Exercício 1 - 12/09/2022

	UFV - Universidade Federal de Viçosa DPI - Departamento de Informática Prof. André Gustavo dos Santos INF 630 - Projeto e Análise Algoritmos - 2022/1	Atividade 1 Para segunda 12/09/2022 Jeronimo Costa Penha - ES 91669

Algoritmos para 3-SUM

O problema 3-Sum consiste em, dado uma lista de *n* números, decidir se existem 3 deles cuja soma seja 0. Na versão desta atividade, considere que deve descobrir todos os conjuntos de 3 com essa propriedade.

1. Algoritmo força-bruta:
    - Implementar o algoritmo força-bruta $O(n^3)$ comentado na aula
    - Anotar o tempo de execução para diferentes n (ex: 100, 500, 1000, 2000, 5000)
    - Estimar uma função de tempo em função de n: T (n) = ?
    - Verificar se a função de tempo é uma boa estimativa para n maiores (ex: 10000)
2. Comparação de algoritmos
    - Implementar o algoritmo $O(n^2 \log_2 n)$ com busca binária comentado na aula
    - Implementar um algoritmo $O(n^2)$ (pesquise!)
    - Anotar o tempo de execução dos três algoritmos para diferentes n
    - Fazer gráfico comparativo dos tempos dos 3 algoritmos
    - Estimar ou verificar até que valor de n cada algoritmo resolve em 10 segundos

Obs.: não é necessário fazer análise teórica ou estatística dos resultados, trata-se apenas de experimento.

Relatório

Para a resolução da atividade foram utilizados:

- Um Desktop com o processador Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz e 64GB de memória RAM DDR4 2133 MHz.
- Sistema Operacional Ubuntu GNU/Linux 20.04.05 x86_64 com kernel 5.15.0-46-generic.
- Linguagem de programação python 3.8.10.
- Visual Studio Code versão 1.71.0.
- Função bisect() para vetores em python que possui ordem de complexidade $O(\log_2^{n})$.Ref
- Os gráficos foram gerados com o auxílio da biblioteca Matplotlib.
- Utilização da biblioteca Time para a contagem do tempo de execução.
- As listas foram criadas da mesma forma para cada experimento com números sequenciais, sendo o menor valor igual a [(N/2) - 1] * -1, e o maior igual a N/2 e foram entregues ordenadas para cada experimento.
- Cada algoritmo foi executado 10 vezes para a obtenção dos valores dos experimentos.

Relatório de execução:

- Para estimar o tempo de execução para N=10000, foi acrescentada uma constante $K_{médio}$ multiplicada à equação de complexidade de cada algoritmo
- A constante $K_{médio}$ foi definida com a média das constantes K_n calculadas para cada instância.

1. Algoritmo 3-SUM Força bruta
   Código

```
def sum3_fbruta(vec):
    start = time.time_ns()
    sum3 = 0
    qtde_valores = len(vec)
    for i in range(qtde_valores):
        for j in range(i+1, qtde_valores):
            for k in range(j+1, qtde_valores):
                if vec[i]+vec[j]+vec[k] == 0:
                    sum3 += 1
                    break

    end = time.time_ns()
    return sum3, end - start
```

Execução:

N	$T_{(s)}$	$K_{(n)}$
100	0.022	2.212e-08
500	2.782	2.225e-08
1000	21.543	2.154e-08
2000	165.590	2.070e-08
5000	2482.418	1.986e-08

- Onde $K_{(n)} = T_{(n)} / n^3$
- $K_{médio}$ = 2.129e-08
- $T_{(n)(estimado)} = K * n^3$

