PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

CAMPUS POÇOS DE CALDAS

CIÊNCIA DA COMPUTAÇÃO - 7° PERÍODO

ANÁLISE DA COMPLEXIDADE DE ALGORITMOS QUE UTILIZAM O CRIVO DE ERATOSTENES

João Marcelo Danza Gandini

Poços de Caldas

2023

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

CAMPUS POÇOS DE CALDAS

CIÊNCIA DA COMPUTAÇÃO - 7° PERÍODO

ANÁLISE DA COMPLEXIDADE DE ALGORITMOS QUE UTILIZAM O CRIVO DE ERATOSTENES

Trabalho nº 1 apresentado à disciplina de Complexidade e Algoritmos da Faculdade de Ciência da Computação como parte integrante da avaliação global.

Autor:
João Marcelo Danza Gandini

Professor responsável:

João Carlos de Moraes Morselli Júnior

Poços de Caldas

2023

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

1. RESUMO

O presente trabalho consiste na análise de algoritmos que utilizem o Crivo de Eratóstenes para a obtenção dos números primos em um intervalo entre zero e o valor máximo fornecido pelo usuário.

Foram analisados algoritmos desenvolvidos nas linguagens C e Python, sendo que um deles foi desenvolvido pelo próprio autor.

O objetivo é analisar os algoritmos no intuito de se obter suas funções de complexidade no pior caso, ou seja, O(n), compará-los entre si e com as funções de complexidade informadas em sites da internet, onde alguns dos algoritmos analisados foram disponibilizados.

2. INTRODUÇÃO

> *Um algoritmo é um procedimento, que consistindo em um conjunto de regras não ambíguas, as quais especificam, para cada entrada, uma sequência finita de operações, terminando com uma saída correspondente. Um algoritmo resolve um problema quando, para qualquer entrada, produz uma resposta correta, se forem concedidos tempo e memória suficientes para sua execução. O fato de um algoritmo resolver (teoricamente) um problema não significa que seja aceitável na prática. Os recursos de espaço e tempo requeridos têm grande importância em casos práticos.(TOSCANI, 2012, p.2)*

A complexidade de algoritmos é um ramo da ciência da computação que estuda o desempenho dos algoritmos em termos de tempo e espaço. A análise de algoritmos é o processo de determinar a complexidade de um algoritmo.

Entre as medidas de complexidade, a complexidade no pior caso (O) é o critério de avaliação mais utilizado.

A complexidade de tempo é geralmente expressa em notação assintótica f(n), onde n é o tamanho da entrada do algoritmo, que é uma maneira de descrever o comportamento de uma função à medida que seu argumento tende ao infinito.

As classes de complexidade de tempo mais comuns são:

- O(1): O algoritmo leva um tempo constante, independentemente do tamanho do problema.
- O(n): O algoritmo leva um tempo linear, que é proporcional ao tamanho do problema.
- $O(n^2)$: O algoritmo leva um tempo quadrático, que é proporcional ao quadrado do tamanho do problema.
- Outras funções, como por exemplo, $O(n^k)$ que também é exponencial, sendo k um inteiro qualquer e O(n log n) também podem ser encontradas.

A análise de algoritmos é o processo de determinar a complexidade de um algoritmo. A análise de algoritmos pode ser realizada de várias maneiras, incluindo:

- Análise informal: A análise informal é uma abordagem intuitiva para determinar a complexidade de um algoritmo.
- Análise matemática: A análise matemática é uma abordagem mais formal através do uso de conceitos e resultados matemáticos para a determinação da complexidade de um algoritmo.
- Análise experimental: A análise experimental é uma abordagem que utiliza experimentos para determinar a complexidade de um algoritmo.

De acordo com Toscani (2012), existem vários aspectos que devem ser considerados na determinação da complexidade de um algoritmo. Uma questão é a da medida empírica, onde pode-se pensar em medir experimentalmente a quantidade de trabalho (tempo ou memória) requerida por um algoritmo executado em um computador específico. Entretanto, uma medida empírica é fortemente dependente, tanto do programa quanto da máquina usada para implementar o algoritmo. Assim, uma pequena mudança no programa pode não representar uma mudança significativa no algoritmo, mas apesar disso afetar a velocidade de execução. Outro ponto importante a ser destacado é que, se dois programas, para o mesmo algoritmo, são comparados, primeiro numa máquina, depois em outra, as comparações podem apresentar resultados diferentes.

As medidas de desempenho podem ser classificadas em tempo e espaço requeridos por um algoritmo, relacionadas à velocidade e à quantidade de memória, respectivamente. A complexidade em espaço usa como medida de desempenho a quantidade de memória necessária para a execução de algoritmo.

Uma das medidas de desempenho mais importante, e que receberá mais atenção neste trabalho, é o tempo de execução, tendo-se, portanto, a complexidade em tempo, que pode ser medido pelo número de execuções de algumas operações ou custo de suas sequências de execução.

Toscani (2012) afirma também que:

> *“Para medir a quantidade de trabalho realizado por um algoritmo, costuma-se escolher uma operação, chamada operação fundamental. A operação escolhida como fundamental deve ser tal que a contagem do número de vezes que ela é executada expresse a quantidade de trabalho do algoritmo, dispensando outras medidas.”*

2.1 Crivo de Eratóstenes

Na matemática computacional há diversas maneiras de checar se um número é primo ou não, com diferentes algoritmos que possuem seus pontos fortes e fracos. (PAES e FIGUEIREDO, 2019)

O algoritmo chamado Crivo de Eratóstenes, objeto de análise do presente trabalho, é um algoritmo muito antigo, que foi originalmente descrito pelo matemático grego Eratóstenes (a.C. 285-194 a.C.) no século III a.C. O algoritmo foi posteriormente redescoberto por vários matemáticos ao longo da história, incluindo Al-Khwarizmi, Leonardo Fibonacci e Leonhard Euler.

Este é um algoritmo simples e eficiente, frequentemente utilizado em aplicações que requerem a geração de uma lista de números primos menores ou iguais a um dado número N.

De maneira simplificada, o algoritmo funciona da seguinte forma:

1. Começa com um conjunto de números de 2 até N.
2. Remove todos os múltiplos de 2 do conjunto.
3. Repete o passo 2 com o próximo número primo, que é o próximo número não removido do conjunto.
4. Continua repetindo os passos 2 e 3 até que não haja mais números no conjunto.

A Figura 1 apresenta uma visualização da obtenção do conjunto de números primos entre 2 e 120 que ilustra o funcionamento do algoritmo.

