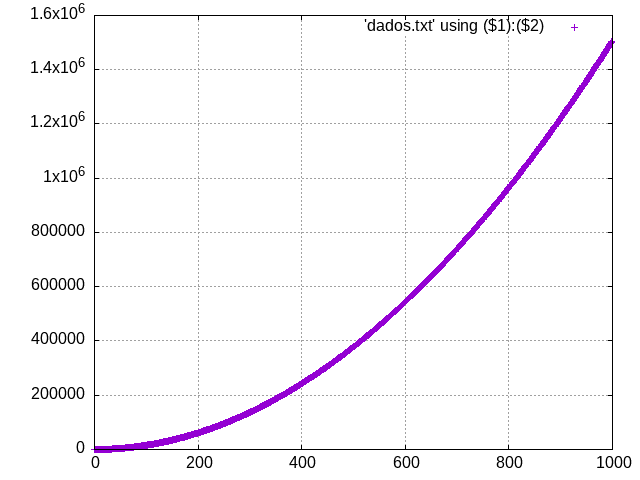
        return cont\_op;

    }

}

**Tabela**

|  |  |
| --- | --- |
| **n** | **f(n)** |
| 0 | 0 |
| 15 | 412 |
| 30 | 1525 |
| 45 | 3318 |
| 60 | 5782 |
| 75 | 8923 |
| 90 | 12744 |
| 105 | 17238 |
| 120 | 22400 |

**Gráfico**

**Cálculo**

B = (LOG(708)-LOG(7))/(LOG(20)-LOG(2))

B = 2,0049

**Conclusão**

O algoritmo apresenta um crescimento quadrático (***n2***). Embora seja razoavelmente eficiente para entradas pequenas, seu tempo de execução cresce bastante à medida que ***n*** aumenta. Portanto, o modelo não é indicado para ser utilizado em um grande volume de dados, uma vez que o tempo de execução se torna muito longo.

**Algoritmo 4:**

**Código**

public class Algoritmo4 {

    public static void main(String[] args) {

        for(long i = 0; i<58; i = i + 15){

            System.out.println(i+" "+f(i));

        }

    }

    public static int f( long n ) {

        long i, j, k, res = 0;

        int cont\_op = 0;

        for( i = n; i <= n\*n; i += 2 ) {

            for( j = n+1; j <= n\*n; j += 2 ) {

                for( k = j; k <= 2\*j; k += 2 ) {

                    res = res + 1;

                    cont\_op++;

                }

            }

        }

        return cont\_op;

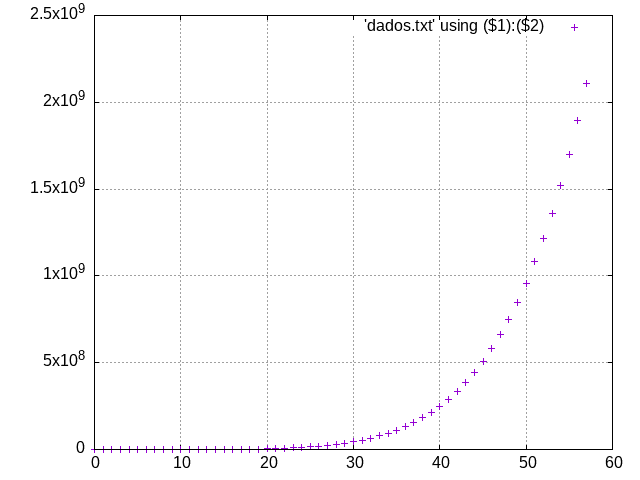
    }

}

**Tabela**

|  |  |
| --- | --- |
| **n** | **f(n)** |
| 0 | 0 |
| 15 | 678930 |
| 30 | 44190780 |
| 45 | 508695165 |

**Gráfico**

**Cálculo**

B = (LOG(3828595)-LOG(4))/(LOG(20)-LOG(2))

B = 5,9809

**Conclusão**

O algoritmo apresentado possui um grau de complexidade ***n6***, atingindo valores altíssimos com um número ainda baixo de ***n****.* É o algoritmo de maior complexidade entre os apresentados e sua utilização se torna impraticável, uma vez que para qualquer valor significativo de ***n***, o tempo de execução cresce excessivamente.

**Algoritmo 5:**

**Código**

public class Algoritmo5 {

    public static void main(String[] args) {

        for(int i =0; i<240; i = i + 15){

            System.out.println(i+" "+f(i));

        }

    }

    public static int f( int n ) {

        int i, j, k, res = 0;

        int cont\_op = 0;

        for( i = 1; i <= n\*n; i += 1 ) {

            for( j = 1; j <= i; j += 2 ) {

                for( k = n+1; k <= 2\*i; k += i\*j ) {

                    res = res + k+1;

                    cont\_op++;

                }

            }

        }

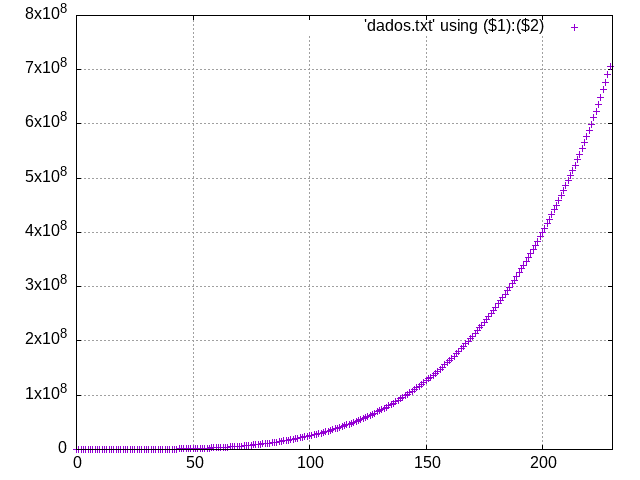
        return cont\_op;

    }

}

**Tabela**

|  |  |
| --- | --- |
| **n** | **f(n)** |
| 0 | 0 |
| 15 | 12963 |
| 30 | 203756 |
| 45 | 1028017 |
| 60 | 3245100 |
| 75 | 7918158 |
| 90 | 16414031 |
| 105 | 30403387 |
| 120 | 51860550 |

**Gráfico**

**Cálculo**

B = (LOG(40550)-LOG(7))/(LOG(20)-LOG(2))

B = 3,7628

**Conclusão**

O algoritmo apresenta complexidade de ***n4***. Embora não tão extremo quanto o algoritmo anterior, ainda apresenta um crescimento bastante acelerado e é inadequado para uso em médio/grande volume de dados.

*Link do projeto no Github com os scripts utilizados para plotar os gráficos. (*[*https://github.com/gustavo-losch/T1-Alest*](https://github.com/gustavo-losch/T1-Alest)*)*