#uni/semester3/Betriebssysteme/chapter9/simulator_fragen

1. Compute the solutions for simulations with 3 jobs and random seeds of 1, 2, and 3.

```
vladb@VladB ~/G/h/S/B/H/HW5-Lottery (master)> ./lottery.py -j 3 -s 1
Here is the job list, with the run time of each job:
  Job 0 ( length = 1, tickets = 84 )
  Job 1 ( length = 7, tickets = 25 )
  Job 2 ( length = 4, tickets = 44 )
```

Tickets: 84 + 25 + 44 = 153
Job 0 ➜ 0 bis 83
Job 1 ➜ 84 bis 108
Job 2 ➜ 109 bis 152

Random 651593 % 153 = 119 JOB 2, left 3
Random 788724 % 153 = 9 JOB 0, left 0, FINISHED

Tickets: 25 + 44 = 69
Job 1 ➜ 0 bis 24
Job 2 ➜ 25 bis 68

Random 93859 % 69 = 19 JOB 1, left 6
Random 28347 % 69 = 57 JOB 2, left 2
Random 835765 % 69 = 37 JOB 2, left 1
Random 432767 % 69 = 68 JOB 2, left 0

Tickets: 25 + 0 = 25
Job 1 ➜ 0 bis 24

Random 762280 % 25 = 5 JOB 1, left 5
Random 2106 % 25 = 6 JOB 1, left 4
Random 445387 % 25 = 12 JOB 1, left 3
Random 721540 % 25 = 15 JOB 1, left 2
Random 228762 % 25 = 12 JOB 1, left 1

Random 945271 % 25 = 21 JOB 1, left 0

```
vladb@VladB ~/G/h/S/B/H/HW5-Lottery (master)> ./lottery.py -j 3 -s 2
Here is the job list, with the run time of each job:
  Job 0 ( length = 9, tickets = 94 )
  Job 1 ( length = 8, tickets = 73 )
  Job 2 ( length = 6, tickets = 30 )
```

Tickets: 94 + 73 + 30 = 197
Job 0 ➜ 0 bis 93
Job 1 ➜ 94 bis 166
Job 2 ➜ 167 bis 196

Random 605944 % 197 = 169 JOB 2, left 5
Random 606802 % 197 = 42 JOB 0, left 8
Random 581204 % 197 = 54 JOB 0, left 7
Random 158383 % 197 = 192 JOB 2, left 4
Random 430670 % 197 = 28 JOB 0, left 6
Random 393532 % 197 = 123 JOB 1, left 7
Random 723012 % 197 = 22 JOB 0, left 5
Random 994820 % 197 = 167 JOB 2, left 3
Random 949396 % 197 = 53 JOB 0, left 4
Random 544177 % 197 = 63 JOB 0, left 3
Random 444854 % 197 = 28 JOB 0, left 2
Random 268241 % 197 = 124 JOB 1, left 6
Random 35924 % 197 = 70 JOB 0, left 1
Random 27444 % 197 = 61 JOB 0, left 0

Tickets: 73 + 30 = 103
Job 1 ➜ 0 bis 72
Job 2 ➜ 73 bis 102

Random 464894 % 103 = 55 JOB 1, left 5
Random 318465 % 103 = 92 JOB 2, left 2
Random 380015 % 103 = 48 JOB 1, left 4
Random 891790 % 103 = 16 JOB 1, left 3
Random 525753 % 103 = 41 JOB 1, left 2

Random 560510 % 103 = 87 JOB 2, left 1
Random 236123 % 103 = 47 JOB 1, left 1
Random 23858 % 103 = 65 JOB 1, left 0

Tickets: 0 + 30 = 30
Job 2 ➜ 0 bis 29

Random 325143 % 30 = 3 JOB 2, left 0

```
vladb@VladB ~/G/h/S/B/H/HW5-Lottery (master)> ./lottery.py -j 3 -s 3
Here is the job list, with the run time of each job:
  Job 0 ( length = 2, tickets = 54 )
  Job 1 ( length = 3, tickets = 60 )
  Job 2 ( length = 6, tickets = 6 )
```