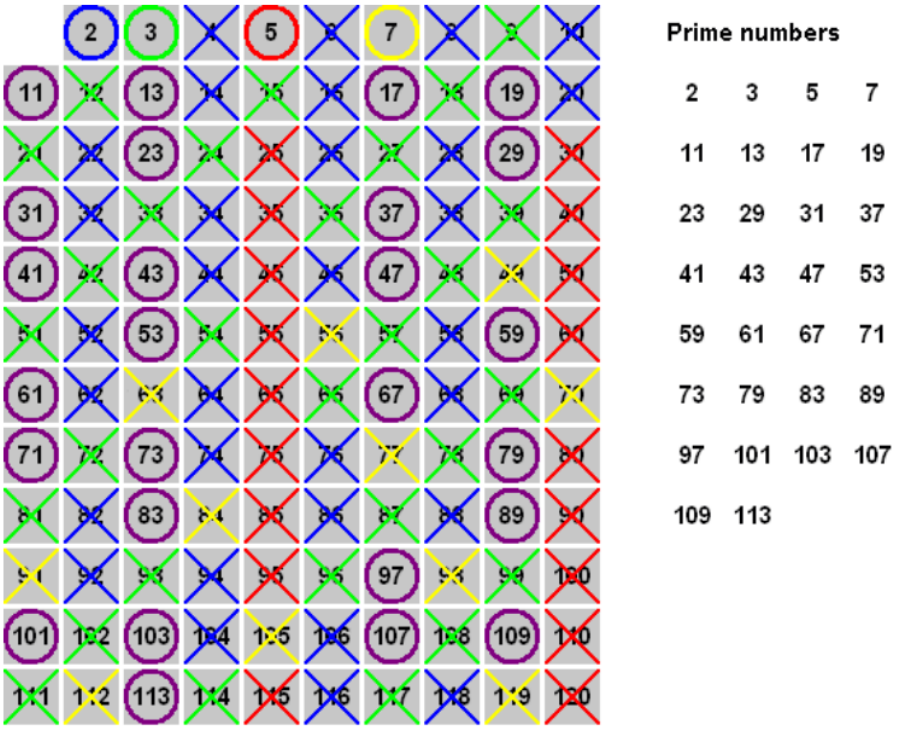

Figura 1 – Visualização da obtenção de números primos.
Fonte: (PAES e FIGUEIREDO, 2019)

O Núcleo Olímpico de Incentivo ao Conhecimento – NOIC, apresenta um exemplo de código fonte deste algoritmo, anexo 1, e afirma que sua complexidade é O(N.logN).

3. OBJETIVOS

3.1 Objetivo Geral

O objetivo geral deste trabalho é comparar a complexidade, em termos de tempo de execução e quantidade de instruções realizadas, de algoritmos que utilizem o crivo de Eratóstenes nas linguagens Python e C.

3.2 Objetivos Específicos

1. Verificar a quantidade de instruções e o tempo de execução em algoritmos obtidos na internet, um deles em linguagem C e outro em Python para diferentes valores de n.
2. Verificar a quantidade de instruções e o tempo de execução em uma algoritmo desenvolvido pelo autor na liguagem Python para diferentes valores de n.
3. Plotar gráficos com os valores obtidos nas verificações anteriores, realizar as comparações entre as linguagens e entre os algoritmos da internet e os desenvolvidos pelo autor.

4. Metodologia

Para realizar a comparação, serão analisados quatro algoritmos, dois em Python e dois em C, que implementem o algoritmo do crivo de Eratóstenes. Os algoritmos serão executados para diferentes valores de N, registrando-se o tempo de execução e a quantidade de instruções realizadas.

Após cada uma das execuções, os valores obtidos serão registrados e utilizados na geração de gráficos, utilizando o software Microsoft Excel®, no intuito de se obter uma função que represente a complexidade dos algoritmos analisados no seu pior caso, O(n).

Os resultados serão analisados e comparados, no intuito de se obter uma avaliação das complexidades dos algoritmos utilizados.

Todas as execuções serão realizadas no mesmo equipamento, com as seguintes características:

- Notebook HP Pavillion G4-1130br
  CPU Intel Core i3 M 370 2.4Ghz
  6.0Gb memória RAM DDR3 1067Mhz
  Disco rígido com capacidade de 500Gb com 252Gb livres
  Sistema Operacional Windows 10

5. MATERIAIS UTILIZADOS

- Computador com sistema operacional Windows 10
- Interpretador Python
- Compilador para a linguagem C
- Software Visual Studio Code

6. DESENVOLVIMENTO

Os algoritmos utilizados para a realização das análises e comparações serão apresentados nos Anexos do presente trabalho, sendo no Anexo 1 o algoritmo disponível no site do Núcleo Olímpico de Incentivo ao Conhecimento (NOIC), no Anexo 2 o algoritmo na linguagem C obtido na internet, no Anexo 3 o algoritmo na linguagem Python obtido na internet e no Anexo 4 o algoritmo em Python desenvolvido pelo próprio autor.

O objetivo das análises apresentadas abaixo é a obtenção das funções de complexidade dos algoritmos analisados para seu pior caso, ou seja, O(n).

6.1 Algoritmo internet NOIC (linguagem C)

Este algoritmo foi disponibilizado pelo Núcleo Olímpico de Incentivo ao Conhecimento (NOIC), disponibilizado no endereço eletrônico: https://noic.com.br/materiais-informatica/curso/math-03/.

No site, os autores afirmam que o algoritmo possui complexidade O(n)=n.log(n) e apresentam o desenvolvimento dos cálculos das séries matemáticas.

Contudo, após a utilização do algoritmo para verificação dos números primos nos intervalos supracitados, os valores dos tempos de execução e quantidade de instruções foram inseridos no software Microsoft Excel e através dele foram gerados os gráficos referentes aos tempos de execução e a quantidade de instruções realizadas pelo algoritmo.

Após a geração dos gráficos e a inserção da linha de tendência no Excel, foi possível obter a equação da função e o valor de R^2. Percebeu-se que o resultado não convergiu para uma equação condizente com a complexidade informada no site do NOIC.

Os resultados das execuções do algoritmo para uma quantidade de valores de N diferentes são apresentados na Tabela 1, Gráfico 1 e Gráfico 2.

Crivo Eratostenes C (NOIC)			
N	TEMPO EXECUÇÃO (milisegundos)	TEMPO EXECUÇÃO (segundos)	QUANTIDADE DE INSTRUÇÕES
100	11	0,011	345
500	38	0,038	1912
1.000	73	0,073	3957
10.000	656	0,656	43070
100.000	4132	4,132	456807
500.000	17297	17,297	2358601
1.000.000	33381	33,381	4775209
10000000	296405	296,405	49465737
20.000.000	571031	571,031	99857425
30.000.000	825277	825,277	150569830

Tabela 1 – Resultados obtidos execução algoritmo NOIC
Fonte: Autor, 2023

A tabela acima apresenta os valores de N inseridos, o tempo de execução do algoritmo para cada um deles em milissegundo, em segundos e a quantidade de instruções realizadas para cada N.

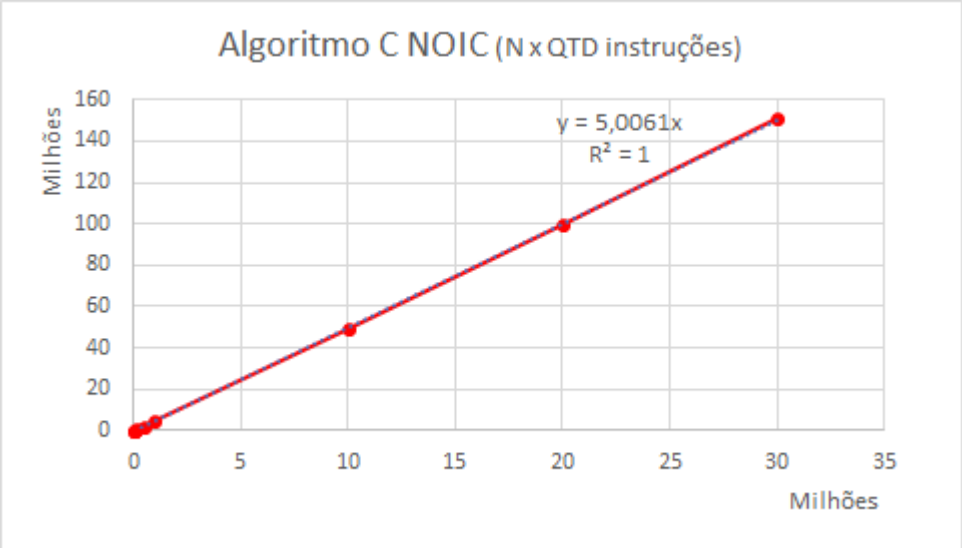

Gráfico 1 – N x quantidade de execuções (algoritmo NOIC)
Fonte: Autor, 2023

No gráfico acima, em relação à quantidade de valores de N inseridos versus a quantidade de execuções, a função apresenta um comportamento linear com $R^2 = 1$ e a seguinte equação:

$$y = 5{,}0061.x$$

onde,

y = quantidade de instruções realizadas

x = quantidade de valores inseridos (n)

Em se tratando do tempo de execução do algoritmo, tem-se o comportamento apresentado no Gráfico 2.

