Universidade Federal da Bahia
Instituto de Computação

Programa de Pós-Graduação em Ciência da Computação

TRABALHO DE IC0004 - ALGORITMOS E GRAFOS – 2024.1 – PROF. GEORGE LIMA

Everton Roberto Zanotelli

Salvador
19 de Maio de 2024

Capítulo 1

INTRODUÇÃO

Esse trabalho está dividido em cinco partes, cada uma representando a respectiva questão, o código dos respectivos experimentos estão no arquivo em anexo.
As sessões começam propondo a solução do problema sem utilizar divisão e conquista, onde é apresentado uma explicação da solução, seguido do seu pseudocódigo, da derivação da complexidade no pior caso, da solução usando divisão e conquista nos mesmos moldes e a analise dos experimentos com base na média de 10 execuções de cada instancia e suas respectivas quantidades de iterações.
As soluções foram desenvolvidas utilizando a linguagem Python e suas bibliotecas em uma maquina Linux usando um processador AMD Ryzen 7 3800X com 64GB de ram.

1 Primeira Questão

Dado um conjunto de n pontos em um plano cartesiano, derive um algoritmo para encontrar neste conjunto o par de pontos mais próximos. O tempo de execução deve ser O(n log n) no pior caso.

1.1 Solução proposta sem divisão e conquista

A abordagem ao problema descrito sem o uso de divisão e conquista foi feita usando o método da busca sequencial, onde cada ponto foi comparado com todos os outros pontos para determinar qual par de pontos estava mais próximo.

A fórmula da função de distância entre dois pontos $P(x_1, y_1)$ e $Q(x_2, y_2)$ é dada por:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Algorithm 1 Comparando todos os pontos para Calcular a Menor Distância

```
function CalcularMenorDistancia(pontos)
    menorDistancia ← ∞
    if len(pontos) > 1 then
        menorDistancia ← distancia(pontos[0], pontos[1])
        for i ← 0 até len(pontos) − 1 do
            for j ← i + 1 até len(pontos) do
                if distancia(pontos[i], pontos[j]) < menorDistancia then
                    menorDistancia ← distancia(pontos[i], pontos[j])
                end if
            end for
        end for
    end if
    return menorDistancia
```

1.2 Derivação de complexidade no pior caso

$$\begin{aligned} T(n) &= O(1) + O(n) + O(n) + O(1) \\ &= O(n^2) + O(1) \\ &= O(n^2) \end{aligned}$$

1.3 Solução proposta usando divisão e conquista

Essa solução usa a estratégia de divisão e conquista para encontrar o par de pontos mais próximos em um conjunto bidimensional. Ele divide o conjunto em duas partes, calcula os menores deltas para cada uma e identifica os pontos próximos à linha divisória. Em seguida, determina o menor delta entre esses pontos e retorna como resultado o menor valor entre esse delta e os menores deltas das duas partes divididas.

Algorithm 2 DeC(pontos, Py)

```
function DeC(pontos, Py)
    if len(pontos) ≤ 3 then
        return ForçaBruta(pontos)
    else
        n ← len(pontos)
        c ← n//2
        esquerda ← pontos[: c]
        direita ← pontos[c :]
        Pye ← []
        Pyd ← []
        metade ← pontos[c][0]
        for e em Py do
            if e[0] < metade then
                Pye.append(e)
            else
                Pyd.append(e)
            end if
        end for
        delta ← min(DeC(esquerda, Pye), DeC(direita, Pyd))
        y ← []
        for e em Py do
            if (|e[0] − metade| ≤ δ) then
                y.append(e)
            end if
        end for
        delta2 ← ∞
        if (len(y) > 0) then
            for i ← 0 até len(y) − 1 do
                j ← i + 1
                while ((j < len(y))e(y[j][1] − y[i][1] ≤ δ)) do
                    if distancia(y[i], y[j]) < delta2 then
                        delta2 ← distancia(y[i], y[j])
                    end if
                    j ← j + 1
                end while
            end for
            return min(δ, delta2)
        else
            return δ
```

1.4 Derivação de complexidade no pior caso

$$\begin{cases} O(1) & \text{se } n \leq 3 \\ 2T(n/2) + O(n) + O(n) + O(n) + O(n \log n) & \text{se } n > 3 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) + O(n) + O(n) + O(n \log n) \\ &= 4T(n/4) + O(n) + O(n) + O(n) + O(n \log n) + O(n) + O(n) + O(n) + O(n \log n) \\ &= O(n \log n) \end{aligned}$$

1.5 Descrição dos experimentos

Os experimentos foram executados 10 vezes para cada tamanho de entrada, os respectivos códigos foram implementados utilizando arquivos auxiliares para fornecer 4 tamanhos de entrada de forma crescente para cada instância no formato x1,y1,x2,y2 , o primeiro com 10 pontos, seguido com 100 pontos, 10000 pontos, e 100000 pontos e calculado o tempo médio de execução para cada entrada e seu respectivo algoritmo depois da decima execução.

Entradas	**Ponto mais próximo**	**Tempo médio de execução médio**	**Iterações**
10	2.8635642126552705	5.841×10^{-5} segundos	90
100	9.91036114377263	1.240×10^{-3} segundos	9900
10000	0.07858753081757049	13.082388639450073 segundos	100M
100000	0.01664331697710184	1358.544866323471 segundos	10B

Table 1: Resultados de execução da solução sem usar Divisão e Conquista

Entradas	**Ponto mais próximo**	**Tempo de execução médio**	**Iterações**
10	2.8635642126552705	5.102×10^{-5} segundos	14
100	9.91036114377263	2.024×10^{-4} segundos	198
10000	0.07858753081757049	2.349×10^{-2} segundos	19.998
100000	0.01664331697710184	2.859×10^{-1} segundos	199.998

Table 2: Resultados de execução da solução usando Divisão e Conquista

Comparando os resultados dos experimentos observamos que a solução que não usa divisão e conquista precisa de vários minutos para solucionar a instancia com 100000 entradas, enquanto que a solução com divisão e conquista resolve em um segundo e com bem menos iterações, evidenciando a vantagem dessa técnica e o custo de uma complexidade O(n^2) para uma O(n log n)

1.6 Bibliografia

MATHWORKS. https://blogs.mathworks.com/cleve/2024/03/28/closest-pair-of-points-proble
Acessado em 15 de Abril de 2024.

GEEKSFORGEEKS. https://www.geeksforgeeks.org/closest-pair-of-points-onlogn-impleme
Acessado em 16 de Abril de 2024.

MIT. 6.838. https://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf. Acessado em 16 de Abril de 2024.

GEEKSFORGEEKS. https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-a
Acessado em 17 de Abril de 2024.

YOUTUBE. IDEAR7. https://www.youtube.com/watch?v=6u_hWxbOc7E. Acessado em 18 de Abril de 2024.

USP. IME. https://www.ime.usp.br/~cris/aulas/10_1_6711/slides/aula2pmp.pdf. Acessado em 18 de Abril de 2024.

YOUTUBE. ALGORITHMS BY SHARMA THANKACHAN. https://www.youtube.com/watch?v=kCLGVat2SHk. Acessado em 16 de Abril de 2024.

2 Segunda Questão

Sejam X e Y dois números inteiros de n dígitos. Apos pesquisar sobre o assunto, desenvolva um algoritmo que encontre a multiplicação de X e Y em menos que O(n2) passos. Encontre a complexidade do algoritmo desenvolvido.

2.1 Solução proposta sem divisão e conquista

A solução proposta sem utilizar divisão e conquista é uma multiplicação usual entre números conforme aprendemos no ensino médio que tem seu consumo de tempo equivalente a O($\mathbf{n^2}$)

Algorithm 3 Multiplicação tradicional

```
function MULTIPLICAR(num1, num2)
    produto ← num1 · num2
    return produto
end function=0
```

2.2 Derivação de complexidade no pior caso

$$= O(n) \times O(n)$$
$$= O(n^2)$$

O pior caso dessa solução é igual ao caso médio pois sempre vamos multiplicar cada numero por todos os outros, mantendo a complexidade em O($\mathbf{n^2}$) sempre que n for maior que zero.

2.3 Solução proposta usando divisão e conquista

Para uma solução com menos de 0($\mathbf{n^2}$) passos usando divisão e conquista, podemos usar o algoritmo de Karatsuba descrito no pseudocódigo abaixo:

Algorithm 4 KARATSUBA(X, Y, n)

if $n \leq 3$ **then**
    return $X \cdot Y$
end if
$q \leftarrow \lceil n/2 \rceil$
$A \leftarrow X[q + 1..n]$
$B \leftarrow X[1..q]$
$C \leftarrow Y[q + 1..n]$
$D \leftarrow Y[1..q]$
$E \leftarrow$ KARATSUBA(A, C, q)
$F \leftarrow$ KARATSUBA(B, D, q)
$G \leftarrow$ KARATSUBA$(A + B, C + D, q + 1)$
$R \leftarrow G - F - E$
$R \leftarrow E \times 10^n + H \times 10^{\lceil n/2 \rceil} + F$
return R

2.4 Derivação de complexidade no pior caso

$$\begin{aligned} T(n) &= T\left\lfloor \frac{n}{2} \right\rfloor + T\left\lceil \frac{n}{2} \right\rceil + T\left\lceil \frac{n}{2} \right\rceil + 1 + O(5n + 2) \\ &= T\left\lfloor \frac{n}{2} \right\rfloor + T\left\lceil \frac{n}{2} \right\rceil + T\left\lceil \frac{n}{2} \right\rceil + O(n) \\ &= 3T\left(\frac{n}{2}\right) + n \\ &= O(n^{\log 3}) \\ &= O(n^{1.58}) \end{aligned}$$

2.5 Descrição dos experimentos

Para essa questão ficou ficou difícil demonstrar de forma pratica a superioridade assintótica do algoritmo de Karatsuba pois para as entradas máximas que consegui simular(4300 dígitos), o algoritmo de multiplicação tradicional é bem mais eficiente.

