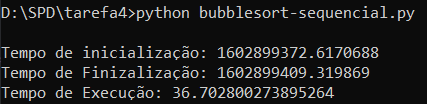
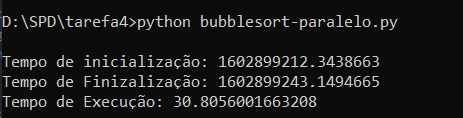
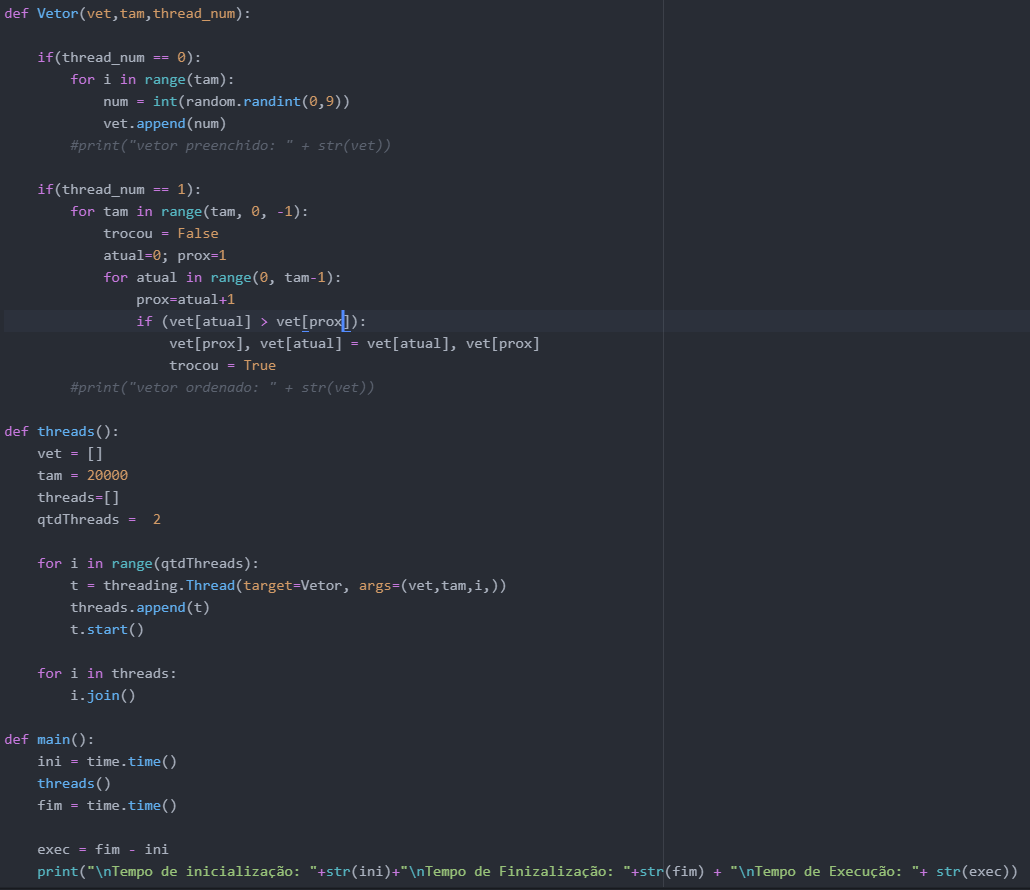
**Tarefa 4 - Speed up (Júlia)**

De início eu escolhi o algoritmo BubbleSort para verificar a diferença no tempo de execução entre um algoritmo paralelo e outro sequencial. Iniciei um vetor com 20.000 posições e fiz o teste. Porém como se trata de um algoritmo com muitas trocas e computacionalmente ineficiente, tive problemas com valores de vetores maiores.

O tempo de execução do Bubblesort sequencial foi de 36.70 e o tempo de execução do Bubblesort em paralelo foi de 30.80, resultando em um speed up de aproximadamente 1,20.