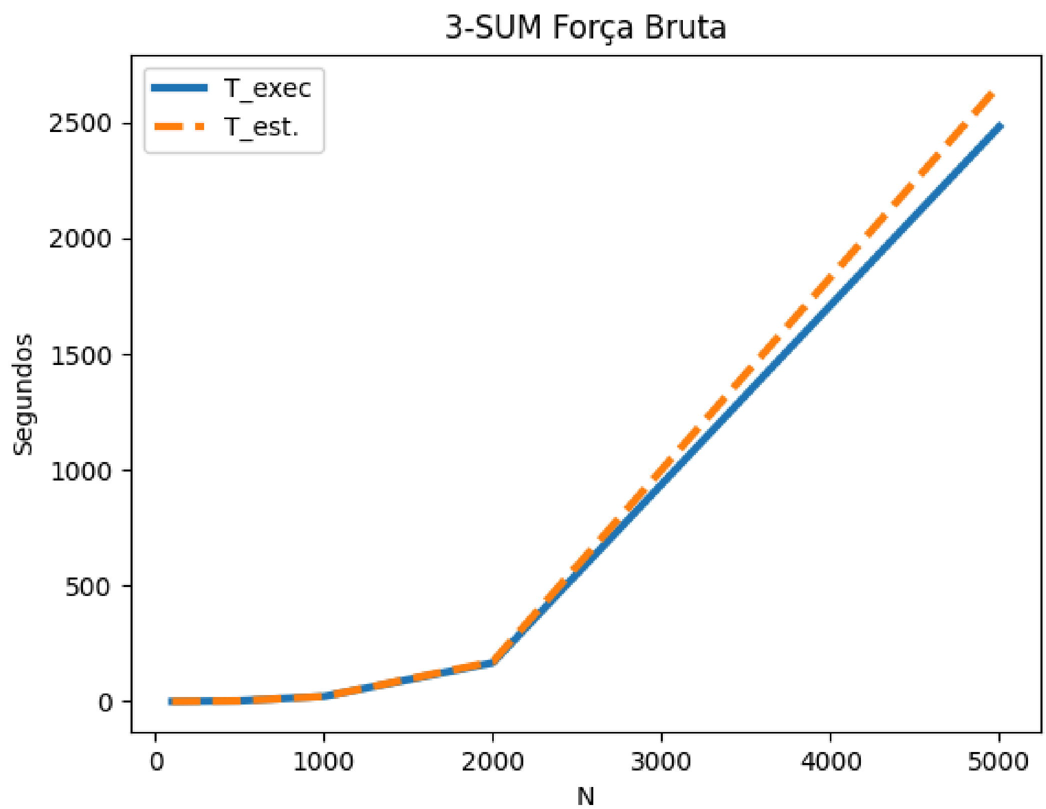
Estimativa de tempos de execução

N	$T_{(s)}$	$T_{(e)(estimado)}$
100	0.022	0.021
500	2.782	2.662
1000	21.543	21.295
2000	165.590	170.359

N	$T_{(s)}$	$T_{(e)(estimado)}$
5000	2482.418	2661.860

- Para N = 10000, o valor estimado foi de **21294.882s** equivalente a 5h 54m 53s

Gráfico com o tempo medido e o tempo estimado



- As estimativas de tempo de execução foram razoavelmente precisas ao se considerar as curvas observadas no gráfico, porém verifica-se um aumento na distância entre as curvas para valores maiores. Creio que a estimativa feita possa ser usada para se ter uma ideia da tendência do tempo de execução do algoritmo.

2. Algoritmo 3-SUM $n^2 \log_2^n$

   Código

```
def sum3_bisect(vec):
    sum3 = 0
    qtde_valores = len(vec)
    for i in range(qtde_valores):
        for j in range(i+1, qtde_valores):
            l = (vec[i] + vec[j]) * -1
            k = bisect.bisect_left(vec[j+1:qtde_valores], l)
            if (k + j + 1) != qtde_valores and vec[(k + j + 1)] == l:
                sum3 += 1

    end = time.time_ns()
    return sum3, end - start
```

Execução:

N	**$T_{(s)}$**	**$K_{(n)}$**
100	0.003	4.709e-08
500	0.127	5.672e-08
1000	0.719	7.213e-08
2000	4.837	1.103e-07
5000	70.744	2.303e-07
10000	584.878	4.402e-07

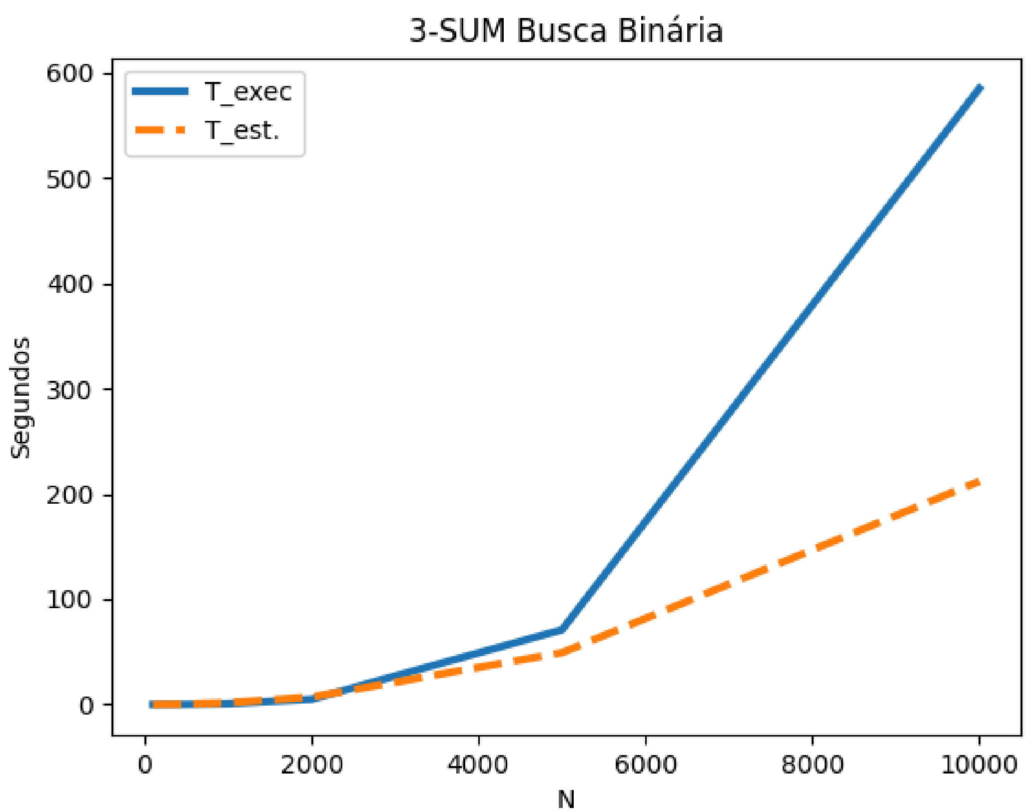
- Onde $K_{(n)} = T_{(n)} / (n^2 \log_2{}^n)$
- $K_{médio}$ = 1.594e-07
- $T_{(n)(estimado)} = K * (n^2 \log_2{}^n)$

