Relatório Projeto 2 AED 2023-2024

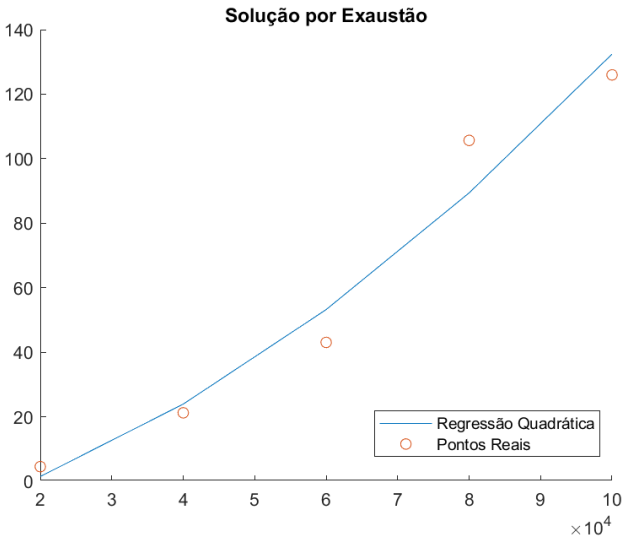
Nome: Arthur Sophiatti Nº Estudante: 2022115599
PL (inscrição): PL2

A fim de comparar o tempo de execução, foram executados os três algoritmos 10 (dez) vezes e a mediana dos tempos obtidos foi utilizado para fins de análise. Sobre as amostras, foram usados arrays (listas) de tamanhos (N): 20.000, 40.000, 60.000, 80.000, 100.000; todos foram preenchidos com números de [1, N-1] e após ser embaralhado (shuffle) foi adicionado um número aleatório desse intervalo em uma posição aleatória. Segue resultados obtidos:

Tabela para as 3 soluções

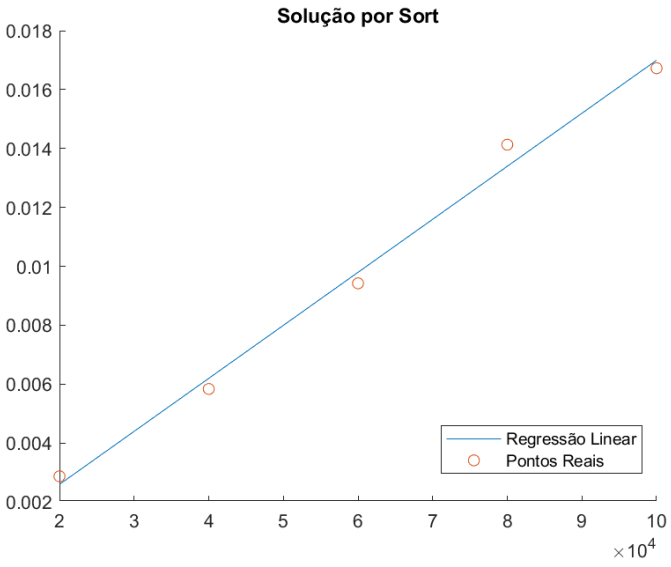
Tamano do Array\ Algoritmo	Exaustão	Sort	Soma
20 000	4.530172657966614	0.0028690338134765626	0.0006046533584594726
40 000	21.221055650711058	0.005833101272583008	0.0013065814971923828
60 000	43.0513578414917	0.009419965744018554	0.002679324150085449
80 000	105.66545600891114	0.014128947257995605	0.0025393486022949217
100 000	125.95995230674744	0.016724252700805665	0.004190278053283691

Análise da solução pelo método de exaustão



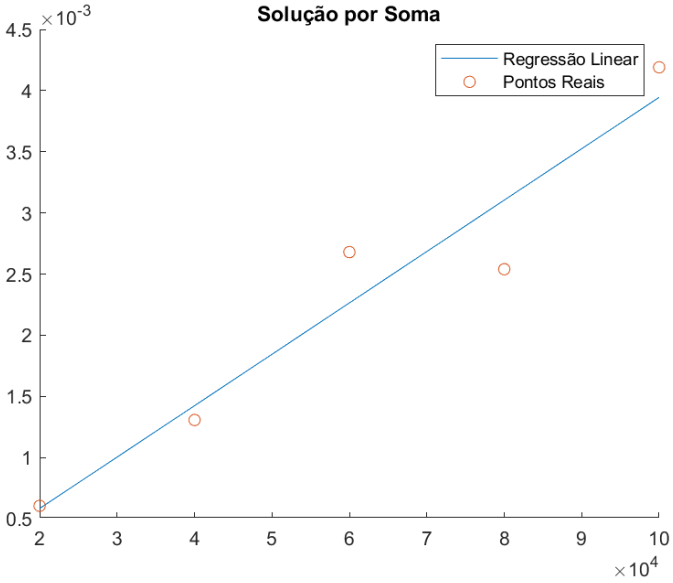
Este método possui uma complexidade quadrática, pois há dois ciclos que percorrem a lista inteira comparando os elementos, isso resultaria em complexidade $O(N^2)$. Porém foi feira uma pequena otimização no algoritmo em que o ciclo mais interno percorre a lista apenas de i até o final.