Dígitos	Karatsuba	Multiplicação Tradicional
2150	0.24380874633789062	0.0006077289581298828
1075	0.18170666694641113	0.00018525123596191406
538	0.026221275329589844	7.700920104980469e-05
270	0.009067535400390625	5.125999450683594e-05
135	0.002771139144897461	4.172325134277344e-05

Table 3: Tempo de execução em segundos

Como a evidencia de superioridade do algoritmo de Karatsuba só aparece depois de algumas centenas de dígitos, fiz uma suposição de que o algoritmo levaria 1 minuto para multiplicar dois números de n dígitos cada, calculei quanto ficaria esse tempo para as entradas 10n, 100n e 1000n e fiz uma aproximação para o algoritmo de multiplicação usual.

Número de dígitos	Algoritmo de Karatsuba	Multiplicação usual
10n	$T(10n) \approx 32$ minutos	$T(10n) = 100 \cdot n^2$ minutos
100n	$T(100n) \approx 316$ minutos	$T(100n) = 10000 \cdot n^2$ minutos
1000n	$T(1000n) \approx 3162$ minutos	$T(1000n) = 1000000 \cdot n^2$ minutos

Table 4: Comparação dos tempos de execução dos algoritmos

Tamanho dos Números	**Karatsuba (d:h:m:s)**	**Multiplicação Usual (d:h:m:s)**
10n	0:00:32:00	0:00:01:40
100n	0:05:18:00	0:27:46:40
1000n	2:05:30:00	11:37:46:40

Table 5: Comparação com o tempo convertido.

Tamanho dos Números	Karatsuba ($n^{1.585}$)	Multiplicação Tradicional (n^2)
n	$n^{1.585}$	n^2
$10n$	$38.32 \cdot n^{1.585}$	$100n^2$
$100n$	$1584.89 \cdot n^{1.585}$	$10000n^2$
$1000n$	$65513.03 \cdot n^{1.585}$	$1000000n^2$

Table 6: Comparação do número de iterações

2.6 Bibliografia

USP. IME. https://www.ime.usp.br/~cris/aulas/10_1_6711/slides/aula6.pdf. Acessado em 22 de Abril de 2024.

CARNEGIE MELLON. https://www.cs.cmu.edu/~cburch/251/karat/. Acessado em 24 de Abril de 2024.

USP. IME. https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/karatsuba.html. Acessado em 25 de Abril de 2024.

STACKOVERFLOW. https://stackoverflow.com/questions/42324419/karatsuba-multiplication-implementation. Acessado em 16 de Abril de 2024.

CODEANDGADGETS. https://www.codeandgadgets.com/karatsuba-multiplication-python/. Acessado em 26 de Abril de 2024.

3 Problema numero 3

Seja X um vetor de n inteiros distintos dispostos em ordem crescente. Desenvolva um algoritmo para encontrar algum i tal que Xi = i. O tempo de execução do algoritmo deve ser O(log n).

3.1 Solução proposta sem divisão e conquista

A solução proposta sem utilizar divisão e conquista consiste em uma busca linear por todo o vetor procurando pelo i definido conforme pseudocódigo abaixo:

Algorithm 5 EncontreIndice(X, n)

```
for i ← 0 to n − 1 do
    if X[i] = i then
        return i
    end if
end for
```

3.2 Derivação de complexidade no pior caso

$$\begin{aligned} T(n) &= O(n) + O(1) + O(1) + O(1) \\ &= n + 3 \\ &= O(n) \end{aligned}$$

3.3 Solução proposta usando divisão e conquista

A solução proposta utilizando divisão e conquista é uma busca binaria que vai dividindo o vetor até encontrar o i definido conforme pseudocódigo abaixo:

Algorithm 6 EncontreIndice(X, n)

$esquerda \leftarrow 0$
$direita \leftarrow n - 1$
while $esquerda \leq direita$ **do**
    $meio \leftarrow \left\lfloor \frac{esquerda+direita}{2} \right\rfloor$
    if $X[meio] == meio$ **then**
        return $meio$
    else if $X[meio] > meio$ **then**
        $direita \leftarrow meio - 1$
    else
        $esquerda \leftarrow meio + 1$
    end if
end while

3.4 Derivação de complexidade no pior caso

$$\begin{aligned}
T(n) &= O(1) + O(1) \cdot O\log(n) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) \\
&= 2 \cdot \log(n) + 8 \\
&= O(\log(n))
\end{aligned}$$

3.5 Descrição dos experimentos

Nesse experimento eu criei 3 testes usando respectivamente 1000, 10000, 100000 entradas e gerei os gráficos do tempo médio de execução para cada entrada e seu respectivo algoritmo depois da decima execução. Podemos observar que os tempos em todos os casos demonstram que a solução usando divisão e conquista executa em menos tempo do que a solução sem usar divisão e conquista conforme dado pela complexidade assintótica dos algoritmos.

