1. **Of the simulated algorithms, which algorithm is the “best” algorithm for CPU-bound processes? Justify your answer.**

From the CPU-scheduling algorithms explored in this project, the arguably “best” is the FCFS algorithm.

From the work that we were able to accomplish, we can observe and reason that the SRT and RR algorithms would not favor CPU-bound processes. For SJF and SRT, these algorithms favor I/O-bound processes due to them generally having shorter burst times. This means that the I/O-bound processes will typically end up towards the front of the CPU queue, while the longer CPU-bound processes will be towards the back, leading to a lack of work being done for CPU-bound processes since they are being starved of access to the CPU. Unfortunately, our team developed an SRT algorithm that runs to completion but is incorrect in evaluation. Instead, the team looked at the SJF algorithm, which functions mostly the same as SRT but without any preemptions. We observed that processes with shorter jobs are placed in the queue first; more often than not, due to how processes are created, I/O-bound processes are usually placed earlier in the priority queue as they typically have shorter burst times. Using these results, we reasoned and deduced that many of our findings for SJF would similarly apply to SRT.

For RR, the constant preemptions are highly inefficient for CPU-bound processes since not much work is done for each CPU-bound process during their allotted time slice. We observed that the CPU-bound average wait times and turnaround times were higher than those of FCFS, indicating that CPU-bound processes take more time to complete as compared to FCFS.

This leaves FCFS as the “best” algorithm for CPU-bound processes. Since the CPU blocks other processes whenever a process is running, it means that these long CPU-bound processes can reach completion without any interruption. The results also support this idea, as FCFS consistently had lower CPU-bound turnaround times and wait times.

2. **For the SJF and SRT algorithms, what value of α produced the "best" results? Compare with baseline results that do not use exponential averaging.**
   Range of alpha values: ($0.5 \leq \text{alpha} \leq 0.8$)
   Processes exhibit moderate correlation between successive CPU bursts, which is common in real-world workloads. Alpha values in this range balance recent observations (t_n, the actual observed CPU burst time of the current burst) with historical predictions (tau), smoothing out noise while adapting to trends. They also reduce prediction errors compared to static assumptions.

   A good default choice for alpha would be 0.5, which balances responsiveness and stability. If processes have correlated bursts, alpha = 0.5 outperforms the baseline alpha = -1. If bursts are truly random, the baseline may avoid prediction errors but performs worse in most real systems.

   For example, for processes with alternating short/long bursts (e.g., I/O-bound tasks), alpha = 0.5 adjusts predictions dynamically. For purely random bursts with no correlation, alpha = -1 avoids misleading predictions. In practice, exponential averaging (alpha ≈ 0.5) typically outperforms non-predictive baselines (alpha = -1) in real-world workloads, where burst times often exhibit some temporal correlation.

3. **For the RR algorithm, a heuristic is that RR performs best when 80% of processes are completing their CPU bursts within one time slice. Do your results confirm this heuristic?**

   From what we were able to test, our results align with this heuristic. If a process manages to finish its burst within the time slice, it reduces the number of preemptions, meaning that it is closer in efficiency to that of FCFS and SJF when comparing the metrics of the number of context switches occurring within the simulation. Another aspect that is also affected by this is a reduction in wait times and turnaround times. From on results, we observed that the CPU-bound, I/O-bound, and overall average turnaround times and wait times were lower, meaning that this is another area where the RR algorithm would be closer in efficiency to that of the FCFS and SJF algorithms. All this would mean that if processes in the simulation were able to complete their CPU bursts within the time slice, the RR algorithm would be the "best-performing" as it gains all of the benefits and efficiencies of FCFS and SJF while maintaining fairness and avoiding starvation for processes waiting to use the CPU.