Estimativa de tempos de execução

N	**$T_{(s)}$**	**$T_{(e)(estimado)}$**
100	0.003	0.011
500	0.127	0.357
1000	0.719	1.589
2000	4.837	6.994

N	$T_{(s)}$	$T_{(e)(estimado)}$
5000	70.744	48.980
10000	584.878	211.863

Gráfico com o tempo medido e o tempo estimado



- As estimativas de tempo de execução foram mais próximas apenas para os valores menores. Isto pode ter ocorrido por conta da execução desses experimentos terem sido executados em tempos curtos e o cálculo para o $k_{médio}$ ter sido afetado por falta de precisão. Imagino que um $k_{médio}$ gerado a partir de valores maiores que 500 possam entregar uma previsão mais próxima.

3. Algoritmo 3-SUM n^2

   Código

```
def sum3_optimized(vec):
    vec.sort()
    start = time.time_ns()
    sum3 = []
    qtde_valores = len(vec)
    for i in range(qtde_valores):
        j = i+1
        k = qtde_valores - 1
        while (j < k):
            s = vec[i] + vec[j] + vec[k]
            if s > 0:
                k -= 1
            elif s < 0:
                j += 1
            else:
                sum3.append([vec[i], vec[j], vec[k]])
                j += 1
    end = time.time_ns()
    return sum3, end - start
```

Execução:

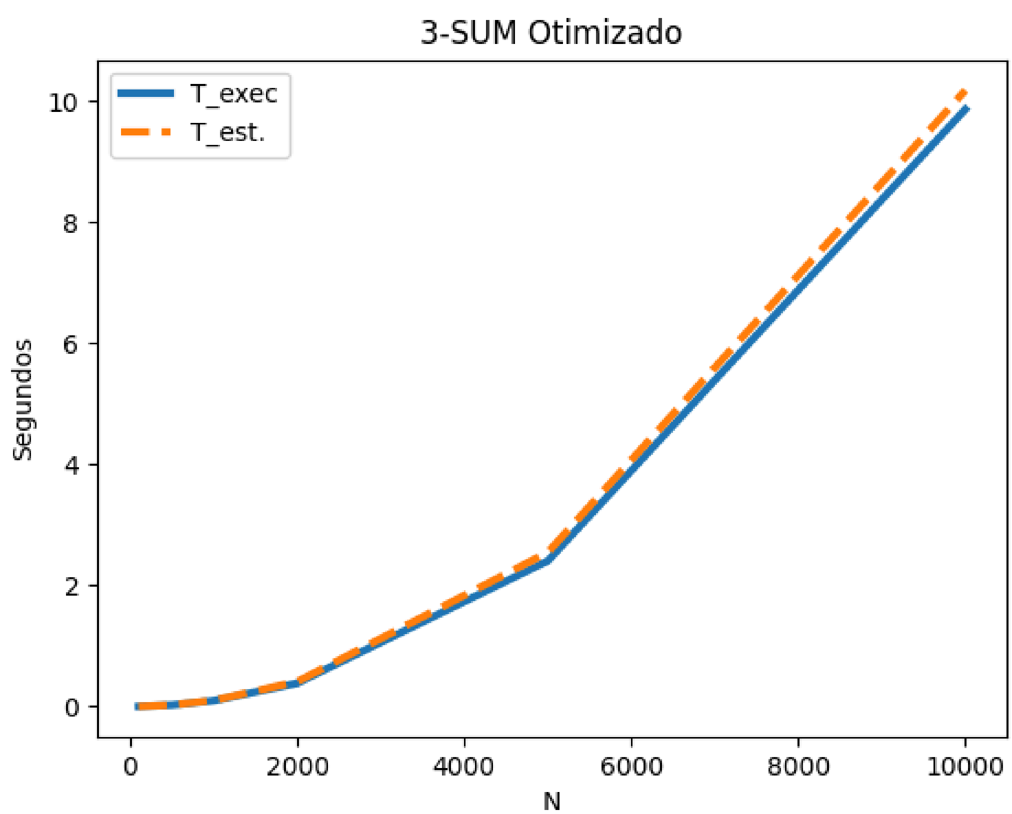
N	$T_{(s)}$	$K_{(n)}$
100	0.001	1.266e-07
500	0.024	9.746e-08
1000	0.096	9.637e-08
2000	0.382	9.555e-08
5000	2.405	9.620e-08
10000	9.870	9.870e-08