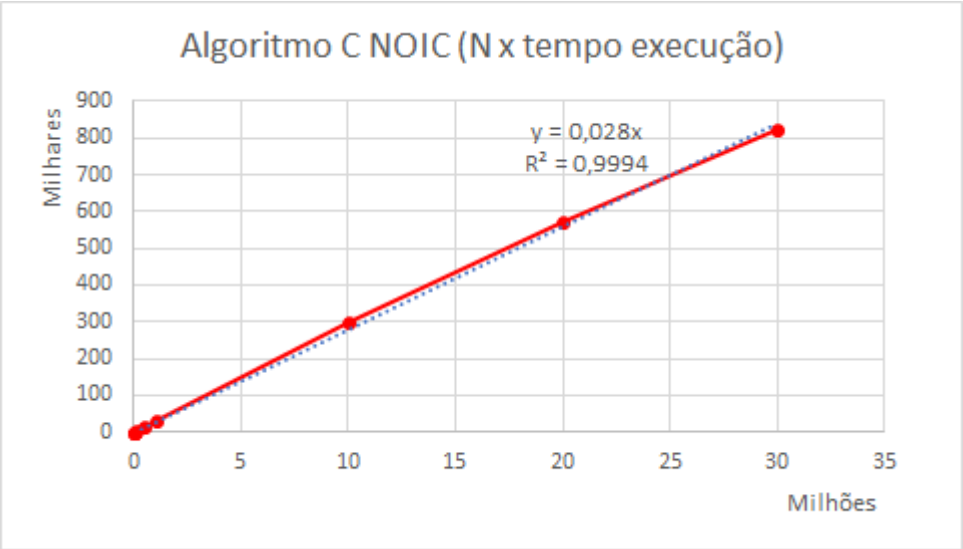

Gráfico 2 – N x tempo de execução (algoritmo NOIC)
Fonte: Autor, 2023

No gráfico apresentado, pode-se verificar que, em relação ao tempo de execução o algoritmo apresentou uma função com comportamento linear, com R^2=0,9994 e a seguinte equação:

$$y = 0{,}028x$$

onde,

y = tempo de execução do algoritmo

x = quantidade de valores inseridos (n)

Realizando-se uma simulação com a equação acima e os mesmos valores de N fornecidos para a execução do algoritmo, obteve-se os resultados apresentados na Tabela 2.

Tempo de execução do algoritmo NOIC			
	Equação gráfico y = 0,028x	Valor obtido algoritmo	Diferença
100	3	11	-292,86%
500	14	38	-171,43%
1.000	28	73	-160,71%
10.000	280	656	-134,29%
100.000	2800	4132	-47,57%
500.000	14000	17297	-23,55%
1.000.000	28000	33381	-19,22%
10.000.000	280000	296405	-5,86%
20.000.000	560000	571031	-1,97%
30.000.000	840000	825277	1,75%

Tabela 2 – Comparação resultados (equação x valor obtido algoritmo NOIC)
Fonte: Autor, 2023

Pode-se perceber relativa proximidade entre os valores obtidos na execução do algoritmo e a simulação realizada através da equação fornecida pelo Excel à medida que o valor de N se aproxima de 30.000.000, indicando uma convergência, fato que, evidencia ainda mais que o comportamento da função de complexidade do algoritmo como linear e não de ordem logarítmica como apresentado pelo NOIC.

Para um maior aprofundamento no estudo da função de complexidade, foram pesquisados mais 2 algoritmos na internet, sendo 1 em linguagem C, para uma comparação mais direta com o algoritmo do NOIC e outro na linguagem Python, para que os resultados dos algoritmos de duas linguagens diferentes pudessem ser comparados.

6.2 Algoritmo internet (linguagem C)

Este algoritmo foi adaptado pelo autor, tendo como base o algoritmo disponibilizado no site https://blog.tiagomadeira.com/2007/06/crivo-de-eratostenes/ e apresentado no Anexo 2 do presente trabalho.

Foi adotado o mesmo procedimento realizado anteriormente, ou seja, sucessivas execuções do algoritmo, lançamento dos valores obtidos no Excel, geração dos gráficos, inserção da linha de tendência, obtenção da equação da função e valor de R^2.

Os resultados das execuções do algoritmo para uma quantidade de valores de N diferentes são apresentados na Tabela 3, Gráfico 3 e Gráfico 4.

Crivo Eratostenes C (internet)			
N	TEMPO EXECUÇÃO (milisegundos)	TEMPO EXECUÇÃO (segundos)	QUANTIDADE DE INSTRUÇÕES
100	23	0,023	230
500	44	0,044	1734
1.000	70	0,070	3577
10.000	480	0,480	38088
100.000	3716	3,716	249998
500.000	17059	17,059	1249998
1.000.000	32707	32,707	4198836
10000000	283355	283,355	43495187
20.000.000	563712	563,712	87845489
30.000.000	823856	823,856	132469335

Tabela 3 – Resultados obtidos execução algoritmo em C internet
Fonte: Autor, 2023

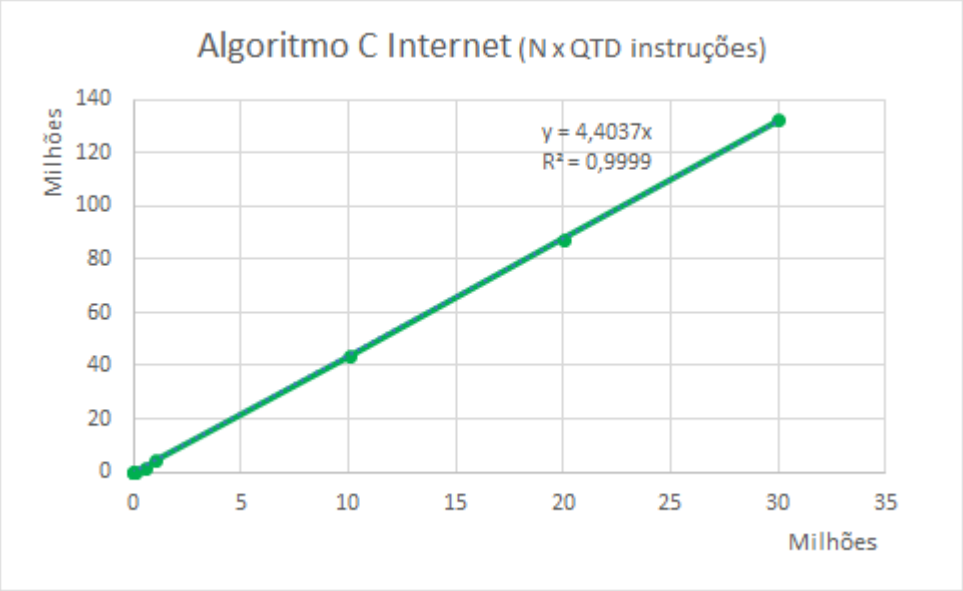

Gráfico 3 – N x quantidade de execuções (algoritmo C internet)
Fonte: Autor, 2023