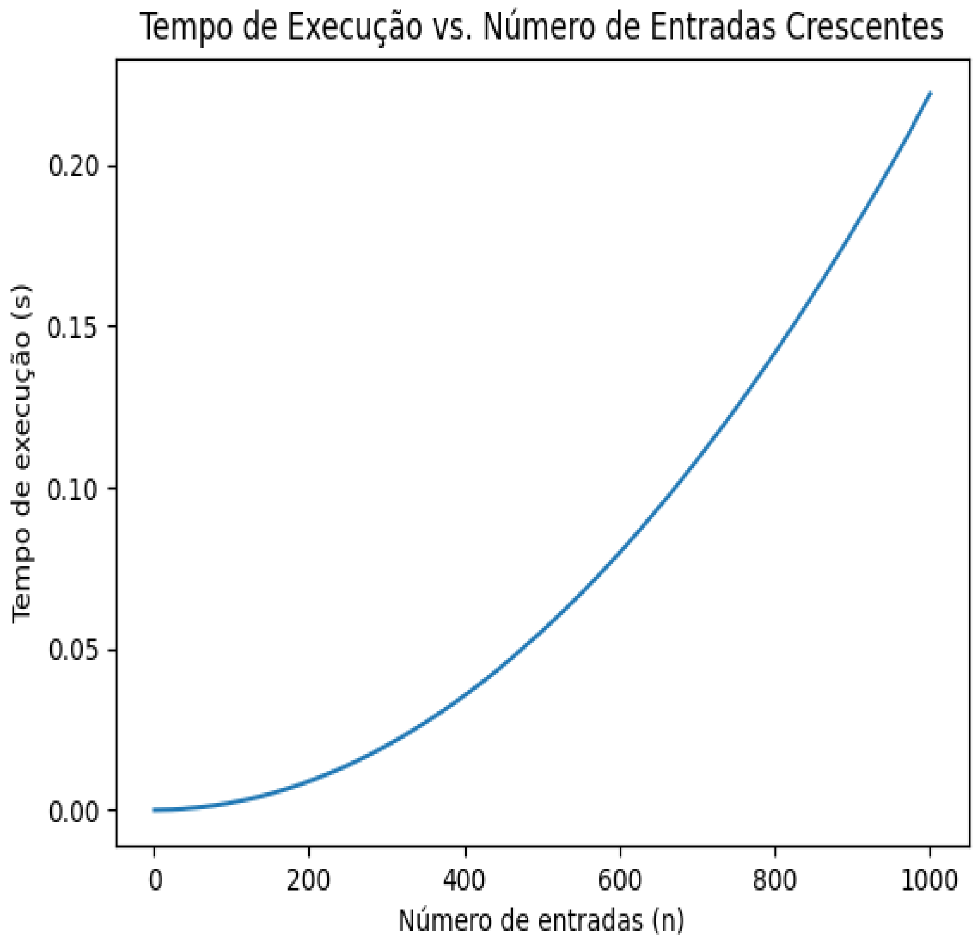

Figura 1: Experimento com 1000 entradas sem DeC

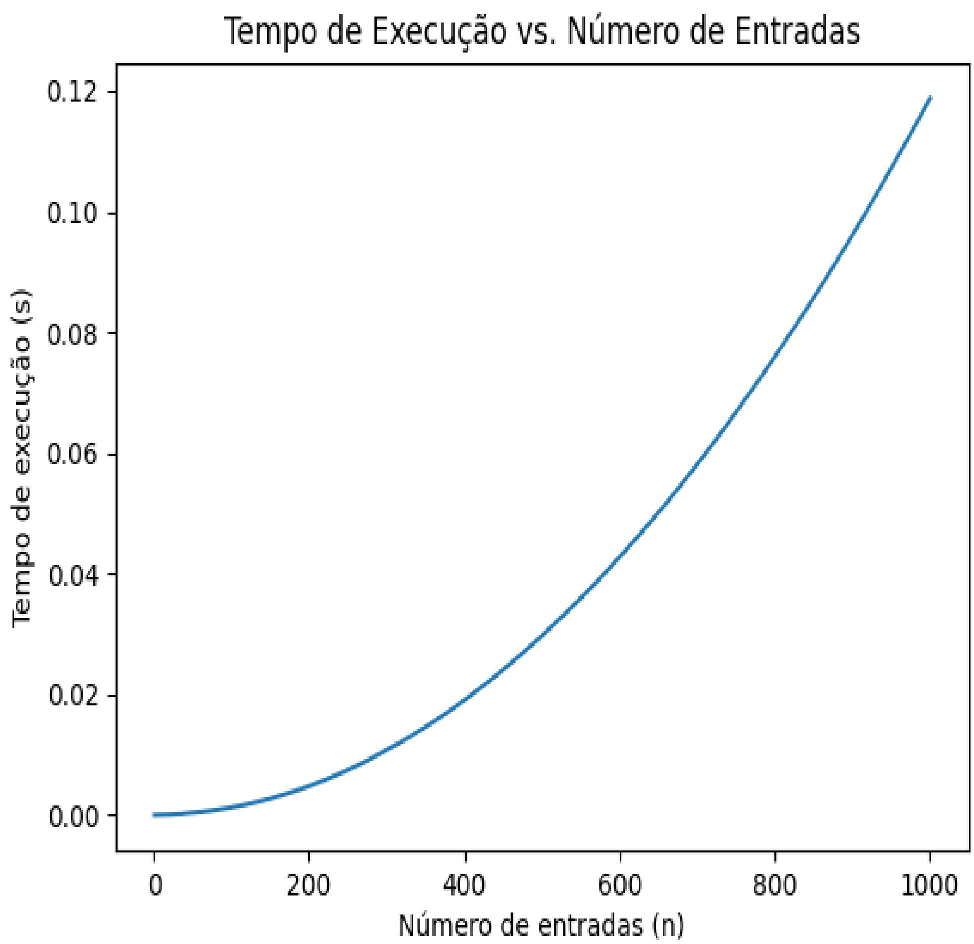

Figura 2: Experimento com 1000 entradas com DeC

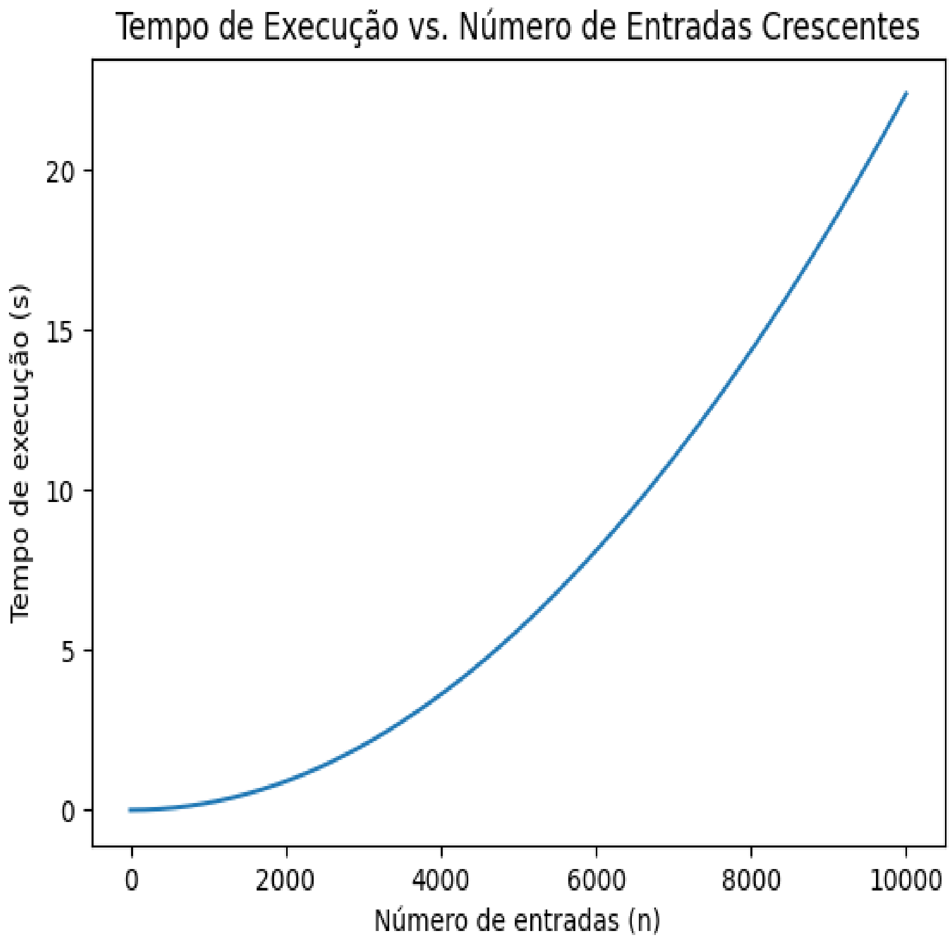

Figura 3: Experimento com 10000 entradas sem DeC

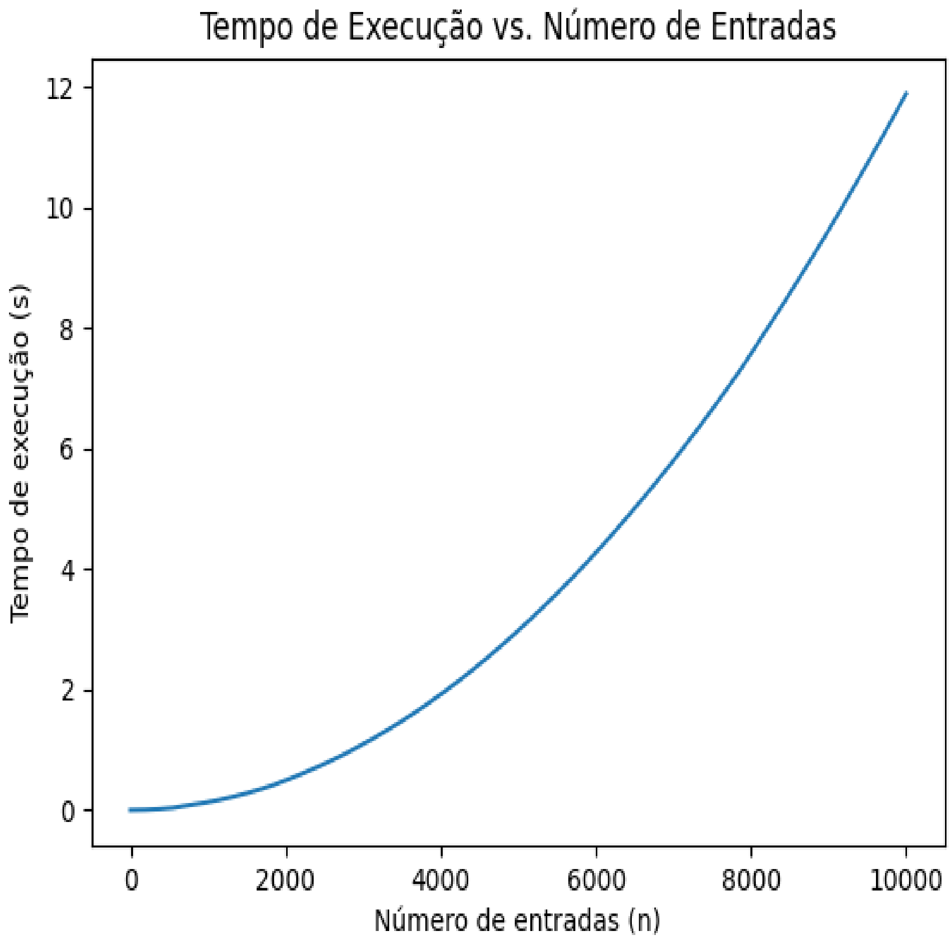

Figura 4: Experimento com 10000 entradas com DeC

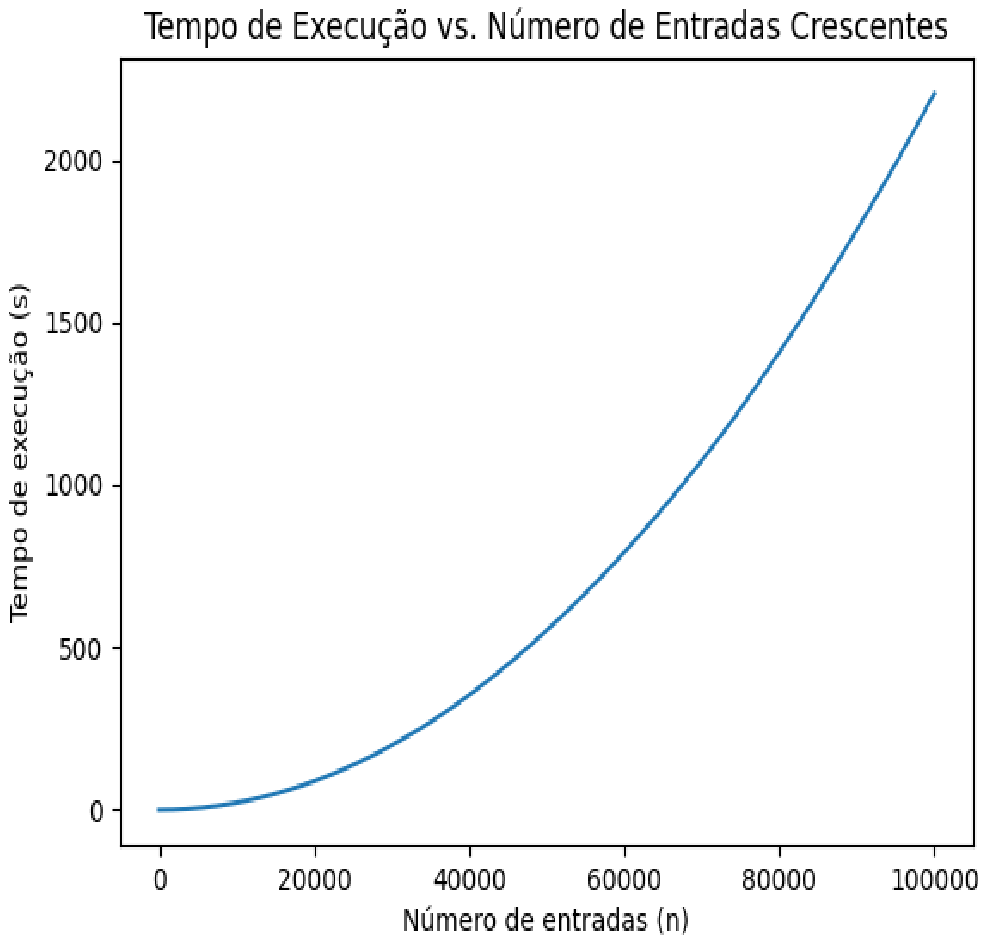

Figura 5: Experimento com 100000 entradas sem DeC

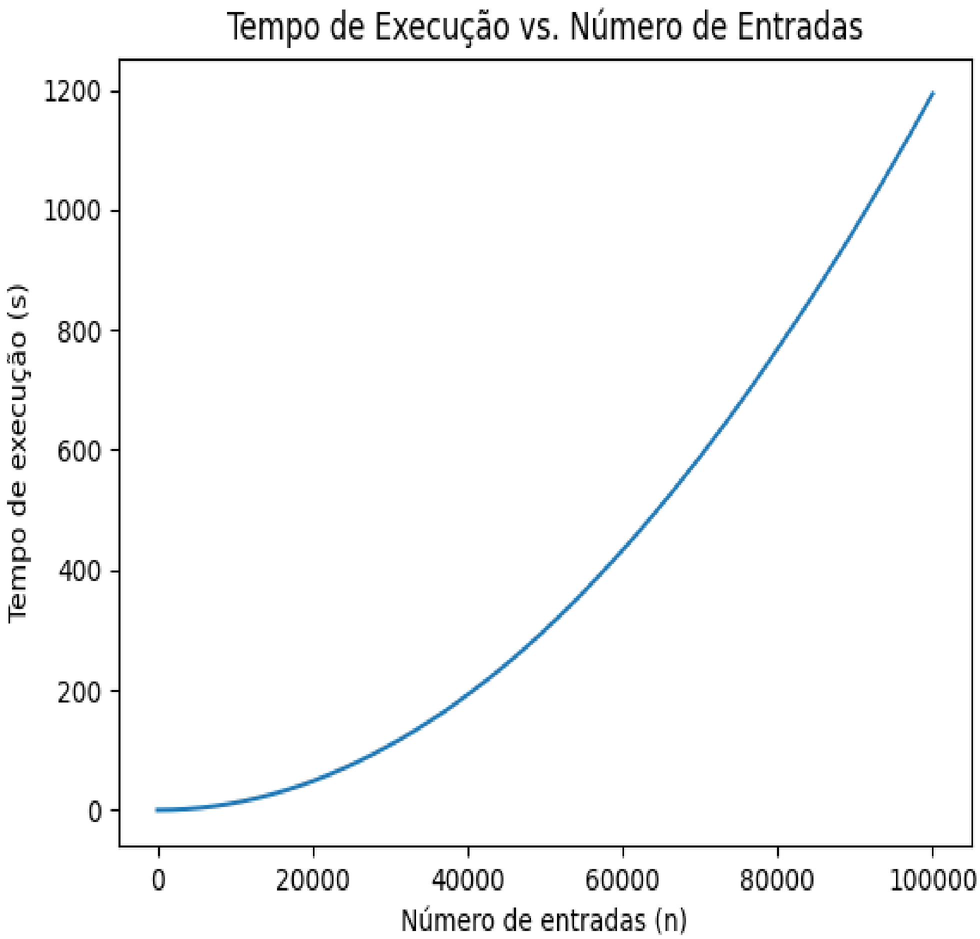

Figura 6: Experimento com 100000 entradas com DeC

Em termos de quantidade de iterações feitas pelos algoritmos o que utiliza a técnica de divisão e conquista consegue resolver o problema com menos iterações.

Tamanho da Entrada	**Busca Binária**	**Busca Sequencial**
1000	10	1000
10000	14	10000
100000	17	100000

Tabela 7: Comparação do número de iterações

3.6 Bibliografia

USP. IME. https://www.ime.usp.br/~cris/aulas/10_1_6711/slides/aula1.pdf. Acessado em 30 de Abril de 2024.

USP. IME. https://www.ime.usp.br/~cris/aulas/10_1_6711/marco.html. Acessado em 30 de Abril de 2024.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. The MIT Press, 2nd edition, 2003.

HARVARD. CS50. https://cs50.harvard.edu/college/2023/fall/weeks/3/. Acessado em 30 de Abril de 2024.

BLOGCYBERINI. https://www.blogcyberini.com/2017/09/busca-binaria.html. Acessado em 02 de Maio de 2024.

KHANACADEMY. https://pt.khanacademy.org/computing/computer-science/algorithms/binary-search/a/running-time-of-binary-search. Acessado em 2 de Maio de 2024.

YOUTUBE. CARLA QUE DISSE. https://www.youtube.com/watch?v=JpbGKB6LF2g. Acessado em 02 de Maio de 2024.

4 Quarto exercicio

Sejam X e Y dois vetores ordenados de tamanho n e m, respectivamente. Desenvolva um algoritmo para encontrar o k-esimo elemento de XuY. O algoritmo deve executar em O(log m + log n) unidades de tempo.

4.1 Solução proposta sem divisão e conquista

A solução proposta sem utilizar divisão e conquista consiste em buscar linearmente por todo o vetor criado da união entre X e Y pelo k-esimo numero definido conforme pseudocódigo abaixo:

```
function ACHAR_KESIMO(X, Y, k)
    i ← 0
    j ← 0
    vetord ← []
    while i < len(X) and j < len(Y) do
        if X[i] < Y[j] then
            APPEND(vetord, X[i])
            i ← i + 1
        else
            APPEND(vetord, Y[j])
            j ← j + 1
        end if
    end while
    while i < len(X) do
        APPEND(vetord, X[i])
        i ← i + 1
    end while
    while j < len(Y) do
        APPEND(vetord, Y[j])
        j ← j + 1
    end while
    return vetord[k − 1]
end function
```

4.2 Derivação de complexidade no pior caso

$$\begin{aligned} T(n) &= O(n+m) + O(n+m) + O(1) \\ &= 2O(n+m) + O(1) \\ &= O(n+m) \end{aligned}$$

4.3 Solução proposta usando divisão e conquista

A solução proposta utilizando divisão e conquista funciona de forma recursiva, dividindo os vetores em sub vetores menores, comparando seus elementos e descartando os vetores com elementos maiores que o elemento procurado.

Algorithm 7 achar_kesimo(X, Y, k)

```
function KESIMOMENOR(A, B, k)
    ladoA, ladoB ← len(A), len(B)
    if ladoA > ladoB then
        return KESIMOMENOR(B, A, k)
    end if
    if ladoA == 0 then
        return B[k − 1]
    end if
    if k == 1 then
        return min(A[0], B[0])
    end if
    i ← min(ladoA, k//2)
    j ← min(ladoB, k//2)
    if A[i − 1] > B[j − 1] then
        return KESIMOMENOR(A, B[j :], k − j)
    else
        return KESIMOMENOR(A[i :], B, k − i)
    end if
end function
return KESIMOMENOR(X, Y, k)
```

4.4 Derivação de complexidade no pior caso

$$
\begin{aligned}
T(n) &= O(1)O(log(n) + O(log(m) \\
&= 2O(n+m) + O(1) \\
&= O(log(n) + log(m)
\end{aligned}
$$

4.5 Descrição dos experimentos

Nesse experimento eu criei 3 testes usando respectivamente 1000, 10000, 100000 entradas e gerei os gráficos do tempo médio de execução para cada entrada e seu respectivo algoritmo depois da decima execução. Podemos observar que os tempos em todos os casos demonstram que a solução usando divisão e conquista executa em menos tempo mesmo nos menores casos.

