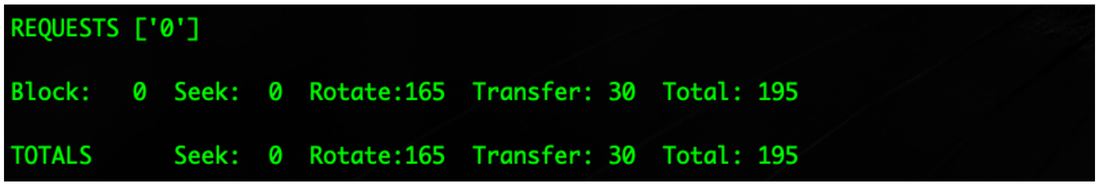
Artem Ivanov
Homework 5
ar286080

Compute the seek, rotation, and transfer times (1 pt)

Compute the seek, rotation, and transfer times for the following sets of requests:

-a 0

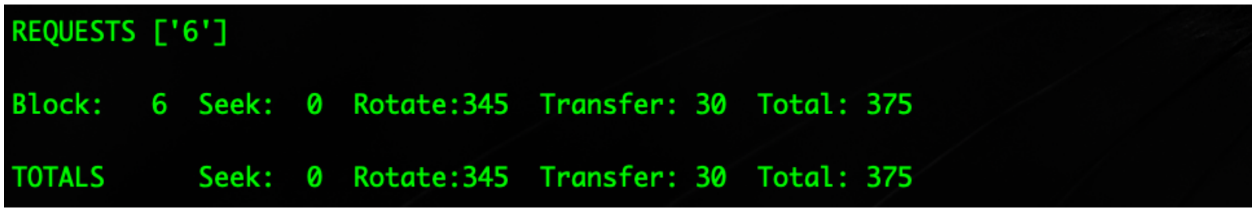


```
REQUESTS ['0']

Block:   0  Seek:  0  Rotate:165  Transfer: 30  Total: 195

TOTALS      Seek:  0  Rotate:165  Transfer: 30  Total: 195
```

-a 6

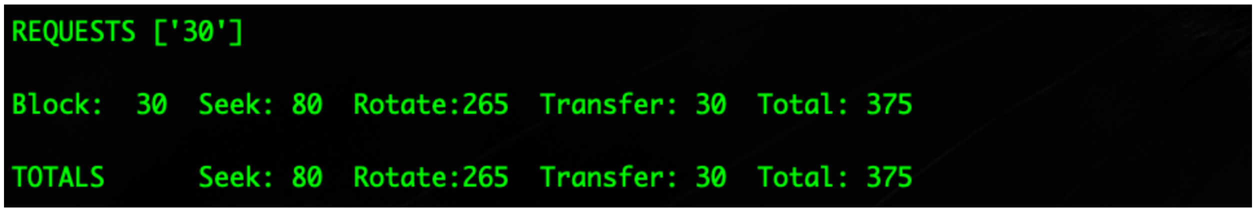


```
REQUESTS ['6']

Block:   6  Seek:  0  Rotate:345  Transfer: 30  Total: 375

TOTALS      Seek:  0  Rotate:345  Transfer: 30  Total: 375
```

-a 30

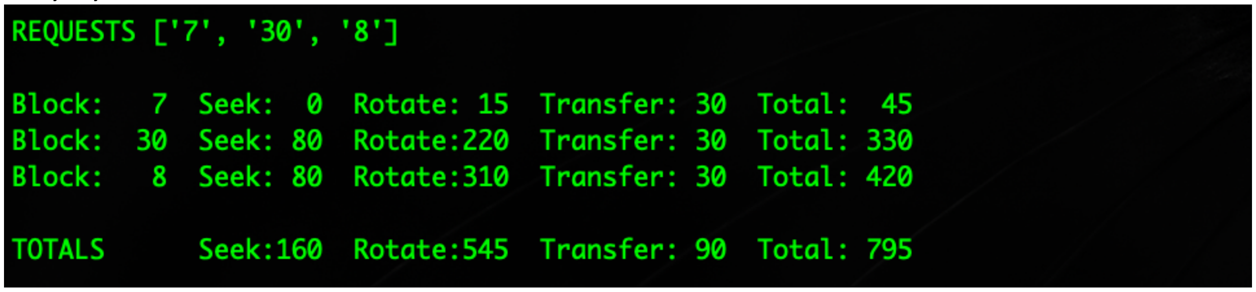


```
REQUESTS ['30']

Block:  30  Seek: 80  Rotate:265  Transfer: 30  Total: 375

TOTALS      Seek: 80  Rotate:265  Transfer: 30  Total: 375
```

-a 7,30,8

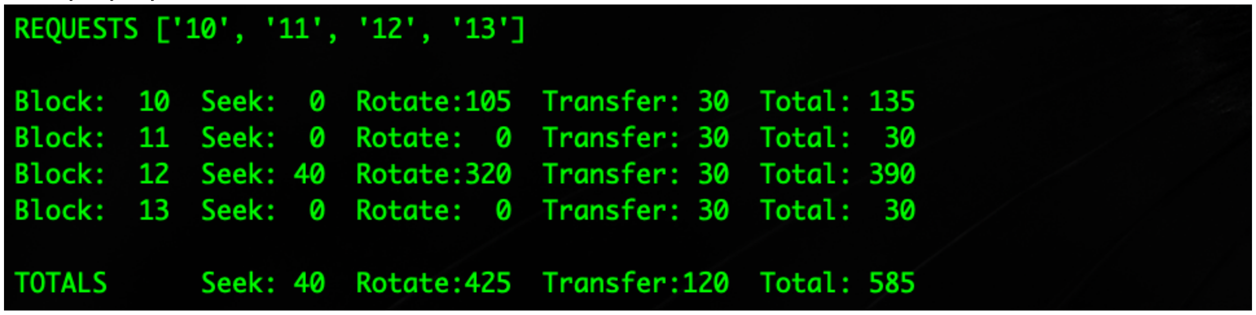


```
REQUESTS ['7', '30', '8']

Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  30  Seek: 80  Rotate:220  Transfer: 30  Total: 330
Block:   8  Seek: 80  Rotate:310  Transfer: 30  Total: 420

TOTALS      Seek:160  Rotate:545  Transfer: 90  Total: 795
```