No gráfico acima, em relação à quantidade de valores de N inseridos versus a quantidade de execuções, a função apresenta um comportamento linear com R^2=0,9999 e a seguinte equação:

$$y = 4{,}4037.x$$

onde,

y = quantidade de instruções realizadas

x = quantidade de valores inseridos (n)

Em se tratando do tempo de execução do algoritmo, tem-se o comportamento apresentado no Gráfico 4

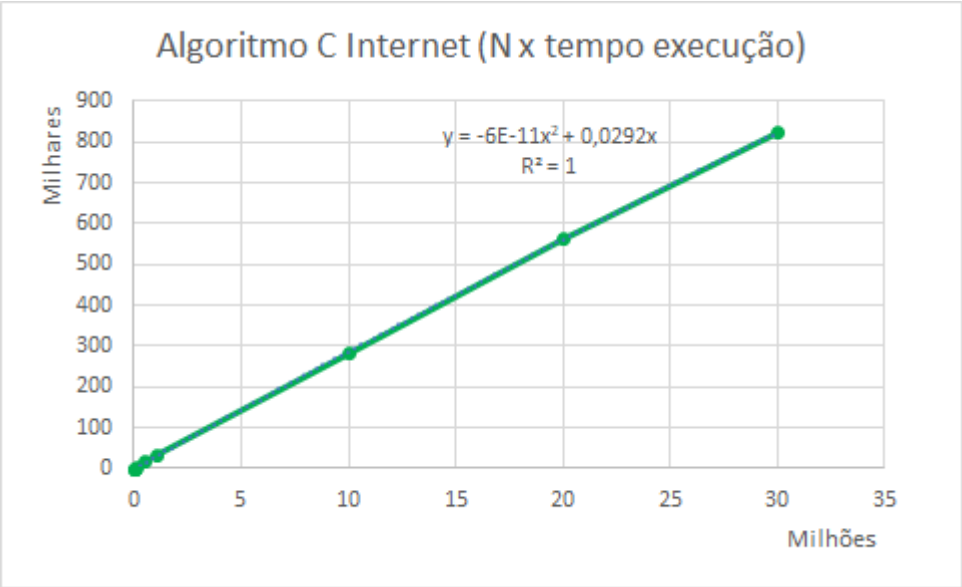

Gráfico 4 – N x tempo de execução (algoritmo C internet)
Fonte: Autor, 2023

No gráfico apresentado, pode-se verificar que, em relação ao tempo de execução o algoritmo apresentou uma função com comportamento polinomial quadrática com $R^2 = 1$ e a seguinte equação:

$$y = -6E^{-11}x^2 + 0{,}0292x$$

onde,

y = tempo de execução do algoritmo

x = quantidade de valores inseridos (n)

Realizando-se uma simulação com a equação acima e os mesmos valores de N fornecidos para a execução do algoritmo, obteve-se os resultados apresentados na Tabela 4.

Tempo de execução do algoritmo (C internet)			
	Equação gráfico $y = -6E\text{-}11x^2 + 0{,}0292x$	Valor obtido algoritmo	Diferença
100	3	23	-687,67%
500	15	44	-201,37%
1.000	29	70	-139,73%
10.000	292	480	-64,39%
100.000	2919	3716	-27,29%
500.000	14585	17059	-16,96%
1.000.000	29140	32707	-12,24%
10.000.000	286000	283355	0,92%
20.000.000	560000	563712	-0,66%
30.000.000	822000	823856	-0,23%

Tabela 4 – Comparação resultados (equação x valor obtido algoritmo C internet)
Fonte: Autor, 2023

Pode-se perceber, novamente, relativa proximidade entre os valores obtidos na execução do algoritmo e a simulação realizada através da equação fornecida pelo Excel à medida que o valor de N se aproxima de 30.000.000, indicando uma convergência, fato que, mais uma vez evidencia o comportamento da função de complexidade do algoritmo como linear e não de ordem logarítmica.

6.3 Algoritmo internet (linguagem Python)

Este algoritmo foi utilizado praticamente na íntegra conforme o disponibilizado no site https://pt.stackoverflow.com/questions/231555/como-gerar-200-000-primos-o-mais-r%C3%A1pido-poss%C3%ADvel-em-python e apresentado no Anexo 3 do presente trabalho. As únicas alterações realizadas pelo autor foram as inserções das instruções responsáveis pela obtenção do tempo de execução e quantidade de instruções realizadas.

Todos os procedimentos descritos anteriormente foram mantidos para a obtenção dos dados, tabelas e gráficos.

Os resultados das execuções do algoritmo para uma quantidade de valores de N diferentes são apresentados na Tabela 5, Gráfico 5 e Gráfico 6.

Crivo Eratostenes Python (internet)			
N	TEMPO EXECUÇÃO (milisegundos)	TEMPO EXECUÇÃO (segundos)	QUANTIDADE DE INSTRUÇÕES
100	0	0,000	230
500	0	0,000	1252
1.000	0	0,000	2580
10.000	0	0,000	28211
100.000	47	0,047	302671
500.000	250	0,250	1575456
1.000.000	500	0,500	3200547
10000000	5656	5,656	33514631
20.000.000	11047	11,047	67858644
30.000.000	16953	16,953	102491969

Tabela 5 – Resultados obtidos execução algoritmo em Python internet
Fonte: Autor, 2023

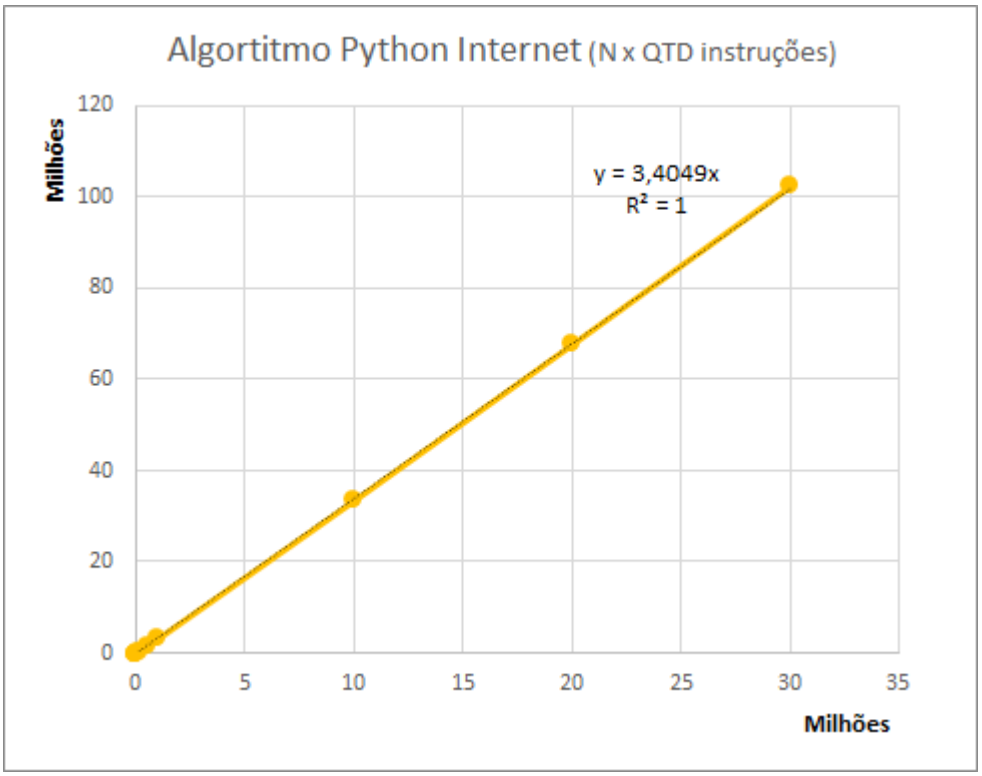

Gráfico 5 – N x quantidade de execuções (algoritmo Python internet)
Fonte: Autor, 2023

