Chapter 8 Questions:

1. Run a few randomly-generated problems with just two jobs and two queues; compute the MLFQ execution trace for each. Make your life easier by limiting the length of each job and turning off I/Os.

Example1:

```
1e18@dhcp-172-17-32-41 cpu-sched-mlfq % ./mlfq.py -s 20 -n 2 -j 2 -i 0 -m 10 -M 0 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS ioTime 0
OPTIONS stayAfterIO False
OPTIONS iobump False


For each job, three defining characteristics are given:
  startTime : at what time does the job enter the system
  runTime   : the total CPU time needed by the job to finish
  ioFreq    : every ioFreq time units, the job issues an I/O
              (the I/O takes ioTime units to complete)

Job List:
  Job  0: startTime   0 - runTime   9 - ioFreq   0
  Job  1: startTime   0 - runTime   7 - ioFreq   0


Execution Trace:

[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 8 (of 9) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 7 (of 9) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 6 (of 9) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 5 (of 9) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 4 (of 9) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 3 (of 9) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 2 (of 9) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 1 (of 9) ]
[ time 8 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 0 (of 9) ]
```

```
[ time 9 ] FINISHED JOB 0
[ time 9 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 6 (of 7) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 5 (of 7) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 4 (of 7) ]
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 3 (of 7) ]
[ time 13 ] Run JOB 1 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 2 (of 7) ]
[ time 14 ] Run JOB 1 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 1 (of 7) ]
[ time 15 ] Run JOB 1 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 0 (of 7) ]
[ time 16 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround   9
  Job  1: startTime   0 - response   9 - turnaround  16

  Avg  1: startTime n/a - response 4.50 - turnaround 12.50
```

Example2:

```
1e18@dhcp-172-17-32-41 cpu-sched-mlfq % ./mlfq.py -s 20 -n 2 -j 2 -i 0 -m 20 -M 0 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS ioTime 0
OPTIONS stayAfterIO False
OPTIONS iobump False


For each job, three defining characteristics are given:
  startTime : at what time does the job enter the system
  runTime   : the total CPU time needed by the job to finish
  ioFreq    : every ioFreq time units, the job issues an I/O
              (the I/O takes ioTime units to complete)

Job List:
  Job  0: startTime   0 - runTime  18 - ioFreq   0
  Job  1: startTime   0 - runTime  15 - ioFreq   0


Execution Trace:
```

```
[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 17 (of 18) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 16 (of 18) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 15 (of 18) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 14 (of 18) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 13 (of 18) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 12 (of 18) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 11 (of 18) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 10 (of 18) ]
[ time 8 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 9 (of 18) ]
[ time 9 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 8 (of 18) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 14 (of 15) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 13 (of 15) ]
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 12 (of 15) ]
[ time 13 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 11 (of 15) ]
[ time 14 ] Run JOB 1 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 10 (of 15) ]
[ time 15 ] Run JOB 1 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 9 (of 15) ]
[ time 16 ] Run JOB 1 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 8 (of 15) ]
[ time 17 ] Run JOB 1 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 7 (of 15) ]
[ time 18 ] Run JOB 1 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 6 (of 15) ]
[ time 19 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 5 (of 15) ]
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 7 (of 18) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 6 (of 18) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 5 (of 18) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 4 (of 18) ]
[ time 24 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 3 (of 18) ]
[ time 25 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 2 (of 18) ]
[ time 26 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 1 (of 18) ]
[ time 27 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 0 (of 18) ]
[ time 28 ] FINISHED JOB 0
[ time 28 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 4 (of 15) ]
[ time 29 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 3 (of 15) ]
[ time 30 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 2 (of 15) ]
[ time 31 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 1 (of 15) ]
[ time 32 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 0 (of 15) ]
[ time 33 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  28
  Job  1: startTime   0 - response  10 - turnaround  33

  Avg  1: startTime n/a - response 5.00 - turnaround 30.50
```

Example 3:
1e18@dhcp-172-17-32-41 cpu-sched-mlfq % ./mlfq.py -s 20 -n 2 -j 2 -i 0 -m 5 -M 0 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue 1 is   1
OPTIONS quantum length for queue 1 is 10
OPTIONS allotments for queue 0 is   1
OPTIONS quantum length for queue 0 is 10
OPTIONS boost 0
OPTIONS ioTime 0
OPTIONS stayAfterIO False
OPTIONS iobump False

For each job, three defining characteristics are given:
  startTime : at what time does the job enter the system
  runTime   : the total CPU time needed by the job to finish
  ioFreq   : every ioFreq time units, the job issues an I/O
              (the I/O takes ioTime units to complete)

Job List:
  Job 0: startTime   0 - runTime   4 - ioFreq   0
  Job 1: startTime   0 - runTime   4 - ioFreq   0

Execution Trace:

[time 0] JOB BEGINS by JOB 0
[time 0] JOB BEGINS by JOB 1
[time 0] Run JOB 0 at PRIORITY 1 [TICKS 9 ALLOT 1 TIME 3 (of 4)]
[time 1] Run JOB 0 at PRIORITY 1 [TICKS 8 ALLOT 1 TIME 2 (of 4)]
[time 2] Run JOB 0 at PRIORITY 1 [TICKS 7 ALLOT 1 TIME 1 (of 4)]
[time 3] Run JOB 0 at PRIORITY 1 [TICKS 6 ALLOT 1 TIME 0 (of 4)]
[time 4] FINISHED JOB 0
[time 4] Run JOB 1 at PRIORITY 1 [TICKS 9 ALLOT 1 TIME 3 (of 4)]
[time 5] Run JOB 1 at PRIORITY 1 [TICKS 8 ALLOT 1 TIME 2 (of 4)]
[time 6] Run JOB 1 at PRIORITY 1 [TICKS 7 ALLOT 1 TIME 1 (of 4)]
[time 7] Run JOB 1 at PRIORITY 1 [TICKS 6 ALLOT 1 TIME 0 (of 4)]
[time 8] FINISHED JOB 1

Final statistics:
  Job 0: startTime   0 - response   0 - turnaround   4
  Job 1: startTime   0 - response   4 - turnaround   8

Avg 1: startTime n/a - response 2.00 - turnaround 6.00

2. How would you run the scheduler to reproduce each of the examples in the chapter?
Example1: A Single Long-Running job
./mlfq.py -n 3 -q 10 -l 0,100,0 -c

Example2: Along came a interactive job
./mlfq.py -n 3 -q 10 -l 0,100,0:50,10,0 -c

Example3: A Mixed I/O-intensive and CPU-intensive Workload
./mlfq.py -n 3 -q 10 -l 0,50,0:10,15,1 -i 5 -S -c

Example4: Without Priority Boost
./mlfq.py -n 3 -q 10 -l 0,120,0:100,50,1:100,50,1 -i 1 -S -c

Example5: With Priority Boost
./mlfq.py -n 3 -q 10 -l 0,150,0:100,50,1:100,50,1 -i 2 -S -B 40 -c

Example6: Without Gaming Tolerance
./mlfq.py -n 3 -l 0,100,0:30,90,9 -i 1 -S -c

Example7: With Game Tolerance
./mlfq.py -n 3 -q 10 -l 0,100,0:30,90,9 -i 1 -c

Example8: Lower Priority, Longer Quanta
./mlfq.py -n 3 -Q 10,20,30 -l 0,200,0:0,200,0 -c

3. How would you configure the scheduler parameters to behave just like a round-robin scheduler?
When there is only one queue, the jobs on one queue would be executed in time slices.

4. Craft a workload with two jobs and scheduler parameters so that one job takes advantage of the older Rules 4a and 4b (turned on with the -S flag) to game the scheduler and obtain 99% of the CPU over a particular time interval.
./mlfq.py -n 3 -l 0,100,0:30,90,9 -i 1 -S -c

5. Given a system with a quantum length of 10ms in its highest queue, how often would you have to boost jobs back to the highest priority level (with the -B flag) to guarantee that a single longrunning (and potentially starving) job gets at least 5% of the CPU?
T(boost) = 10 / 5% = 200.

For each turn of the long job, there are 10ms to run. It's at least 5% CPU. Thus 10/5% = 200 which means every 200 ms the long job needs to get turn to run.

6. One question that arises in scheduling is which end of a queue to add a job that just finished I/O; the -I flag changes this behavior for this scheduling simulator. Play around with some workloads and see if you can see the effect of this flag.

-I means that if specified, jobs that finished I/O move immediately to the front of the current queue.
./mlfq.py -n 2 -q 10 -l 0,30,0:0,30,11 -i 1 -S -c
./mlfq.py -n 2 -q 10 -l 0,30,0:0,30,11 -i 1 -S -I -c

Chapter 9 Questions:
1. Compute the solutions for simulations with 3 jobs and random seeds of 1, 2, and 3.
./lottery.py -j 3 -s 1 -c
./lottery.py -j 3 -s 2 -c
./lottery.py -j 3 -s 3 -c

2. Now run with two specific jobs: each of length 10, but one (job 0) with just 1 ticket and the other (job 1)with 100 (e.g., -l 10:1,10:100). What happens when the number of tickets is so imbalanced? Will job 0 ever run before job 1 completes? How often? In general, what does such a ticket imbalance do to the behavior of lottery scheduling?
./lottery.py -l 10:1,10:100 -c
Job0 would rarely have chance to run before job1 completes. The probability is 1/100.
A ticket imbalance would make lottery scheduling meaningless.