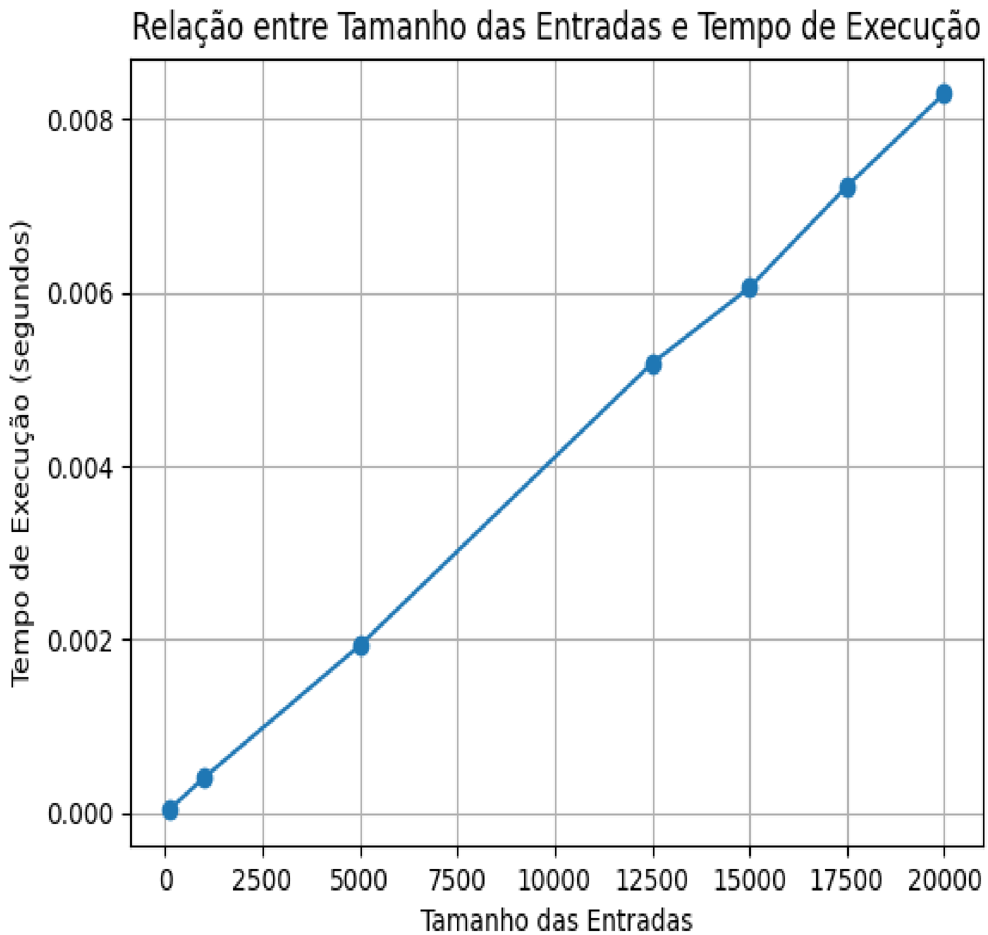

Figura 7: Experimento com 20000 entradas com DeC

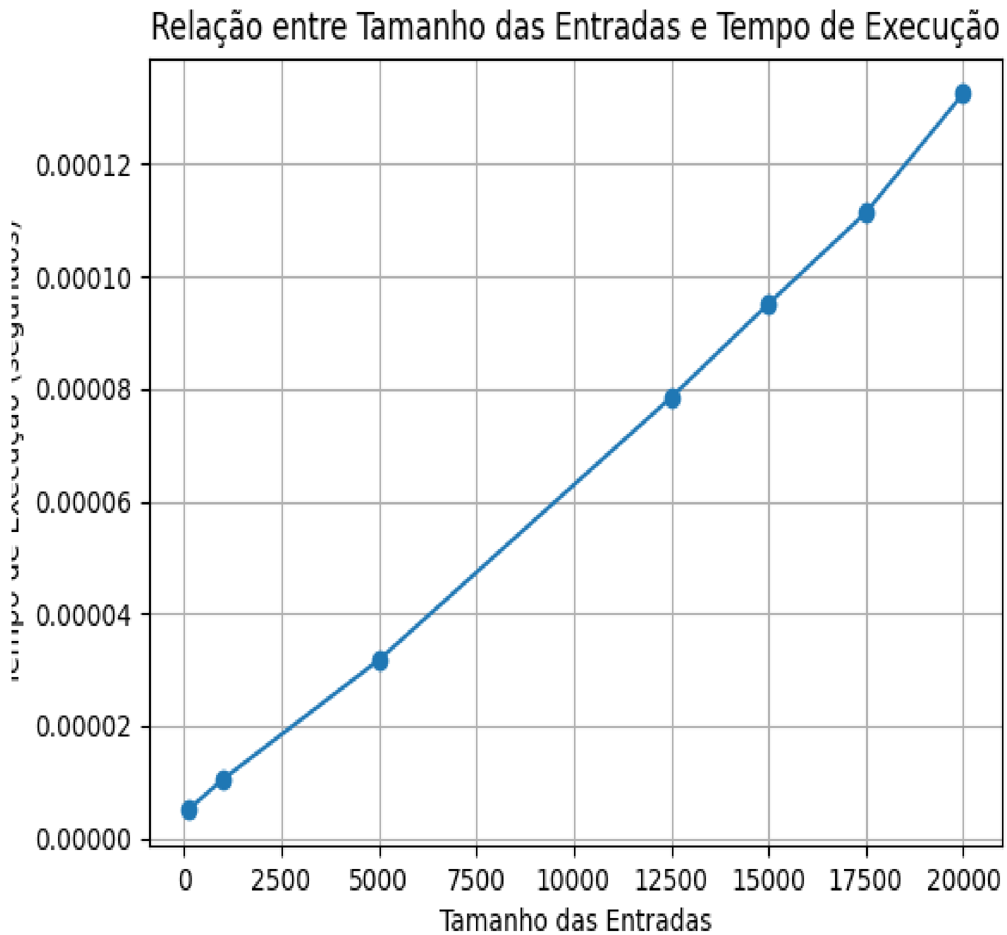

Figura 8: Experimento com 20000 entradas com DeC

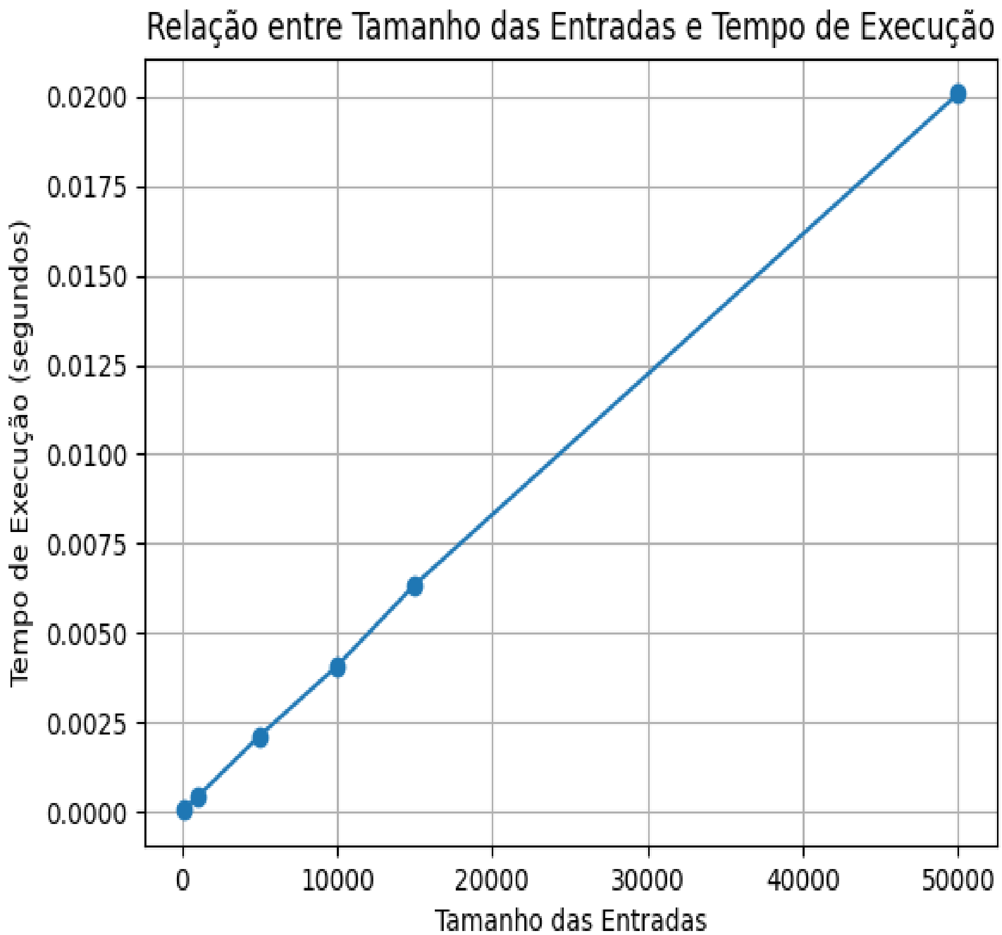

Figura 9: Experimento com 50000 entradas com DeC

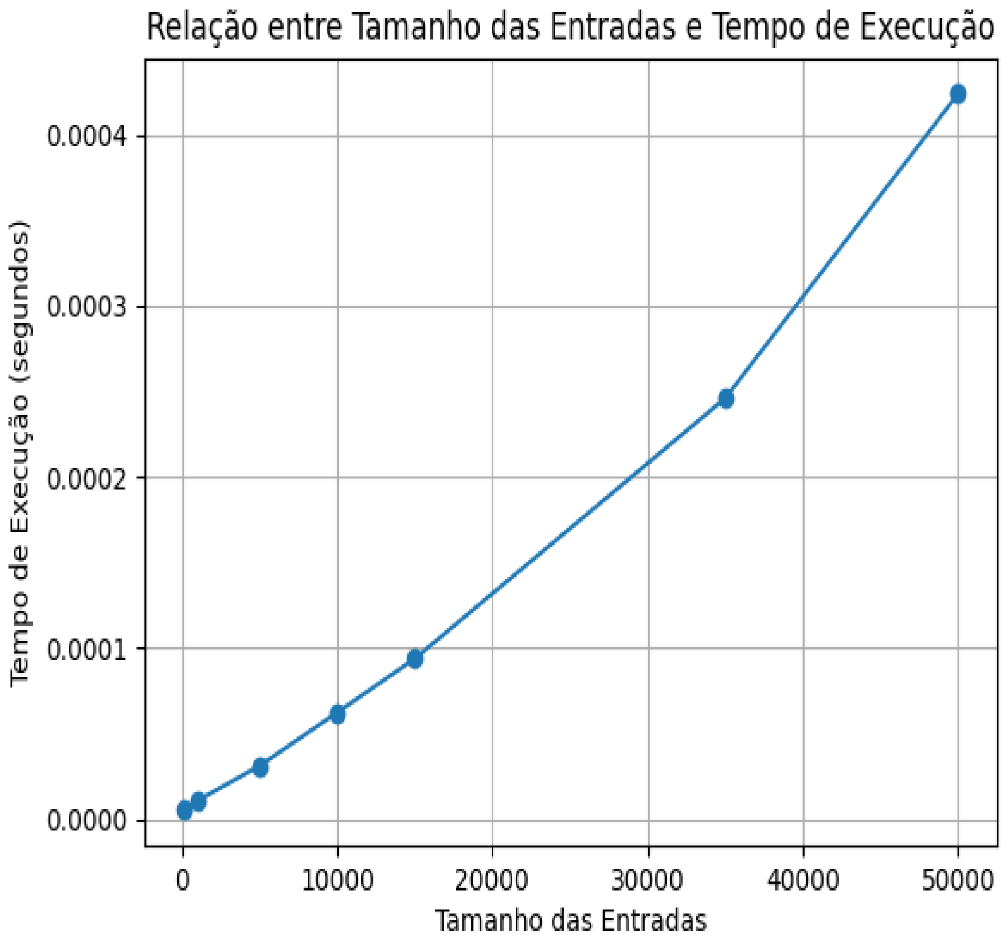

Figura 10: Experimento com 50000 entradas com DeC

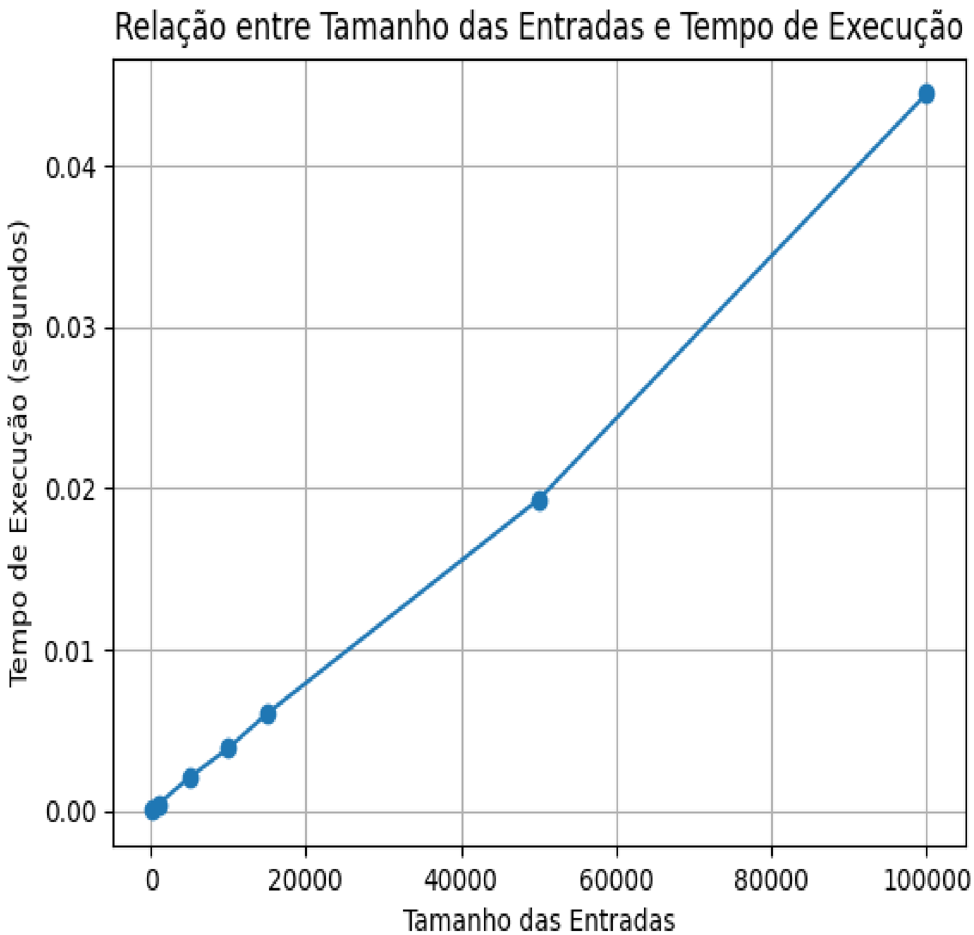

Figura 11: Experimento com 100000 entradas com DeC

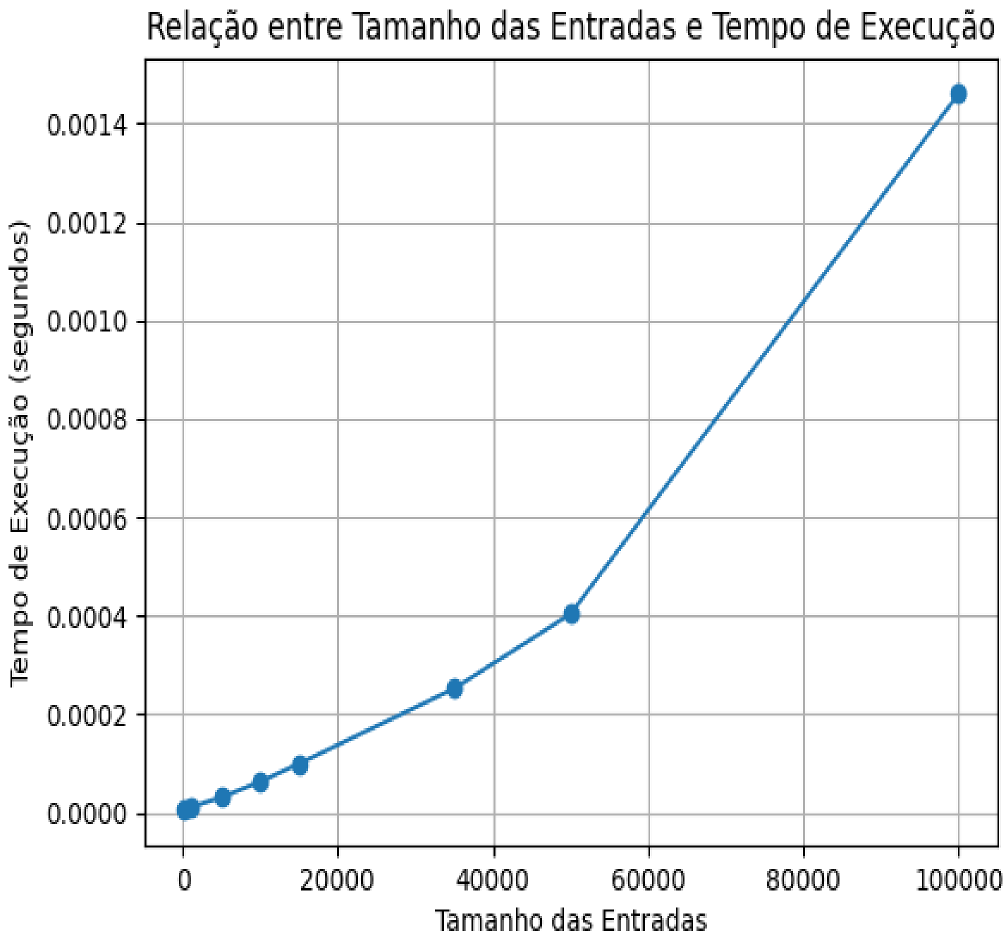

Figura 12: Experimento com 100000 entradas com DeC

Tamanho da Entrada	**Com DeC**	**Sem DeC**
20000	14	40000
50000	15	100000
100000	16	200000

Tabela 8: Comparação do número de iterações

4.6 Bibliografia

USP. IME. https://www.ime.usp.br/~cris/aulas/10_1_6711/slides/aula7.pdf. Acessado em 10 de Maio de 2024.

USP. IME. https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/median.html. Acessado em 10 de Maio de 2024.

BAELDUNG. CS. https://www.baeldung.com/cs/kth-smallest-element-in-sorted-arrays. Acessado em 11 de Maio de 2024.

BOWDOIN. CS. https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/2020fall/Lectures/L5-selection.pdf. Acessado em 12 de Maio de 2024.

MEDIUM. https://medium.com/@sreeku.ralla/k%E1%B5%97%CA%B0-smallest-element-in-an-array- Acessado em 14 de Maio de 2024.