No gráfico acima, em relação à quantidade de valores de N inseridos versus a quantidade de instruções, a função apresenta um comportamento linear com $R^2 = 1$ e a seguinte equação:

$$y = 3{,}4049.x$$

onde,

y = quantidade de instruções realizadas

x = quantidade de valores inseridos (n)

Em se tratando do tempo de execução do algoritmo, tem-se o comportamento apresentado no Gráfico 6.

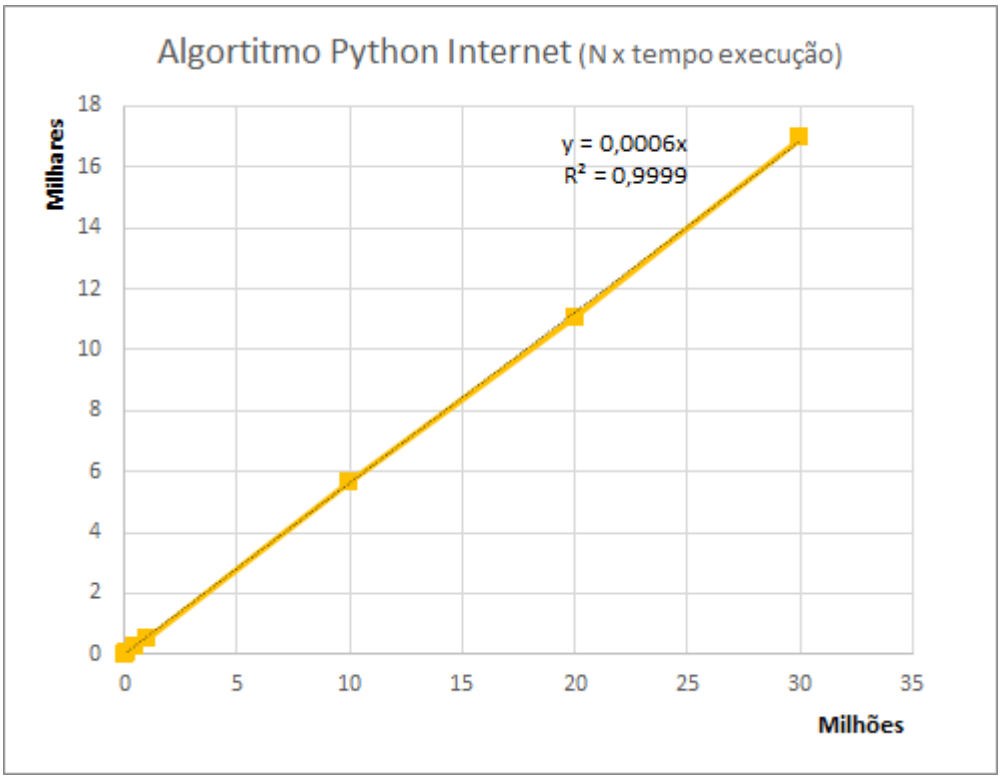

Gráfico 6 – N x tempo de execução (algoritmo Python internet)
Fonte: Autor, 2023

No gráfico apresentado, pode-se verificar que, em relação ao tempo de execução o algoritmo apresentou uma função com comportamento linear com R^2=0,9999 e a seguinte equação:

$$y = 0{,}0006x$$

onde,

y = tempo de execução do algoritmo

x = quantidade de valores inseridos (n)

Realizando-se uma simulação com a equação acima e os mesmos valores de N fornecidos para a execução do algoritmo, tem-se os resultados apresentados na Tabela 4.

Tempo de execução do algoritmo			
	Equação gráfico y = 0,0006x	Valor obtido algoritmo	Diferença
100	0	0	100,00%
500	0	0	100,00%
1.000	1	0	100,00%
10.000	6	0	100,00%
100.000	60	46,875	21,88%
500.000	300	250	16,67%
1.000.000	600	500	16,67%
10.000.000	6000	5656,25	5,73%
20.000.000	12000	11046,875	7,94%
30.000.000	18000	16953,125	5,82%

Tabela 6 – Comparação resultados (equação x valor obtido algoritmo Python internet)
Fonte: Autor, 2023

Pode-se perceber, novamente, relativa proximidade entre os valores obtidos na execução do algoritmo e a simulação realizada através da equação fornecida pelo Excel à medida que o valor de N se aproxima de 30.000.000, indicando uma convergência, fato que, mais uma vez evidencia o comportamento da função de complexidade do algoritmo como linear e não de ordem logarítmica, mesmo utilizando-se uma outra linguagem de programação.

6.4 Algoritmo autor (linguagem Python)

A título de comparação com os demais algoritmos apresentados foi desenvolvido pelo autor uma implementação de um algoritmo recursivo, em Python, apresentado no Anexo 4. Contudo, em relação aos demais algoritmos apresentados, nota-se que os valores de N são inferiores aos apresentados anteriormente, pois diferentemente dos demais algoritmos, que possuem uma função de complexidade linear, este possui uma função polinomial quadrática tanto para a quantidade de instruções realizadas quanto para os tempos de execução, o que demanda um custo computacional muito superior aos demais.

Os resultados das execuções do algoritmo para uma quantidade de valores de N diferentes são apresentados na Tabela 7, Gráfico 7 e Gráfico 8.

Crivo Eratostenes Python (autor)			
N	TEMPO EXECUÇÃO (milisegundos)	TEMPO EXECUÇÃO (segundos)	QUANTIDADE DE INSTRUÇÕES
100	15,625	0,016	211
500	16	0,016	1476
1.000	31	0,031	3442
10.000	594	0,594	52754
100.000	55734	55,734	939524
500.000	1720516	1720,516	7348183
1.000.000	6874250	6874,250	18045627

Tabela 7 – Resultados obtidos execução algoritmo em Python autor
Fonte: Autor, 2023

Pode-se notar, na tabela acima os valores de tempo de execução e quantidade de instruções muito superiores aos dos algoritmos previamente apresentados.