Tickets: 120
Job 0 ➜ 0 bis 53
Job 1 ➜ 54 bis 113
Job 2 ➜ 114 bis 119

Random 13168 % 120 = 88 Job 1, left 2
Random 837469 % 120 = 109 Job 1, left 1
Random 259354 % 120 = 34 Job 0, left 1
Random 234331 % 120 = 91 Job 1, left 0

Tickets: 60
Job 0 ➜ 0 bis 53
Job 2 ➜ 54 bis 59

Random 995645 % 60 = 5 Job 0, left 0

Tickets: 6
Job 2 ➜ 0 bis 5

Random 470263 % 60 = 43 Job 2, left 5
Random 836462 % 60 = 2 Job 2, left 4
Random 476353 % 60 = 13 Job 2, left 3

Random 639068 % 60 = 8	Job 2, left 2
Random 150616 % 60 = 16	Job 2, left 1
Random 634861 % 60 = 1	Job 2, left 0

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 behaviour of lottery scheduling?

```
vladb@VladB ~/G/h/S/B/H/HW5-Lottery (master)> ./lottery.py -l 10:1,10:100 -s 2
-c

Job 0 runs before Job 1 completes
```

➜ Es wird zwischen 101 Tickets ausgewählt. Die Wahrscheinlichkeit, dass Job 1 ununterbrochen fertig wird ist 0.9901^10 = 0.9053 ➜ 90.53%
Ein solches Ungleichgewicht, führt dazu, dass das System sich wie FIFO verhält, also der beste nach einem Kriterium (Ankommenszeit) zuerst bedient wird.

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 -l 100:100,100:100 -s 0 -c
F = 192 / 200 = 0.96