5 Quinto exercício

Suponha uma pesquisa de opinião publica onde os entrevistados respondem a seguinte pergunta: qual a marca de produto mais popular dentre todas que você conhece? As respostas de n entrevistados são armazenadas num vetor V. Elabore um algoritmo de tempo de execução linear para identificar se existe uma marca citada por mais da metade dos entrevistados. O algoritmo não deve alocar memoria extra além da necessária para armazenar V.

5.1 Solução proposta sem divisão e conquista

A solução proposta sem utilizar divisão e conquista consiste em buscar linearmente por todo o vetor V e verificar se a marca tem mais da metade dos votos:

Algorithm 8 Pesquisa

```
entrevistado ← 0
contador ← 0
for numero ∈ marca do
    if contador = 0 then
        entrevistado ← numero
    end if
    contador ← contador + 1
end for
contador ← sum(somavoto)
if contador > len(marca)//2 then
    return entrevistado
end if
return −1
```

5.2 Derivação de complexidade no pior caso

$$
\begin{aligned}
T(n) &= O(n) + O(n) + O(1) \\
&= O(n)
\end{aligned}
$$

5.3 Solução proposta usando divisão e conquista

Para solução que use divisão e conquista implementei o algoritmo de BOYER-MOORE que tem tempo de execução 0(n) conforme solicitado pelo enunciado.