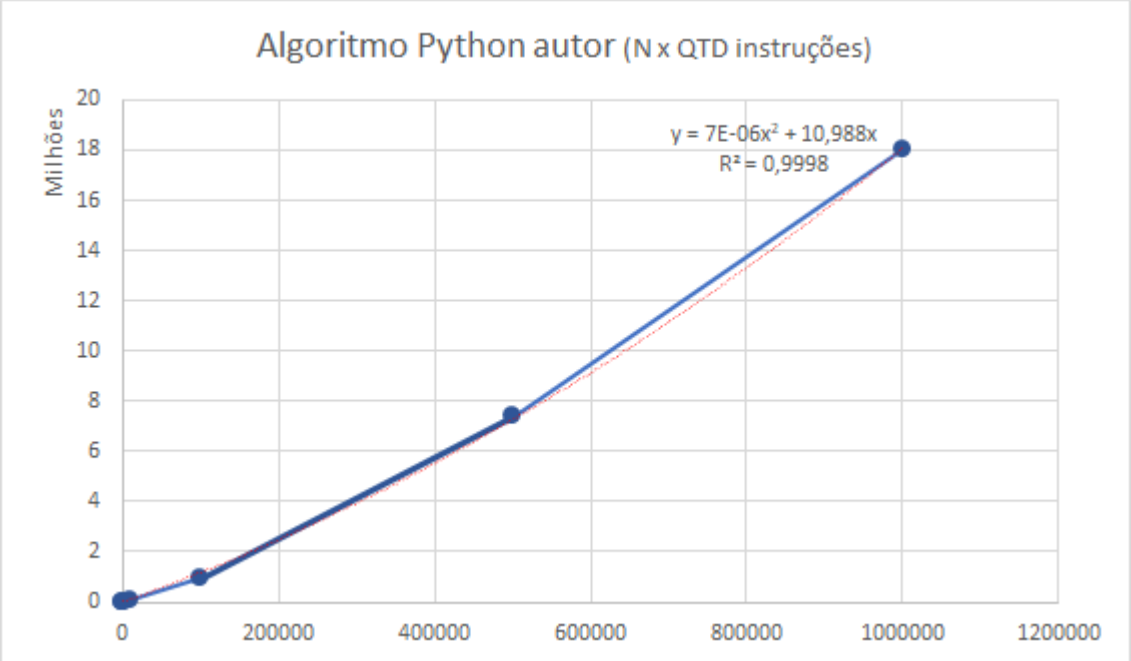

Gráfico 7 – N x quantidade de execuções (algoritmo Python autor)
Fonte: Autor, 2023

No gráfico acima, em relação à quantidade de valores de N inseridos versus a quantidade de execuções, a função apresenta um comportamento linear com R^2=0,9998 e a seguinte equação:

$$y = 7.E^{-6}x^2 + 10{,}988x$$

onde,

y = quantidade de instruções realizadas

x = quantidade de valores inseridos (n)

Em se tratando do tempo de execução do algoritmo, tem-se o comportamento apresentado no Gráfico 8

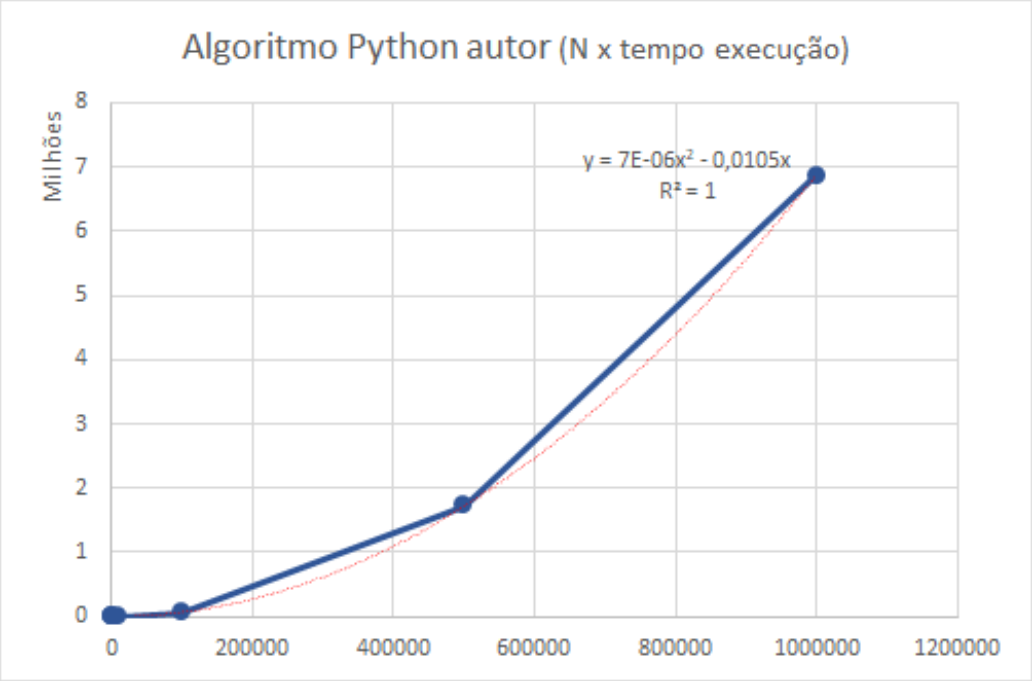

Gráfico 8 – N x tempo de execução (algoritmo Python autor)
Fonte: Autor, 2023

No gráfico apresentado, pode-se verificar que, em relação ao tempo de execução o algoritmo apresentou uma função com comportamento polinomial quadrática com R^2 = 1 e a seguinte equação:

$$y = 7E^{-6}x^2 - 0{,}0105x$$

onde,

y = tempo de execução do algoritmo

x = quantidade de valores inseridos (n)

Pode-se perceber, no gráfico, um afastamento da linha de tendência em relação aos dados obtidos, que pode ser justificado pelo fato da função ser quadrática e a grande distância entre os valores de N inseridos

Realizando-se uma simulação com a equação acima e os mesmos valores de N fornecidos para a execução do algoritmo, tem-se os resultados apresentados na Tabela 8.

Tempo de execução do algoritmo			
N	**Equação gráfico $7E\text{-}06x^2 - 0{,}0105x$**	**Valor obtido algoritmo**	**Diferença**
100	1	15,625	-1295,09%
500	7	15,625	-123,21%
1.000	18	31,25	-78,57%
10.000	805	593,75	26,24%
100.000	71050	55734,375	21,56%
500.000	1755250	1720515,63	1,98%
1.000.000	7010500	6874250	1,94%

Tabela 8 – Comparação resultados (equação x valor obtido algoritmo Python autor)
Fonte: Autor, 2023

Desta vez, pode-se perceber a relativa proximidade entre os valores obtidos na execução do algoritmo e a simulação realizada através da equação fornecida pelo Excel à medida que o valor de N se aproxima de 1.000.000, indicando uma convergência, fato que, mais uma vez evidencia o comportamento da função de complexidade do algoritmo como polinomial e não de ordem logarítmica, mesmo utilizando-se outro algoritmo e uma outra linguagem de programação.

6.5 Resultados comparativos

Após a apresentação individualizada de cada um dos algoritmos analisados foi realizada uma análise comparativa, levando-se em consideração os tempos de execução de todos os algoritmos analisados, apresentada na Tabela 9 e Gráfico 9.

Crivo Eratostenes - Tempo de execução (milisegundos)				
N	**C (NOIC)**	**C (internet)**	**Python (internet)**	**Python (autor)**
100	11	23	0	16
500	38	44	0	16
1000	73	70	0	31
10.000	656	480	0	594
100.000	4132	3716	47	55734
500.000	17297	17059	250	1720516
1.000.000	33381	32707	500	6874250
10.000.000	296405	283355	5656	
20.000.000	571031	563712	11047	
30.000.000	825277	823856	16953	

Tabela 9 – Comparação resultados entre os algoritmos analisados
Fonte: Autor, 2023

Analisando a tabela acima pode-se perceber a superioridade do algoritmo em Python obtido na internet dentre os demais, com tempos de execução muito inferiores para todos os valores de N testados.

