1.

Best “algorithm”: different algorithms behave differently as the input size get larger and larger. In our measurement, best algorithm is the one that has the lowest average turnaround time, as shown in figure 1, SRT (Green-dots) seems to stand out among those four because it shows a consistent performance (comparing with other three algorithms) even when input size is large. However, when it comes to real life application, we might end up with a different result because there are still many other factors we have not taken into an account (such as starvation…).

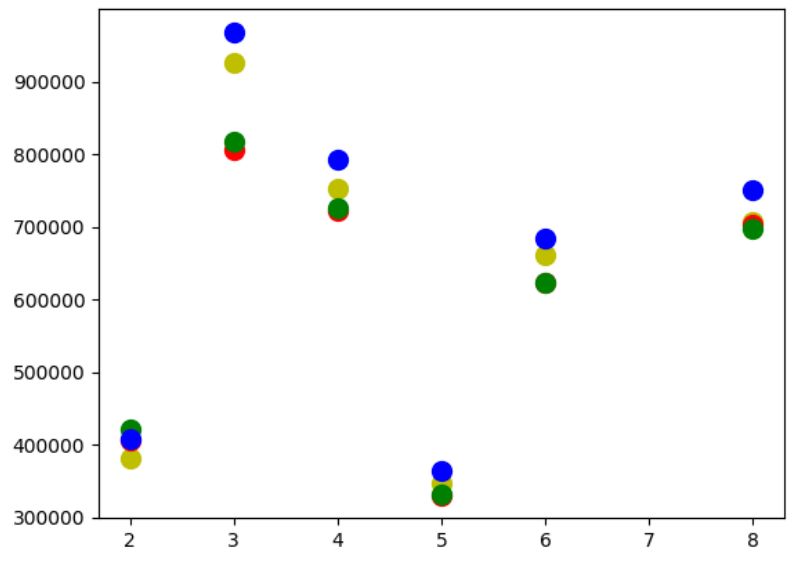

Figure 1:

y-axis is the average time around time, x-axis is seed

Red-dots represents algorithm SJF, Blue-dots represents RR, Green-dots represents SRT, Yellow-dots represents FCFS

Best-suited algorithm for CPU-bound processes: To test this, we have to input a very large CPU burst time, as shown in figure 2, algorithm with lower average turnaround time overall will be best-suited algorithm for CPU-bound process, which in this case is SJF.

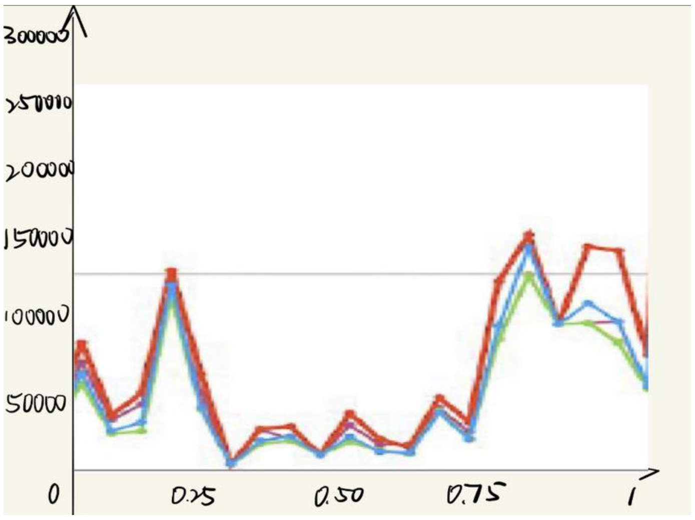

Figure 2:

Blue-line FCFS, Green-line SJF, Yellow-line SRT, Red-line RR

y-axis is the average turnaround time, x-axis is the seed value

Best-suited algorithm for IO-bound processes: To test this, we have to input a very large IO burst time, as shown in figure 3, algorithm with lower average turnaround time overall will be best-suited algorithm for CPU-bound process, which in this case is FCFS.