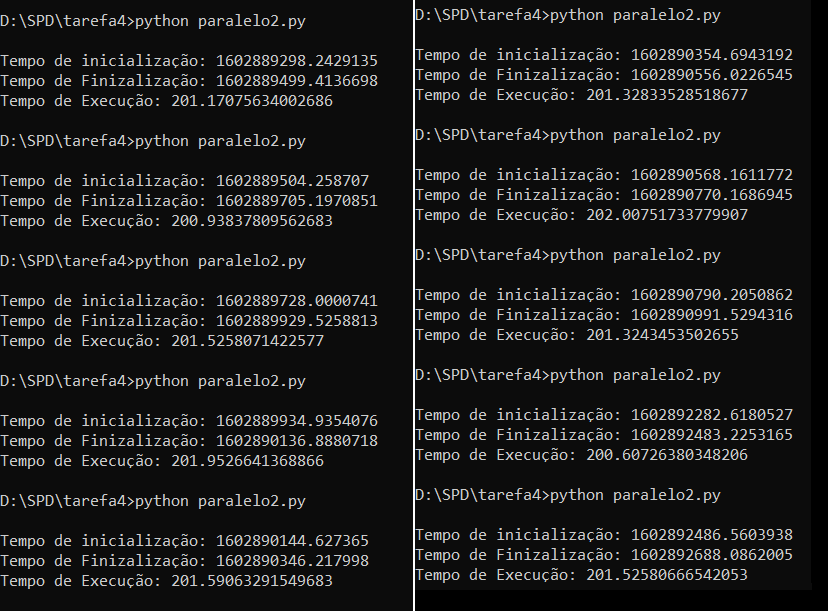
*Figura 1. Algoritmo BubbleSort Sequencial.*

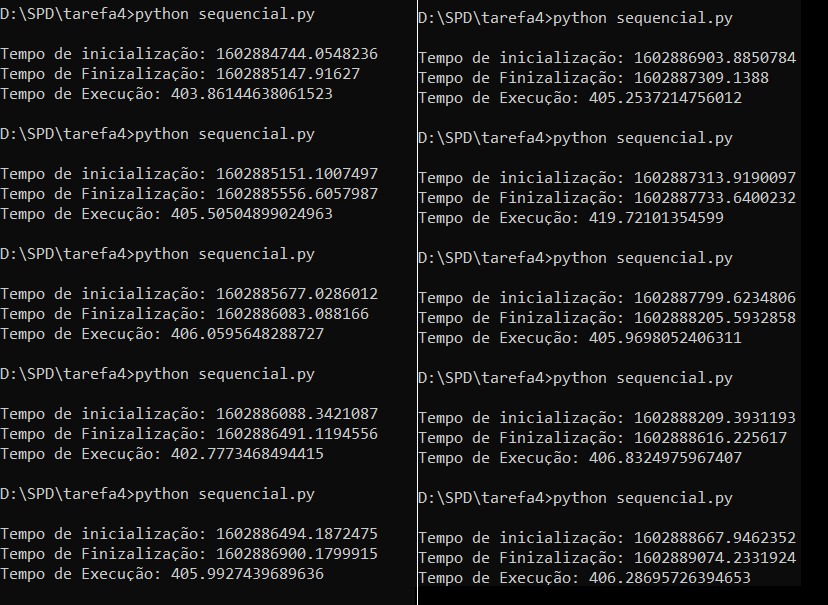
*Figura 2. Algoritmo BubbleSort em Paralelo.*

**

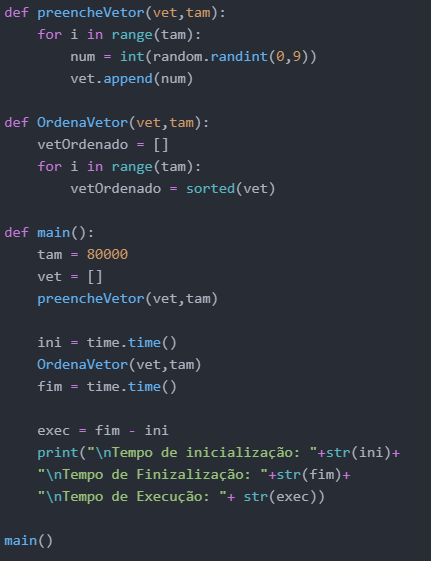
Porém, também tive a curiosidade em fazer o teste em dois algoritmos distintos usando o método próprio de ordenar vetores em ordem crescente (sorted).

*Figura 3. Resultado da saída do algoritmo paralelo usando o método sorted().*



*Figura 4. Resultado da saída do algoritmo sequencial usando o método sorted().*

A média de tempo de execução do algoritmo paralelo foi de 201,39 e a média de tempo de execução do algoritmo sequencial foi de 406,82. Ambos foram testados nas mesmas condições de ambiente. O resultado do speed up foi de aproximadamente 2.

*Figura 5. Algoritmos sequencial e paralelo respectivamente, utilizando o método sorted() que ordena vetores de forma crescente.*

Nos dois algoritmos em paralelos fiz uso da mesma técnica, uma thread para preencher o vetor e a outra para ordenar.

Conclui-se que ambos os algoritmos de ordenação em paralelo (bubblesort e sorted) obtiveram um melhor desempenho em relação ao algoritmo sequencial pois otimizaram tempo de execução.