Outro ponto a ser destacado é que os tempos de execução do algoritmo desenvolvido em Python pelo autor foram compatíveis com os demais até o limite de N=1000, sendo que para valores maiores, o tempo de execução foi muito superior, fato que comprova a diferença das funções de complexidade linear para a polinomial quadrática, conforme apresentado anteriormente.

O Gráfico 9 ilustra os dados apresentados na tabela, a fim de evidenciar e facilitar a percepção do comportamento de cada uma das funções para diferentes valores de N.

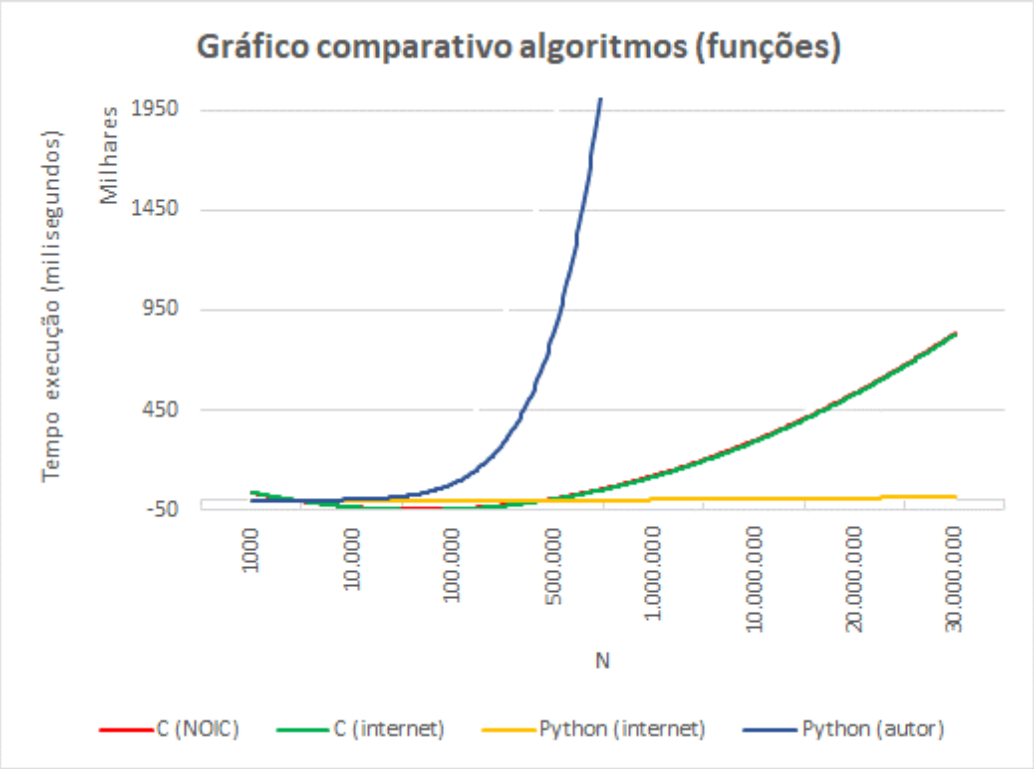

Gráfico 9 – Comparação resultados entre os algoritmos analisados
Fonte: Autor, 2023

Nota-se também que os algoritmos da linguagem C, tanto o do NOIC, quando o obtido na internet, possuem praticamente o mesmo comportamento, uma vez que as curvas de suas funções estão extremamente próximas, sendo que no gráfico, devido à sua escala reduzida, fica difícil distingui-las.

No gráfico acima, percebe-se, de forma mais clara a curva das funções de complexidade O(n) dos algoritmos analisados, sendo o algoritmo em Python obtido na internet o mais eficiente e o algoritmo em Python desenvolvido pelo autor, o mais ineficiente dentre os analisados no presente trabalho.

7. CONCLUSÃO

A complexidade de algoritmos é uma ferramenta importante para a ciência da computação. A análise de algoritmos pode ser utilizada para comparar a eficiência de diferentes algoritmos, e para selecionar o algoritmo mais adequado para uma determinada tarefa.

O crescente avanço tecnológico e a criação de máquinas cada vez mais rápidas, pode, ingenuamente, parecer ofuscar a importância da complexidade de tempo de um algoritmo.

No presente trabalho pode-se notar que as funções apresentadas pelo Excel com $R^2 = 1$, algumas vezes se apresentaram menos eficientes daquelas com R^2 ligeiramente inferior, pois o gráfico apresentava, em alguns casos iniciais, valores negativos para tempos de execução, fato que não procede.

Outro ponto importante a ser destacado é que foi notada uma variação da função apresentada pelo Excel quando os dados foram analisados de maneira isolada daqueles apresentados no gráfico comparativo entre todas as funções, fato que necessita de uma averiguação mais aprofundada.

Como recomendações futuras, faz-se necessário a realização de mais simulações, com valores de N superiores a 30.000.000, para verificação da convergência da equação apresentada pelo Excel e os resultados das execuções dos algoritmos, pois esta foi a tendência apresentada nas análises realizadas.

Após a realização do trabalho pode-se afirmar que o resultado foi satisfatório e o objetivo alcançado, podendo-se comprovar, na prática, os estudos teóricos realizados.

8. REFERÊNCIAS BIBLIOGRÁFICAS

AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. D. Data structures and algorithms. Reading: Addison-Wesley, 1983.

Blog Tiago Madeira - https://blog.tiagomadeira.com/2007/06/crivo-de-eratostenes/, acesso em 19/11/2023

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. Introduction to algorithms. Cambridge: MIT, 1990.

PAES, Leonardo, FIGUEIREDO, Lúcio. Núcleo Olímpico de Incentivo ao Conhecimento – NOIC, 2019. https://noic.com.br/materiais-informatica/curso/math-03/, acesso em 19/11/2023.

SEDGEWICK, R., & WAYNE, K. Algorithms (4th ed.). Addison-Wesley, 2011.

STACK OVERFLOW - https://pt.stackoverflow.com/questions/231555/como-gerar-200-000-primos-o-mais-r%C3%A1pido-poss%C3%ADvel-em-python, acesso em 19/11/2023.

TOSCANI, Laira Vieira; VELOSO, Paulo A. S. Complexidade de algoritmos: análise, projeto e métodos. 3. ed. Porto Alegre: Bookman, 2012. E-book. (Série livros didáticos informática UFRGS, 13). ISBN 9788540701397.

