1. CPU-bound processes are better supported by algorithms that lack implementation for preemptions. While utilizing CPU scheduling algorithms with preemptions can provide many benefits, they will interrupt CPU-bound processes with long burst times and as a result cause these processes to take much longer to complete. Non-preemptive algorithms allow for every process that starts using the CPU to finish using it until it has completed its CPU burst, preventing the processes from acquiring lengthy turnaround times. An example of this is shown in Figure 1, where process N is preempted in Shortest Remaining Time (SRT) and terminates later than in First Come First Serve (FCFS), a non-preemptive algorithm. Shortest Job First (SJF), another non-preemptive algorithm, sorts the queue by placing the jobs with the shortest CPU burst times in the front which are usually I/O-bound processes. This isn't necessarily bad for CPU-bound processes but it could take longer for the CPU-bound processes to enter the running state. Therefore, FCFS is the best algorithm for the CPU-bound processes.

```
time 9533ms: Process P switching out of CPU; will block on I/O until time 11325ms [Q N M H J B]
time 9537ms: Process N (tau 146ms) started using the CPU with 17ms burst remaining [Q M H J B]
time 9543ms: Process K (tau 61ms) completed I/O; preempting N [Q K M H J B]
time 9547ms: Process K (tau 61ms) started using the CPU with 4ms burst remaining [Q N M H J B]
time 9551ms: Process K (tau 61ms) completed a CPU burst; 4 bursts to go [Q N M H J B]
time 9551ms: Recalculated tau (19ms) for process K [Q N M H J B]
time 9551ms: Process K switching out of CPU; will block on I/O until time 10153ms [Q N M H J B]
time 9555ms: Process N (tau 146ms) started using the CPU with 11ms burst remaining [Q M H J B]
time 9557ms: Process D (tau 68ms) completed I/O; preempting N [Q D M H J B]
time 9559ms: Process E (tau 51ms) completed I/O; placed on ready queue [Q E D N M H J B]
time 9561ms: Process E (tau 51ms) started using the CPU with 213ms burst remaining [Q D N M H J B]
time 9619ms: Process I (tau 35ms) completed I/O; placed on ready queue [Q I D N M H J B]
time 9774ms: Process E (tau 51ms) completed a CPU burst; 44 bursts to go [Q I D N M H J B]
time 37234ms: Process O terminated [Q G E]
time 37394ms: Process N terminated [Q E]
time 51361ms: Process E terminated [Q M G]
```

Figure 1: SRT (on top) shows N being preempted twice and terminating at a later time than FCFS (on bottom)
(From output4.txt and output4-full.txt)

```
time 32371ms: Process F terminated [Q H]
time 35911ms: Process N terminated [Q H]
time 36171ms: Process O terminated [Q E]
```

I/O-bound processes, on the other hand, are better supported by algorithms that utilize preemptions. I/O-bound processes usually have much shorted CPU burst times than CPU-bound processes and since preemptions kick out processes with longer running burst times for ones with shorter burst times, the I/O-bound processes can be completed much quicker. Process A preempts several other processes in SRT and as a result is able to terminate earlier than in FCFS (Figure 2). While Round Robin (RR) is dedicated to being fair to both CPU-bound and I/O-bound processes, SRT focuses on completing the processes with the shorter burst times thus making it the superior choice in algorithm for I/O bound processes.

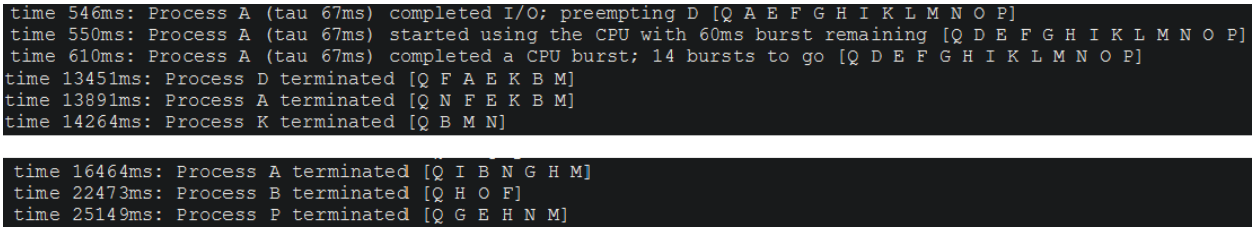

Figure 2: SRT (on top) shows process A preempting process D and terminating at an earlier time than FCFS (on bottom) (From output4.txt and output4-full.txt)