Algorithm 9 Pesquisa

```
function PESQUISA(marca)
    vetor ← len(marca)
    numero ← 0
    contador ← 0
    for e in marca do
        if contador == 0 then
            numero ← e
        end if
        if e == numero then
            contador ← contador + 1
        else
            contador ← contador − 1
        end if
    end for
    contador ← 0
    for e in marca do
        if e == numero then
            contador ← contador + 1
        end if
    end for
    if contador > vetor ÷ 2 then
        print(numero)
        return numero
    end if
    return −1
end function
```

5.4 Derivação de complexidade no pior caso

$$
\begin{aligned}
T(n) &= O(n) + O(n) + O(1) \\
&= O(n)
\end{aligned}
$$

5.5 Descrição dos experimentos

Nesse experimento eu criei 3 testes usando respectivamente 1000, 10000, 100000 entradas com 5 opções de marcas para a pesquisa, gerei os gráficos do tempo médio de execução para cada conjunto de entradas e seu respectivo algoritmo depois da decima amostragem de execução. Os tempos de execução são semelhantes visto que os 2 algoritmos tem complexidade O(n). Em quantidade de iterações os algoritmos aparentemente tem o mesmo numero de iterações.

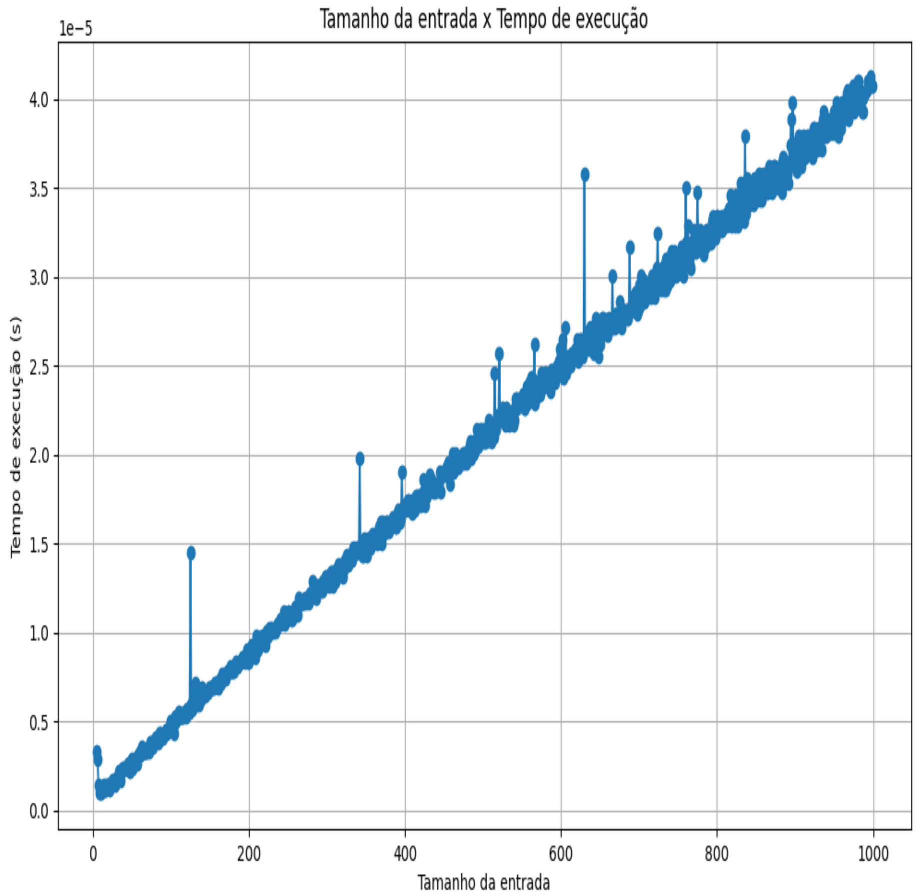

Figura 13: Experimento com 1000 entradas sem DeC

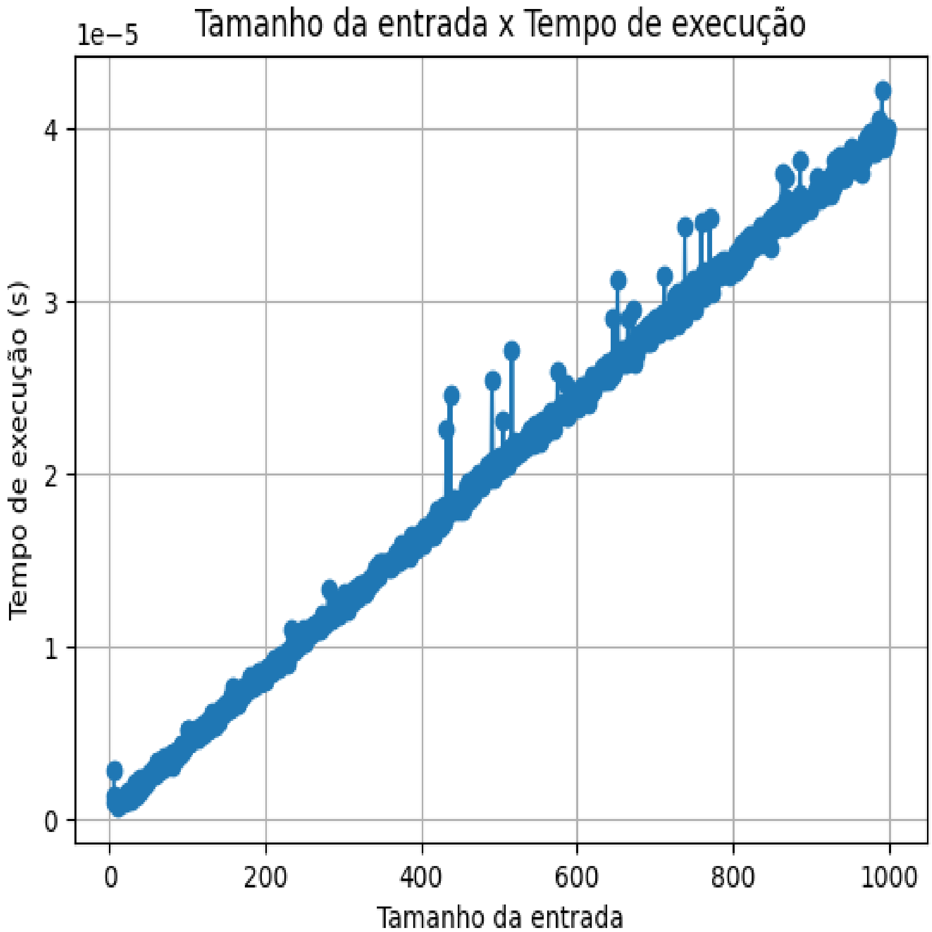

Figura 14: Experimento com 1000 entradas com DeC