ANEXO 1

Algoritmo do crivo de Eratóstenes, disponibilizado pelo NOIC

https://noic.com.br/materiais-informatica/curso/math-03/

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define maxn 30000000 // Substitua 1000 pelo tamanho desejado

int eh_primo[maxn]; // -1 => indefinido, 0 => composto e 1 => primo.

void crivo(int n) {
    int n_exec = 0;
    clock_t start_time, end_time;

    start_time = clock(); // Captura o tempo inicial

    // Inicializando "eh_primo" com -1, pois não sabemos nada sobre nenhum número, inicialmente.
    for (int i = 1; i <= n; i++) {
        eh_primo[i] = -1;
        n_exec++;
    }

    // Para cada número de 2 até n
    for (int i = 2; i <= n; i++) {
        n_exec++;

        // checo se o número atual é indefinido.
        if (eh_primo[i] == -1) {

            // Se ele for, indico que ele é primo
            eh_primo[i] = 1;
            printf("%d eh primo\n", i); // Exibe o número encontrado

            // e que os múltiplos dele são compostos.
            for (int j = i + i; j <= n; j += i) {
                eh_primo[j] = 0;
                n_exec++;
            }
        }
    }

    end_time = clock(); // Captura o tempo final
    double total_time = ((double)(end_time - start_time)) / CLOCKS_PER_SEC * 1000.0; // Calcula o tempo
total em milissegundos
    printf("n-exec: %d\n", n_exec); // Exibe o número de instruções executadas
    printf("Tempo de execucao: %.2f ms\n", total_time); // Exibe o tempo total de execução
}

int main() {
    crivo(maxn);

    printf("Pressione Enter para fechar o programa...");
    getchar(); // Aguarda uma tecla antes de fechar o programa

    return 0;
}
```

ANEXO 2

Algoritmo do crivo de Eratóstenes em C obtido na internet

https://blog.tiagomadeira.com/2007/06/crivo-de-eratostenes/

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <sys/time.h>

#define NMAX 30000000

int main() {
    clock_t start, end;
    start = clock();

    int i, j, qtd_instrucoes = 0;
    unsigned long valorMaximo, raiz;

    valorMaximo = 30000000;

    // Aloca o vetor dinamicamente para evitar problemas de estouro de pilha
    int *vetor = malloc((NMAX + 1) * sizeof(int));

    // Verifica se a alocação foi bem-sucedida
    if (vetor == NULL) {
        printf("Erro na alocação de memória.\n");
        return 1;
    }

    // Segundo passo
    raiz = sqrt(valorMaximo);

    // Terceiro passo
    for (i = 2; i <= valorMaximo; i++) {
        vetor[i] = i;
        qtd_instrucoes++;
    }

    // Quarto passo
    for (i = 2; i <= raiz; i++) {
        qtd_instrucoes++;
        if (vetor[i] == i) {
            for (j = i + i; j <= valorMaximo; j += i) {
                qtd_instrucoes++;
                vetor[j] = 0; // removendo da lista
            }
        }
    }

    //Percorre o vetor, exibindo os números diferentes de zero que sobraram, ou seja, os primos
    for (i = 2; i <= valorMaximo; i++) {
        qtd_instrucoes++;
        if (vetor[i] == i) {
            printf("%d Eh primo!\n", i);
        }
    }

    printf("Quantidade de instrucoes realizadas: %d\n", qtd_instrucoes);

    // Obtém o valor do tempo atual
    end = clock();

    // Calcula o tempo de execução em milissegundos
    long thicks = (end - start) * 1000 / CLOCKS_PER_SEC;

    // Imprime o tempo de execução
    printf("\nO tempo de execução em milissegundos eh: %lu\n", thicks);

    // Libera a memória alocada
    free(vetor);

    printf("Pressione Enter para fechar o programa...\n");
    getchar(); // Aguarda uma tecla antes de fechar o programa

    return 0;
}
```

ANEXO 3

Algoritmo do crivo de Eratóstenes em Python obtido na internet

https://pt.stackoverflow.com/questions/231555/como-gerar-200-000-primos-o-mais-r%C3%A1pido-poss%C3%ADvel-em-python

```python
def sieve_of_eratosthene(N,qtd_instrucoes):
    #qtd_instrucoes = 0

    # Lista de numeros primos:
    numbers = []

    # Cria-se uma lista referente a todos os inteiros entre 0 e N:
    A = [True] * (N+1)

    # Define os numeros 0 e 1 como nao primos:
    A[0] = A[1] = False

    # Percorra a lista ate encontrar o primeiro numero primo:
    for value, prime in enumerate(A):
        qtd_instrucoes = qtd_instrucoes + 1

        # O numero eh primo?
        if prime:
            qtd_instrucoes = qtd_instrucoes + 1
            # Retorna o numero primo:
            numbers.append(value)

            # Remova da lista todos os multiplos do numero enontrado:
            for i in range(value**2, N+1, value):
                A[i] = False
                qtd_instrucoes = qtd_instrucoes + 1

    #print(f'Quantidade de instruções realizadas: ',qtd_instrucoes)
    return numbers, qtd_instrucoes


#Quantidade de números desejados
n=30000000

qtd_instrucoes = 0

import time
# Obtém o valor do tempo atual
start = time.process_time()

primos = sieve_of_eratosthene(n,qtd_instrucoes)
print(f'Lista numeros primos: ',primos[0])
print(f'Quantidade execuções: ',primos[1])


# Obtém o valor do tempo atual
end = time.process_time()

# Calcula o tempo de execução em segundos
thicks = end - start

# Imprime o tempo de execução
print("O tempo de execução em milissegundos é:", thicks*1000)
```

- **Uma breve explicação do código**

A primeira linha da função, A = [True] * (N+1) cria uma lista de N+1 elementos, todos definidos como True, indicando inicialmente que todos os valores entre 0 e N são números primos. Logo após, é definido os valores 0 e 1 como False, indicando que estes não são números primos. E com um laço de repetição, é percorrido todos os outros valores e sempre que achar um valor primo, retorne-o e elimina da lista, definindo como False, todos os números que são múltiplos desse primo encontrado. Para uma melhor visualização, a lista A seria algo como:

A = [False, False, True, True, False, True, False, ...]

Indicando que 0 não é primo, 1 não é primo, 2 é primo, 3 é primo, 4 não é primo, 5 é primo, 6 não é primo, assim sucessivamente. O que a função enumerate faz é retornar um par de valores onde o primeiro representa o índice na lista e o segundo o valor propriamente dito. Assim, fazendo enumerate(A), seria retornado algo semelhante à:

[(0, False), (1, False), (2, True), (3, True), (4, False), (5, True), (6, False), ...]

E é por isso que no for existem dois valores. O primeiro valor retornado de enumerate é atribuído à value e o segundo à prime, assim, quando prime for verdadeiro, sabemos que value será um número primo.

for value, prime in enumerate(A)...

ANEXO 4

Algoritmo do crivo de Eratóstenes em Python, desenvolvido pelo autor

```python
def primo(list, j,i,n,n_exec):
    for num in list:
        n+=1
        if num%j==0 and num !=j:
            n+=1
            list.remove(num)
    print(f'\nj=',j)
    print(f'lista=',list)
    print(f'Tamanho da lista resultante:',len(list))
    j =list[i+1]
    i+=1
    n_exec.append(n)
    n=0

    if j<=k:
        primo(list,j,i,n,n_exec)


import time
# Obtém o valor do tempo atual
start = time.process_time()


#Quantidade de números
n = 500001

#Cria a lista de números
list = [*range(1, n,1)]
print(f'Lista original:\n',list)
Qtd_numeros = len(list)
print(f'Tamanho lista original=', len(list))

k = int(len(list) ** 0.5)
print(f'Condição de parada (k)=',k)

#remove o número 1 da lista  pois ele não é primo
if list[0] == 1:
        list.remove(list[0])

j=list[0]
i=0
n=0
n_exec =[]

#Chama a função
primo(list,j,i,n,n_exec)

#printa a lista final
print(f'\nlista num_primos:',list)

print(f'\nQtd itens verificados por passada:\n',n_exec)

soma =0
for w in n_exec:
    soma += w
print(f'\nQuantidade números:', Qtd_numeros)
print(f'Soma execuções:', soma)

# Obtém o valor do tempo atual
end = time.process_time()

# Calcula o tempo de execução em segundos
thicks = end - start

# Imprime o tempo de execução
print("O tempo de execução em milissegundos é:", thicks*1000)
```