2. Changing rr_add from END to BEGINNING can create various changes in the average turnaround time, average wait time, CPU utilization, total number of context switches, and total number of preemptions. This is because the results are heavily dependent on the order processes arrive as well as the time slice. Processes that have a CPU burst time shorter than the time slice are favored while longer ones constantly pushed to the back. If the processes are all shorter than the time

slice then it becomes Last Come First Serve (LCFS). However, it can create an increased number of context switches if there a lot of processes with CPU burst times longer than the time slice. Overall, this is definitely not better than RR it tailors towards processes that have low burst times while the traditional RR attempts to create more fairness between CPU-bound and I/O bound processes.

```
Algorithm FCFS
-- average CPU burst time: 84.304 ms
-- average wait time: 215.423 ms
-- average turnaround time: 303.726 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 57.779%
Algorithm SJF
-- average CPU burst time: 84.304 ms
-- average wait time: 199.345 ms
-- average turnaround time: 287.648 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 59.366%
Algorithm SRT
-- average CPU burst time: 84.304 ms
-- average wait time: 216.981 ms
-- average turnaround time: 305.851 ms
-- total number of context switches: 613
-- total number of preemptions: 76
-- CPU utilization: 58.679%
Algorithm RR
-- average CPU burst time: 84.304 ms
-- average wait time: 229.361 ms
-- average turnaround time: 320.108 ms
-- total number of context switches: 865
-- total number of preemptions: 328
-- CPU utilization: 58.355%
```

```
Algorithm FCFS
-- average CPU burst time: 84.304 ms
-- average wait time: 215.423 ms
-- average turnaround time: 303.726 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 57.779%
Algorithm SJF
-- average CPU burst time: 84.304 ms
-- average wait time: 199.345 ms
-- average turnaround time: 287.648 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 59.366%
Algorithm SRT
-- average CPU burst time: 79.778 ms
-- average wait time: 209.601 ms
-- average turnaround time: 294.132 ms
-- total number of context switches: 629
-- total number of preemptions: 101
-- CPU utilization: 57.788%
Algorithm RR
-- average CPU burst time: 84.304 ms
-- average wait time: 251.350 ms
-- average turnaround time: 342.246 ms
-- total number of context switches: 885
-- total number of preemptions: 348
-- CPU utilization: 58.214%
```

Figure 3: RR algorithm with rr_add = BEGINNING (right) has a worst stats all around than RR algorithm with rr_add = END (From simout04.txt)

3. Obviously, since SRT is a preemptive algorithm while SJF is not, changing to SRT will result in more context switches from the preemptions and which thus causes a longer turnaround time. However the largest impact that comes from the change is in the response time of the processes. In Figure 4, process C using SJF terminated at 4113ms while using SRT it terminated at 3643ms. Also in Figure 5, process I terminated at 76256ms using SJF while it terminated at 77148ms using SRT. SRT kicks processes with longer CPU burst times for processes with shorter CPU burst

times so it makes sense that process C, which has 4 CPU bursts, terminates faster in SRT than in SJF, while process I, which has 77 CPU bursts, takes longer to terminate in SRT than in SJT.

```
time 4113ms: Process C terminated [Q E O J N I]
time 8245ms: Process L terminated [Q F H N]
time 10655ms: Process J terminated [Q B]
```

Figure 4: Process C terminating earlier in SRT (below) than in SJF (above) (From output04.txt)

```
time 3643ms: Process C terminated [Q A D E O J N I]
time 7944ms: Process L terminated [Q O H B]
time 10084ms: Process J terminated [Q B]
```

```
time 72268ms: Process H terminated [Q <empty>]
time 73993ms: Process G terminated [Q <empty>]
time 77148ms: Process I terminated [Q <empty>]
```

Figure 5: Process I terminating later in SRT (below) than in SJF (above) (From output04.txt)

```
time 69837ms: Process H terminated [Q <empty>]
time 74757ms: Process G terminated [Q <empty>]
time 76256ms: Process I terminated [Q <empty>]
```