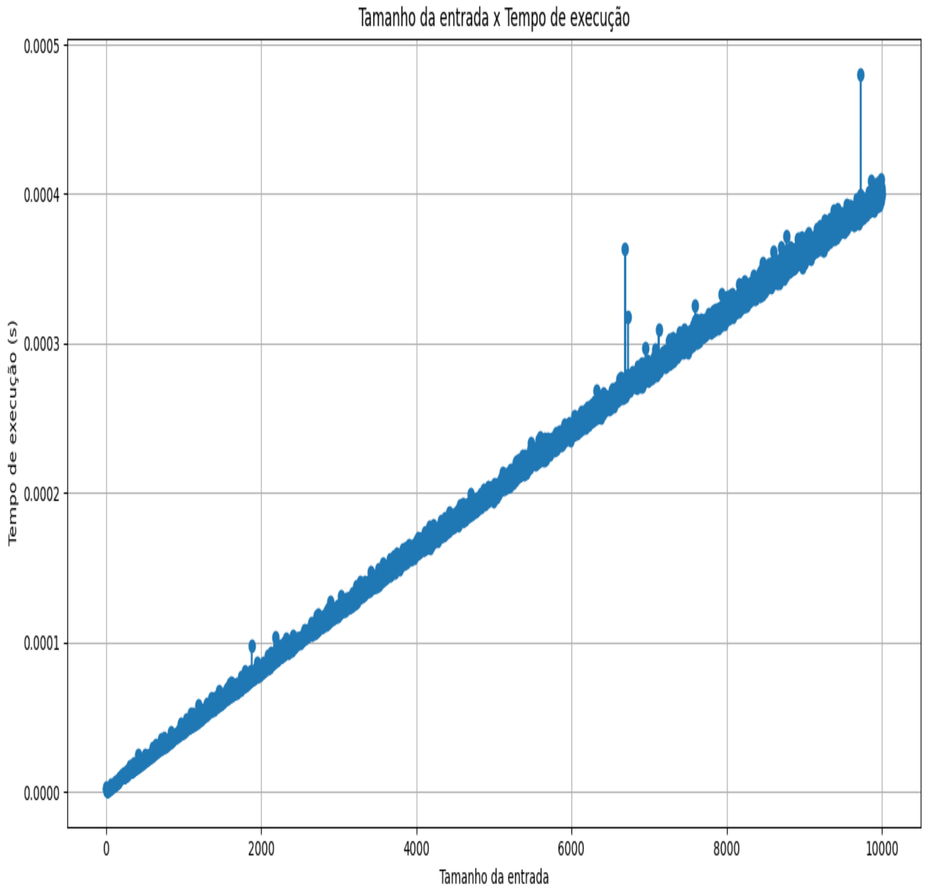

Figura 15: Experimento com 10000 entradas sem DeC

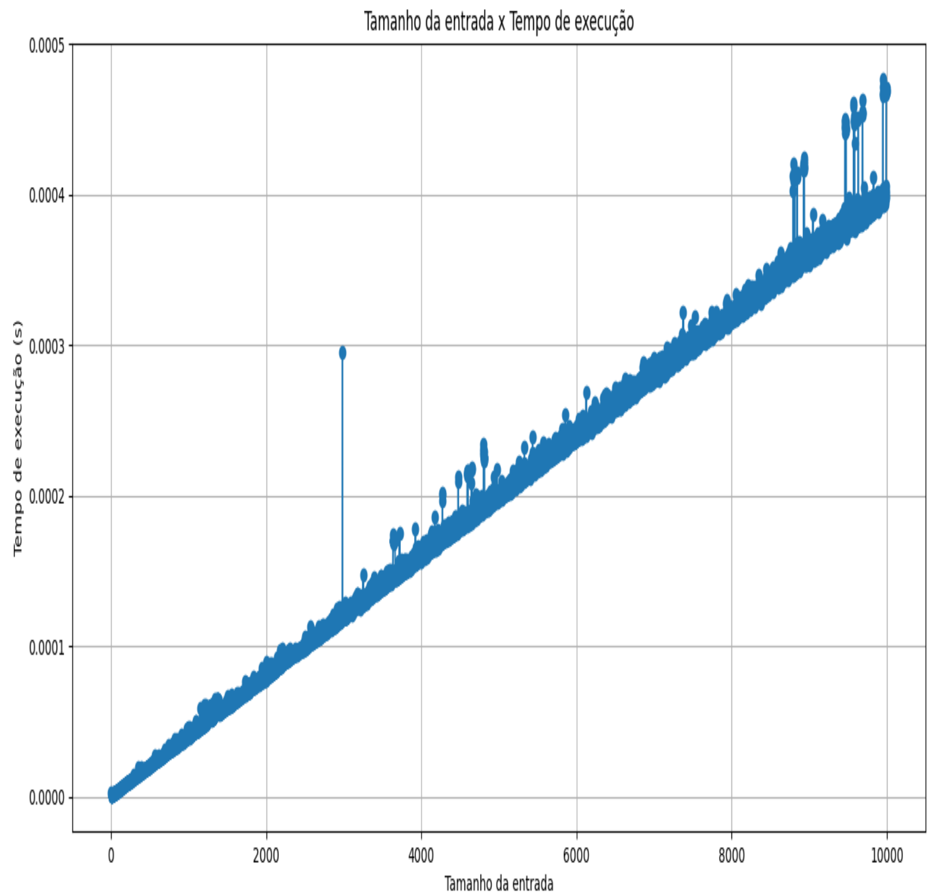

Figura 16: Experimento com 10000 entradas com DeC

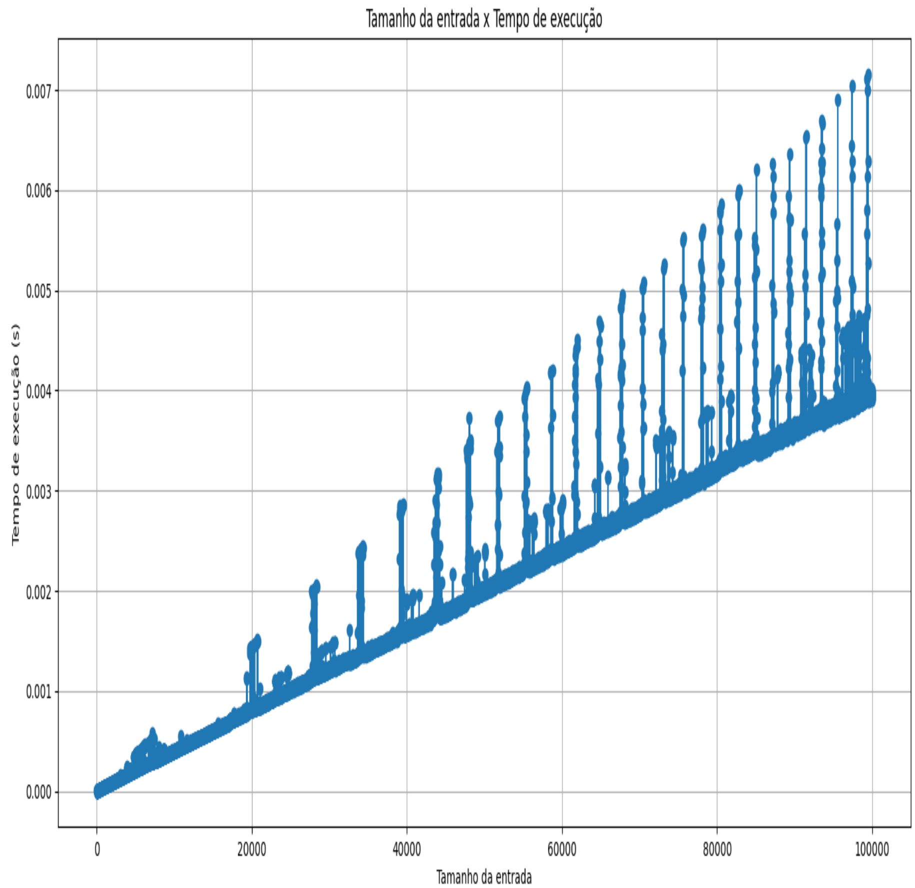

Figura 17: Experimento com 100000 entradas sem DeC

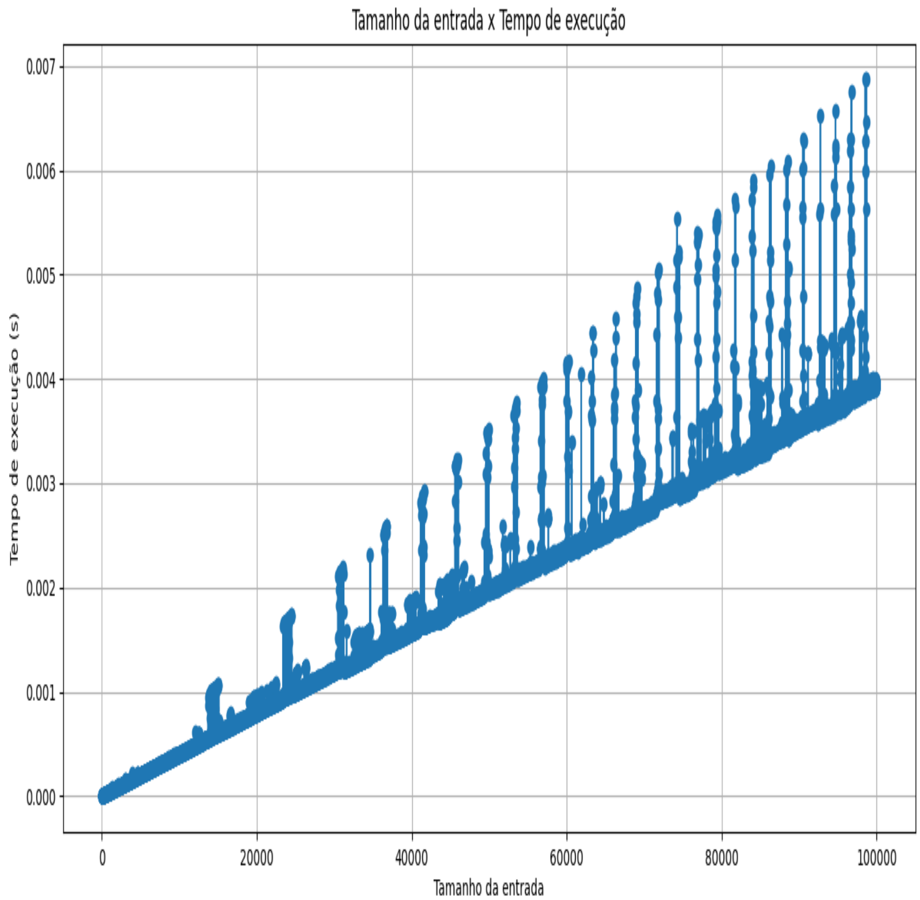

Figura 18: Experimento com 100000 entradas com DeC

Tamanho de Entrada	**Sem DeC**	**Com DeC**
1000	3000	3000
10000	30000	30000
100000	300000	300000

Tabela 9: Comparação do número de iterações

5.6 Bibliografia

DEVTO. https://dev.to/alisabaj/the-boyer-moore-majority-vote-algorithm-finding-the-majorit Acessado em 15 de Maio de 2024.

TOPCODER. https://www.topcoder.com/thrive/articles/boyer-moore-majority-vote-algorithm Acessado em 15 de Maio de 2024.

TECHIEDELIGHT. https://www.techiedelight.com/find-majority-element-in-an-array-boyer-m Acessado em 15 de Maio de 2024.

DEVTO. https://dev.to/tishaag098/leetcode-majority-element-boyer-moore-majority-voting- Acessado em 16 de Maio de 2024.

GEEKSFORGEEKS. https://www.geeksforgeeks.org/boyer-moore-majority-voting-algorithm/. Acessado em 16 de Maio de 2024.

YOUTUBE. CARLA QUE DISSE. https://www.youtube.com/watch?v=JpbGKB6LF2g. Acessado em 16 de Maio de 2024.

USP. IME. https://www.ime.usp.br/~coelho/mac0323-2019/provinhas/p12.pdf. Acessado em 16 de Maio de 2024.