Análise da solução pelo método do sort



Com a solução por meio de sort em que primeiro se ordena a lista e depois comparamos o elemento de índice i com o seu sucessor ($índice = i + 1$) vemos uma complexidade linear (N), ou seja, há uma relação direta entre o tempo e o tamanho da lista que estamos trabalhando. Após a analise de complexidade, em que a parte mais custosa do algoritmo é propriamente o sort, porém como não há como analisarmos a função essa função nativa internamente, apenas ignora-se. Sendo assim, o método possui complexidade $O(N)$.

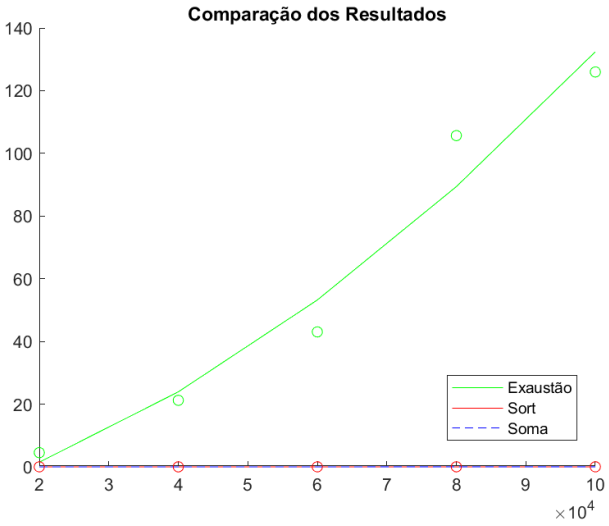
Análise da solução pelo método de soma de uma Progressão Aritmética



O método da soma de uma PA é só possível pois sabe-se o conteúdo da lista, e confere os critérios para a aplicação do método matemático. Neste algoritmo usa-se a fórmula de todos os termos de uma PA de tamanho N e se compara com a soma real dos elementos presentes na lista. Como há a necessidade de percorrer a lista inteira para somar os seus elementos, este método possui complexidade $O(N)$.

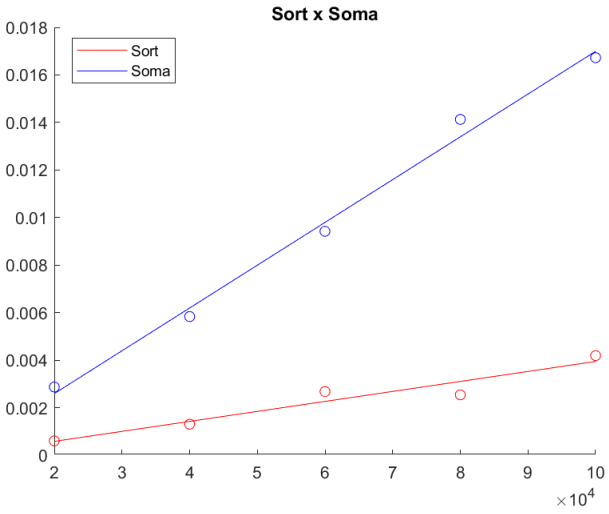
Feito por: Arthur Sophiatti

Comparação dos resultados obtidos



Ao fazer a analise do gráfico com as três soluções juntas, fica evidente a discrepância no tempo de execução do algoritmo por meio da exaustão para os demais. Visto que estes, independentemente do tamanho, das amostras utilizadas nunca é superior a um segundo

Para se entender melhor a diferença dos algoritmos de "Soma" e de "Sort" precisa-se olhas especialmente para essas duas funções



Como abordado anteriormente, ambas são lineares, fazendo a regressão das retas (que são da forma $ax + b$) o coeficiente que define a inclinação da mesma é o "a".

A reta do algoritmo de "Sort" possui

$$a = 4.202008247375488e - 08$$

Já a reta do algoritmo de "Soma" possui

$$a = 1.800314188003e - 07$$

Por fim, após toda análise supracitada torna-se evidente a superioridade do algoritmo de "Sort" aos demais, pois após o ordenamento da lista, dificilmente o método ira percorrer ela por inteiro. Embora o nível de amostragem seja pequeno, tal diferença já foi elucidada, e seguindo as previsões matemáticas mostradas através das regressões é possível concluir que quanto mais se aumente o manho (N) das listas estudadas maior será a diferença do tempo de execução dos algoritmos.

Feito por: Arthur Sophiatti

Imagens dos algoritmos para apreciação do docente

```python
def SolExaustiva(lista: list):  # Analise de complexiadade: O(n^2)
    start = time.time()

    for i in range(len(lista)):

        for j in range(i + 1, len(lista)):

            if lista[i] == lista[j]:

                end = time.time()

                return end - start
```

```python
def SolSort(lista: list):   # Analise de complexiadade: O(nlogn) ou O(n)
    start = time.time()
    lista.sort()

    for i in range(len(lista) - 1):
        if lista[i] == lista[i + 1]:
            end = time.time()

            return end - start
```

```python
def SolSoma(lista: list) -> float:  # Analise de complexiadade: O(n)
    start = time.time()

    n = len(lista) - 1
    SomaPA = (1 + n) * n / 2

    SomaLst = 0
    for i in range(len(lista)):
        SomaLst += lista[i]

    elem = SomaLst - SomaPA

    end = time.time()

    return end - start
```

Feito por: Arthur Sophiatti