**Compare serial and parallel executions for your video download script.**

The script provided shows a significant performance difference between serial and parallel executions using both threads and processes.

**Serial Execution:**

* **Time taken**: 21.20 seconds
* **Characteristics**: Each video is downloaded one after the other. This results in a straightforward but time-consuming process as the script waits for each download to complete before starting the next one.

**Parallel Execution with Threads:**

* **Time taken**: 6.06 seconds
* **Characteristics**: Multiple videos are downloaded concurrently. The use of a ThreadPoolExecutor and a semaphore to limit concurrent threads to 5 ensures that no more than 5 downloads happen at the same time, leveraging I/O-bound parallelism efficiently.

**Parallel Execution with Processes:**

* **Time taken**: 3.28 seconds
* **Characteristics**: Similar to threading, but using ProcessPoolExecutor to manage separate processes. Processes can handle CPU-bound tasks more efficiently than threads but have more overhead. Despite this, the processes execution showed the best performance in this case, likely due to effective distribution of tasks across multiple cores.

**Discuss the complexity of your video download scripts' time and space**

**Time Complexity:**

1. **Serial Execution (**serial\_runner**):** The time complexity is O(n), where n is the number of video URLs. Each video is downloaded one after the other, which means the total time taken is the sum of the times to download each video.
2. **Threaded Execution (**thread\_runner**):** The theoretical time complexity remains O(n), but in practice, it can be closer to O(n/m) due to parallel execution, where m is the number of threads. However, because of the Global Interpreter Lock (GIL) in Python, threads may not execute in true parallel but can still offer performance benefits over serial execution when I/O operations are involved.
3. **Multiprocessing Execution (**process\_runner**):** Similar to threading, the theoretical time complexity is O(n), but practical performance can approach O(n/m), where m is the number of processes. Multiprocessing can achieve better parallelism than threading in Python due to separate memory spaces and avoiding the GIL.

**Space Complexity**

The list of URLs has a linear space complexity, the actual video download operations have a constant space complexity because they are handled one at a time. The additional space required for managing concurrency is also constant, not dependent on the number of videos. Therefore, the overall space complexity of the script, when considering the download operations, is indeed **O(1)**