- Onde $K_{(n)} = T_{(n)} / n^2$
- $K_{médio}$ = 1.018e-07
- $T_{(n)(estimado)} = K * n^2$

Estimativa de tempos de execução

N	**$T_{(s)}$**	**$T_{(e)(estimado)}$**
100	0.001	0.001
500	0.024	0.025
1000	0.096	0.102
2000	0.382	0.407
5000	2.405	2.545
10000	9.870	10.182

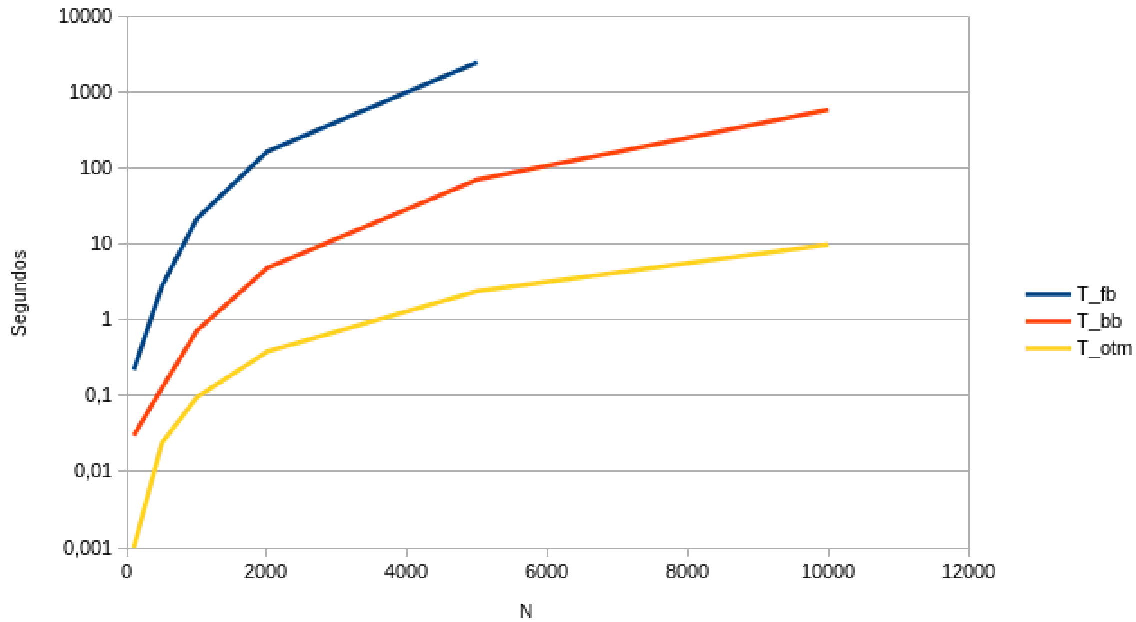
Gráfico com o tempo medido e o tempo estimado



- As estimativas de tempo de execução foram razoavelmente precisas ao se considerar as curvas observadas no gráfico. Creio que a estimativa feita possa ser usada para se ter uma ideia da tendência do tempo de execução do algoritmo.

4. Gráficos de execução

Abaixo pode-se observar o gráfico para a execução dos experimentos com os resultados dos três algoritmos juntos.



É clara a diferença de desempenho do algoritmo de força bruta para os demais por conta de reduzir 1 na potência da ordem de complexidade, porém a versão otimizada é ainda melhor com o crescimento da curva do tempo de execução mais suave.

5. Estimativa de N para uma execução de 10s para cada algoritmo
    1. Força bruta: N ~ raiz_cubica(T/K) ~ 777
    2. Com busca binária: N ~ 2366 (encontrado com o auxílio de planilha eletrônica)
    3. Otimizado: N ~ raiz_quadrada(T/K) ~ 9912