./lottery.py -l 100:100,100:100 -s 7 -c
F = 185 / 200 = 0.925

./lottery.py -l 100:100,100:100 -s 15 -c
F = 184 / 200 = 0.92

./lottery.py -l 100:100,100:100 -s 49 -c
F = 188 / 200 = 0.94
```

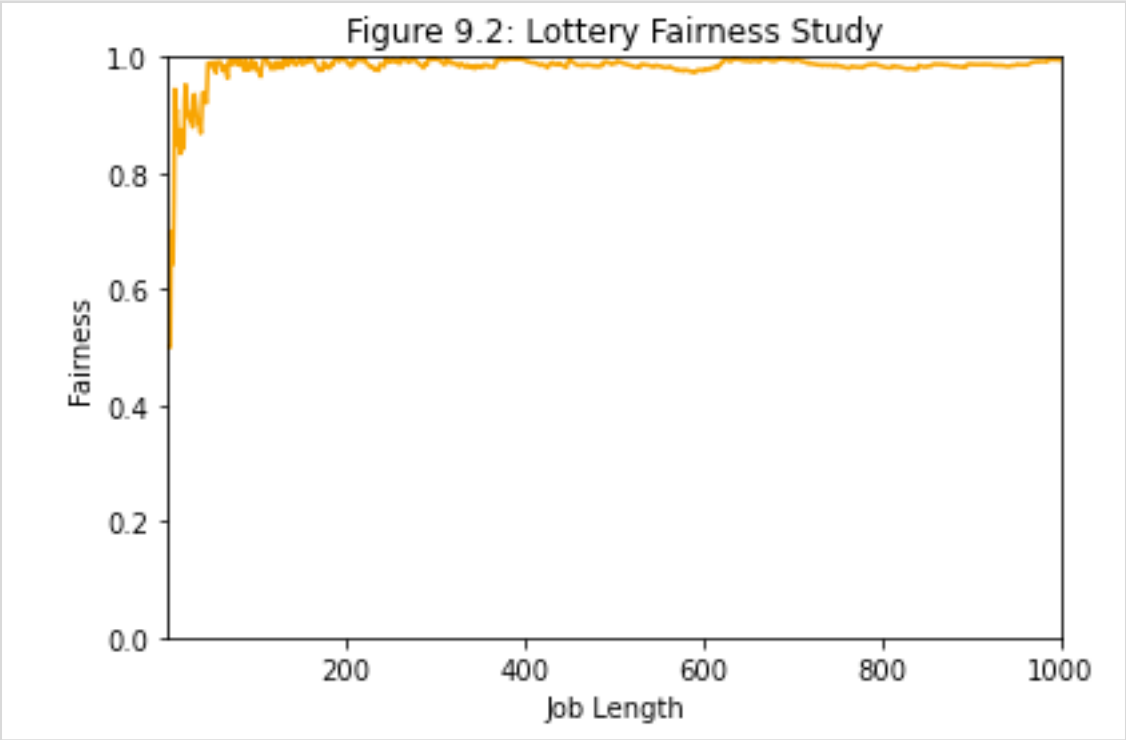
➜ 1 wäre der Idealzustand, also fast identische Zeit, wo sie fertig werden.

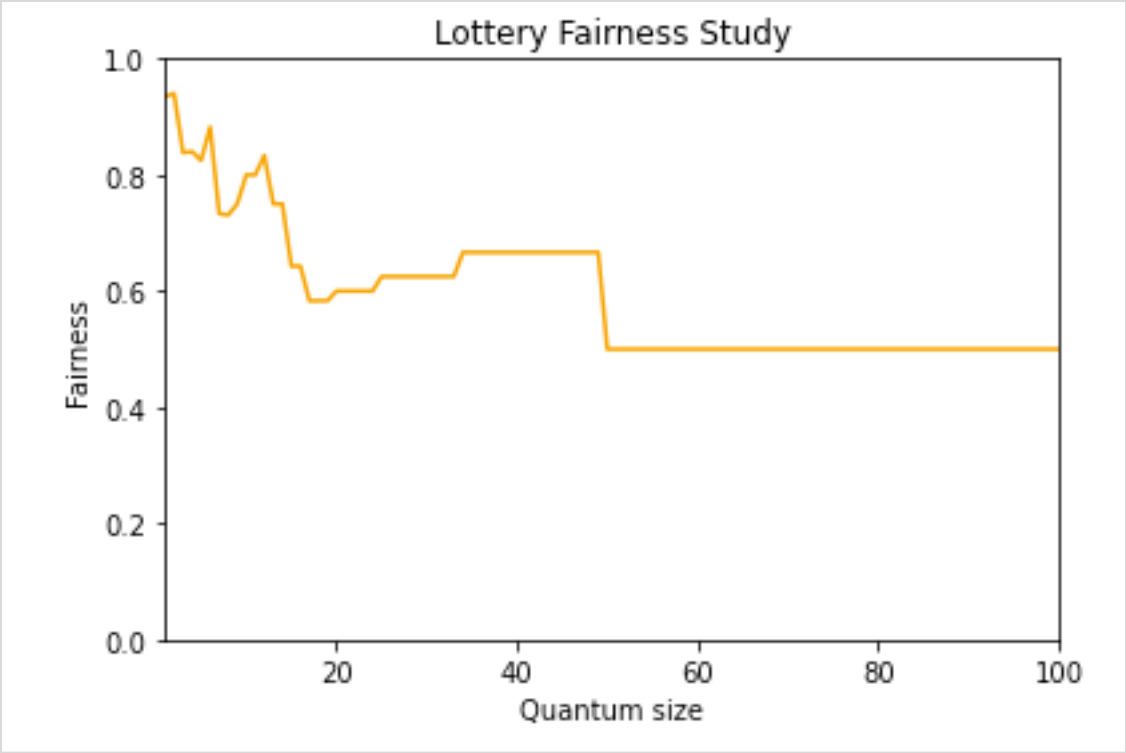
4. How does your answer to the previous question change as the quantum size (-q) gets larger?

```
./lottery.py -l 100:100,100:100 -s 0 -c -q 10
F = 150 / 200 = 0.75

./lottery.py -l 100:100,100:100 -s 7 -c -q 10
F = 140 / 200 = 0.7

./lottery.py -l 100:100,100:100 -s 15 -c -q 10
F = 190 / 200 = 0.95

./lottery.py -l 100:100,100:100 -s 49 -c -q 10
F = 180 / 200 = 0.9
```

➜ Die Fairness Metrik wird deutlich niedriger bei größeren time slices. Das ist so weil, wenn ein Prozess dran kommt, dann macht er auch mehr

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?
ostep-hw/9 at master · xxyzz/ostep-hw · GitHub





Eigentlich sollen die y-Achsen Unfairness heissen!