4. One limitation of the simulation is that there is no implementation to emulate using multiple CPU cores. Nowadays, most computers have multiple cores which would allow for processes to run on the CPU parallel to one another. Adding implementation to simulate the utilization of multiple cores of a CPU would greatly improve the running time and more accurately represent a real operating system. Another is the lack of an implementation of an algorithm for a queue for processes in I/O. Realistically, once a process goes to waiting for I/O, there would be a queue for which processes get done first. If the processes in this queue are arranged by their I/O burst times, then that could change when certain processes get back to

the CPU ready queue, back on the running state, and when they terminate. Adding support for this I/O queue into the system would improve realism in the simulation. This simulation also lacks the ability to recognize the amount of time it takes for memory to actually be accessed and respond to the CPU. The location of the data being used (the hard drive, the RAM, the CPU's cache memory, etc), the speed of the transfer rate of the data can be affected and potentially increase the time that processes spend using the CPU. A real-world OS would account for this and adding such information to the simulation would emulate a true OS better.

5. A different priority scheduling algorithm could be created using elements from the ones used in the simulation along with a few new ideas. The priority would be calculated from a series of important factors: arrival time, CPU burst time, the time a process has been in the queue, and the ratio of I/O-bound processes. A time slice will also be used but they won't be used in calculating the priority of a process. Each time a new process first goes on the ready queue, the addition of their arrival time and their CPU burst time together results in their priority. The smallest number in the queue has the highest priority. After the first time, the addition of their remaining CPU burst time and the time just spent in the I/O subsystem results in the new priority. Ties are settled by checking the current ratio of I/O-bound processes to CPU-bound processes in the queue. If there are more I/O-bound processes than CPU-bound processes, then the processes with the shorter

burst time has the higher priority. If the opposite is true, then the process with the longer burst time has the higher priority. If both processes have the same burst time, then it follows FCFS. This ratio is also used to adjust the time slice. Each time a new process is added to the queue, the ratio is recalculated. Each time there are more I/O-bound processes the time slice will become equal to the burst time of the process with the shortest burst time. Each time there are more CPU-bound processes the time slice will become equal to the burst time of the process with the longest burst time. If the queue is empty or there are an equal number of I/O-bound processes and CPU-bound processes, then the time slice is not changed. To prevent any processes from starving or facing indefinite blocking, aging will also be implemented. Each time a new process is inserted in the queue ahead of a process, its priority increases by one (so subtract 1 from it) and the queue is recalculated. There are many advantages to this algorithm. Processes can't stave and it is relatively fair to CPU-bound and I/O bound processes. Some additional advantages are that the algorithm actually get quite similar to SRT or FCFS depending on which types of processes show up in the queue. The more I/O-bound processes there are the longer the time slice stays low and the longer short CPU burst times are favored making it similar to SRT. The more CPU-bound processes there are the longer the time slice stays high and the longer long CPU burst times are favored making it similar to FCFS. Some disadvantages are if there is a solid mix of I/O-bound and CPU-bound processes then it is possible to create longer

average wait times and more overheads than the other algorithms much similar to RR (Figure 6). A pretty major disadvantage that is similar to SRT and SJF is that CPU burst times must be predicted.

```
Algorithm FCFS
-- average CPU burst time: 84.304 ms
-- average wait time: 215.423 ms
-- average turnaround time: 303.726 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 57.779%
Algorithm SJF
-- average CPU burst time: 84.304 ms
-- average wait time: 199.345 ms
-- average turnaround time: 287.648 ms
-- total number of context switches: 537
-- total number of preemptions: 0
-- CPU utilization: 59.366%
Algorithm SRT
-- average CPU burst time: 84.304 ms
-- average wait time: 216.981 ms
-- average turnaround time: 305.851 ms
-- total number of context switches: 613
-- total number of preemptions: 76
-- CPU utilization: 58.679%
Algorithm RR
-- average CPU burst time: 84.304 ms
-- average wait time: 229.361 ms
-- average turnaround time: 320.108 ms
-- total number of context switches: 865
-- total number of preemptions: 328
-- CPU utilization: 58.355%
```

Figure 6: Shows how RR compares to the other algorithms and the disadvantages that would be shared with the new algorithm

(From simout04.txt)