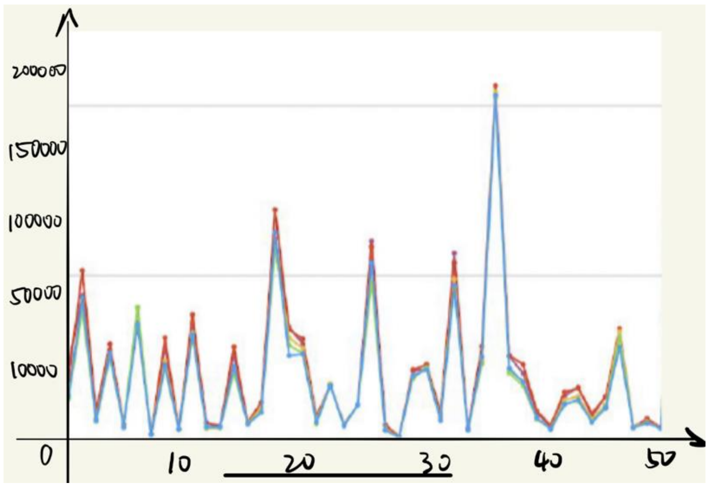

Figure 3:

Blue-line FCFS, Green-line SJF, Yellow-line SRT, Red-line RR

y-axis is the average turnaround time, X-axis is the number of random inputs test cases

2.

To find the better algorithm, we should look at which method gives a better result at average turnaround time. Comparing results between RR_BEGINNING and RR_END, as shown in figure 4, RR_BEGINNING ends up with a lower average turnaround time, therefore we consider RR_BEGINNING has a better performance.

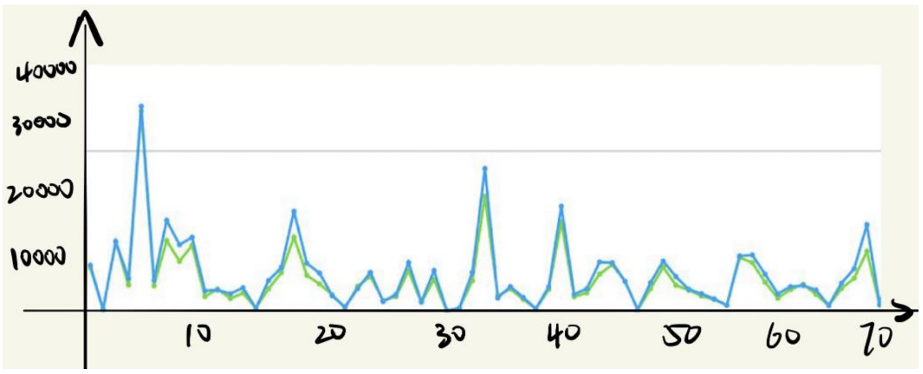

Figure 4:

y-axis is turnaround time, X-axis is the number of random inputs test cases

RR_END is blue line and RR_BEGINNING is green line

3.

Manipulating the value of α while keep all other variables unchanged, we obtain the figure shown below. As you can see, SJF does not have a big fluctuation in its total turnaround time, SRT changes dramatically as value of α changes. To pick the best value of α, we can simply ignore SJF in this case since its turnaround time cost is relatively stable. As shown in figure 5, it is obvious that when α = 1.00, our program will result in the lowest turnaround time for algorithm SRT. Thus, for this question, 1.00 is the value to get the best answer.