3. When running with two jobs of length 100 and equal ticket allocations of 100 (-l 100:100,100:100), how unfair is the scheduler? Run with some different random seeds to determine the (probabilistic) answer; let unfairness be determined by how much earlier one job finishes than the other.
./lottery.py -s 0 -l 100:100,100:100 -c
U = 192/200 = 0.96
./lottery.py -s 1 -l 100:100,100:100 -c
U = 196/200 = 0.98
./lottery.py -s 2 -l 100:100,100:100 -c
U = 190/200 = 0.95
./lottery.py -s 3 -l 100:100,100:100 -c
U = 196/200 = 0.98

4. How does your answer to the previous question change as the quantum size (-q) gets larger?

./lottery.py -q 10 -s 0 -l 100:100,100:100 -c
U = 150/200 = 0.75
./lottery.py -q 10 -s 1 -l 100:100,100:100 -c
U = 160/200 = 0.8
./lottery.py -q 10 -s 2 -l 100:100,100:100 -c
U = 190/200 = 0.95
./lottery.py -q 10 -s 3 -l 100:100,100:100 -c
U = 190/200 = 0.95

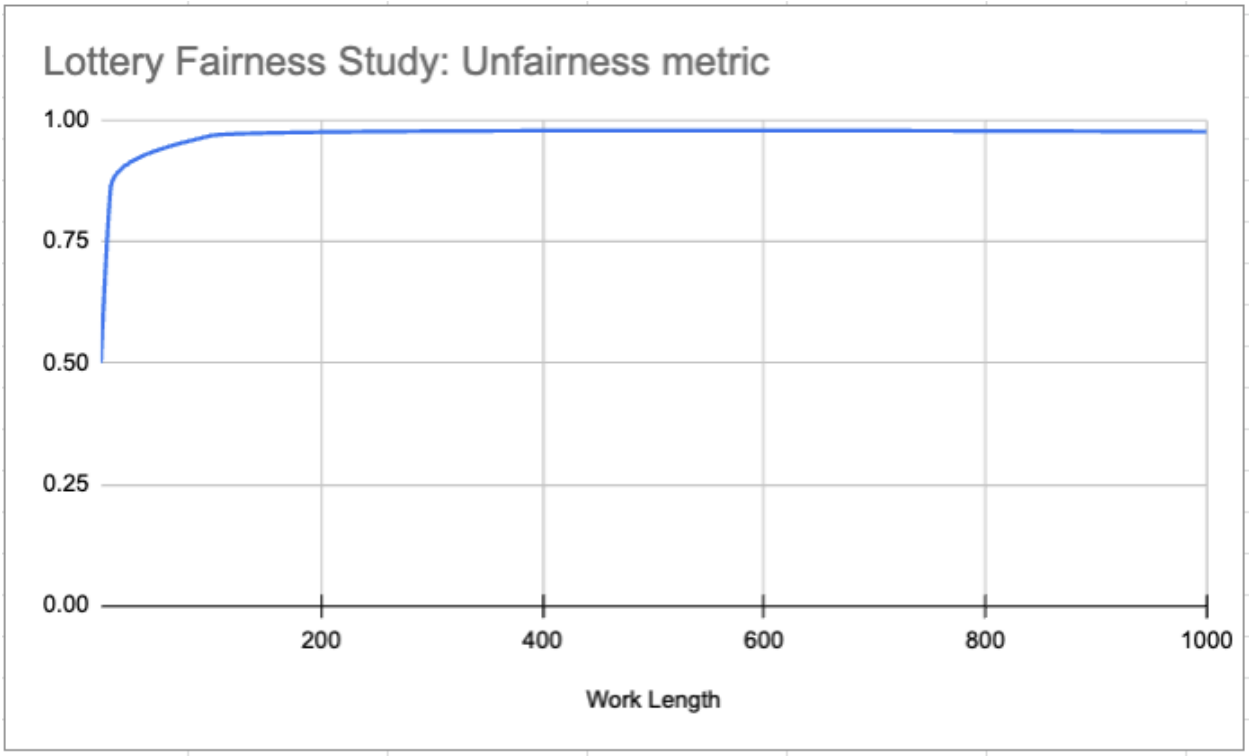
./lottery.py -q 50 -s 0 -l 100:100,100:100 -c
U = 150/200 = 0.75
./lottery.py -q 50 -s 1 -l 100:100,100:100 -c
U = 150/200 = 0.75
./lottery.py -q 50 -s 2 -l 100:100,100:100 -c
U = 100/200 = 0.5
./lottery.py -q 50 -s 3 -l 100:100,100:100 -c
U = 150/200 = 0.75

When the quantum size gets larger, unfairness metric turns to be smaller which means it's more unfair than before.

5. Can you make a version of the graph that is found in the chapter? What else would be worth exploring? How would the graph look with a stride scheduler?

Work length:

Work Length	**1**	**10**	**100**	**1000**
U(seed = 0)	0.5	0.75	0.96	0.971
U(seed = 1)	0.5	0.8	0.98	0.9515
U(seed = 2)	0.5	0.95	0.95	0.9835
U(seed = 3)	0.5	0.95	0.98	0.997
U(average)	**0.5**	**0.8625**	**0.9675**	**0.97575**



When we are using stride scheduler, the graph should be a horizontal line because the stride scheduler has the precision to get each job exactly right at the end of each scheduling cycle.