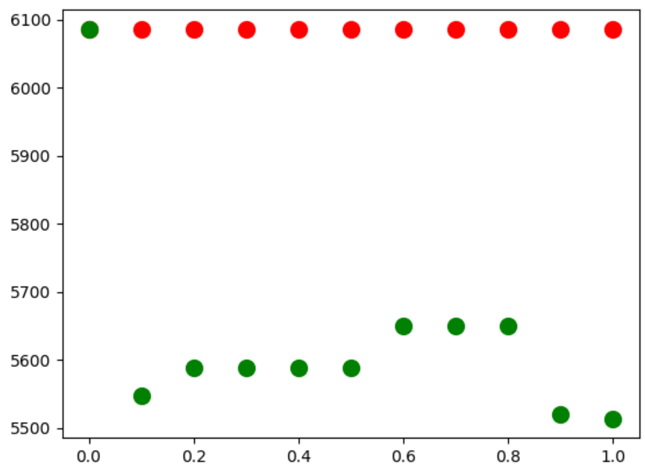

Figure 5:

y-axis is the average time around time, x-axis is different alpha value

Red-dots represents algorithm SJF and Blue-dots represents SRT

Processor = 16, seed = 2, lambda = 0.01, upper bound = 256

4.

After running simulations to compare the results of SRT and SJF, we find that although changing from non-preemptive to preemptive can potentially improve the starvation, the preemptive algorithm would perform longer time, which indicates that the non-preemptive algorithm has a relatively better performance compared with a preemptive algorithm. (we will have a lower average turnaround time for non-preemptive algorithm than preemptive algorithm).

5.

Limit 1: program fails to detect the level of process starvation.

By calculating the standard deviation of waiting time when the program is outputting the results, we can find out the level of process starvation. (bigger standard deviation we get for waiting time, more likely to experience starvation in our program)

Limit 2: program fails to simulate multiple-core CPU.

At present, we use multiple-core CPU daily. In this case, simulating a multiple-core CPU is more realistic and applicable than simulating a single-core CPU.

Simply changing the codes of our program which allows it to simulate multiple-core CPU will essentially solve this problem.

Limit 3: program fails to manage IO time and CPU burst time.

In real world operating system, it is more likely for a program to experience a short CPU burst time to finish running but a relatively long IO time when inputting and outputting the results (because output result is slower than computer calculating speed). Even though in our simulation, we have a relatively large output, but comparing with real world application, it is still not enough considering the CPU burst time.

To solve it, we can ask users to input a certain ratio, which allows the program to simulate real world operating system as much as possible.

6.

- Priority scheduling algorithm: Largest Burst First

  Develop an algorithm that assign processes into different positions in the ready queue based on burst number (from high to low). It is not a preemptive algorithm, process that has been burst the most will be put in the first position of the queue.

- Advantages: Speed of finishing process is steady, early results are obtainable as the program runs.

- Disadvantages: It is more likely to experience starvations than any other algorithms because some processes must sit until last minute for execution.

  According to figure 6, our program does experience a decent amount of waiting time somewhere as the input size changes, however, its speed of finishing each process is also significant.

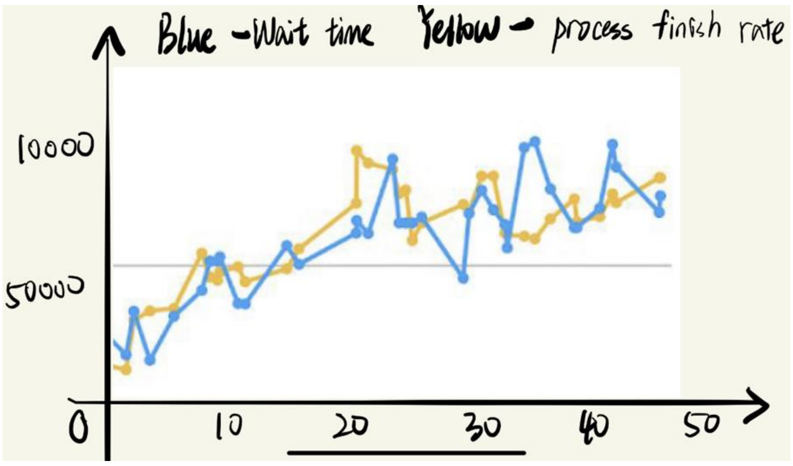

Figure 6:

Y-axis is total time, X-axis is the number of random inputs test cases