-a 10,11,12,13



```
REQUESTS ['10', '11', '12', '13']

Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate:320  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 40  Rotate:425  Transfer:120  Total: 585
```

Different seek and rotation rates (1 pt)

Do the same requests above but change the seek rate to different values: -S 2, -S 4, -S 8, -S 10, -S 40, -S 0.1. How do the times change?

For **-S 2** overall time won't change. For first two requests time will stay exactly the same because we do not jump between the tracks. Even if we do as in the next three requests, our seek time decrease twice but we do more rotation because after starting head starts to move from different position; therefore, our total will remain the same.

For **-S 4** overall time got changed only in case [7,30,8]. The reason of this is that head starts in a better position for sector 8 (we don't need rotate through the whole track). Therefore, ration will be equal to 10 instead of 310 but in sector [30] we need to rotate extra 20 which makes overall time slightly bigger. However, we highly decrease overall time for this request. Other, request remain the same. Jumping between the tracks will be 4 times faster but require more rotation except [7,30,8] request.

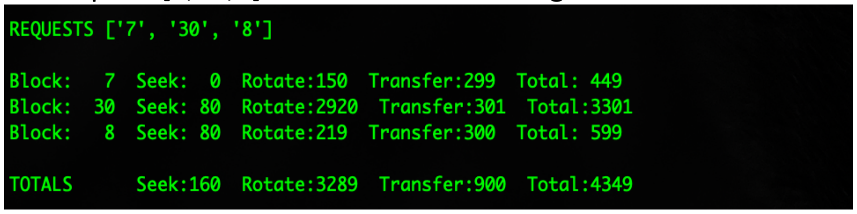
-S 8, **-S 10**, **-S 40** are exactly same as -S 4.

-S 0.1 incredibly increase our overall time for requests except first and second. As was mentioned earlier first and second requests do not require transfers between the tracks.

Conclusion:
With a default rotation rate changing the seek speed may or may not influence on overall time. It depends where the head will start to scan. It can improve the time but also it can hurt overall time if new head position after the transfer will more rotation.

Do the same requests above, but change the rotation rate: -R 0.1, -R 0.5, -R 0.01. How do the times change?

For -**R 0.1** overall time will highly increase because ration will be very slow, comparing with the default settings; of course, seek time will remain the same for all requests. For a request [0] we will have x10 longer time for rotation time, transfer through the sector and overall. For request [6] it is also about 10 times longer time but not exactly 10 because rotation time will be (x10 – 1 ms) and transfer time is (x10 + 1 ms), overall time is just 10 times higher. For request [30] it takes even more for rotation time because our head position after transfer is harmful and we need more rotation.
For request [7,30,8] we have an interesting case:



```
REQUESTS ['7', '30', '8']

Block:   7  Seek:  0  Rotate:150  Transfer:299  Total: 449
Block:  30  Seek: 80  Rotate:2920  Transfer:301  Total:3301
Block:   8  Seek: 80  Rotate:219  Transfer:300  Total: 599

TOTALS      Seek:160  Rotate:3289  Transfer:900  Total:4349
```

As we can see rotation time for block 7 is 10 time higher than a default one, transfer time and total time are (x10 – 1) higher. For block 30 rotation time is more than 10 times higher because again our head starts in a further position from the destination, we need a longer rotation. However, for block 8 our rotation time even smaller than a default time because after rotation head starts in a better position.
For request [10,11,12,13] all times are just 10 times higher except block rotation time because we need more rotation. However, overall time will be still 10 times higher.

For **-R 0.5** we have similar situation as with -R 0.1; however, in this case most of our times increase only x2 instead of x10. Similarly, we dependent on where head will start its trace.

For -**R 0.01** again very similar to -R 0.1 but this time x100 larger.

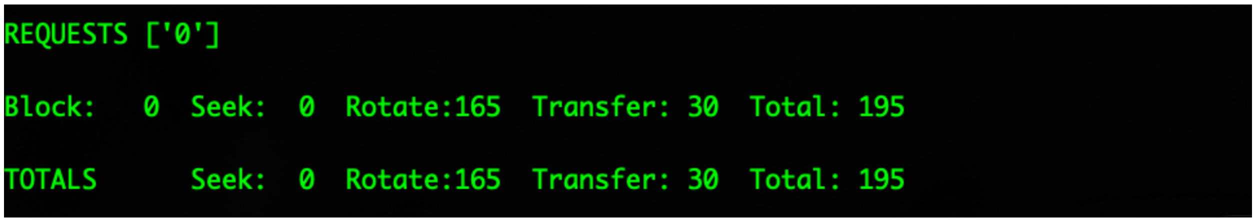
Shortest seek time first scheduler (1 pt)

FIFO is not always best, e.g., with the request stream -a 7,30,8, what order should the requests be processed in?

Requests should be processed in way where closest sectors first or shortest seek-time first. In this case 7,8,30.

Run the shortest seek-time first (SSTF) scheduler (-p SSTF) on this workload; how long should it take (seek, rotation, transfer) for each request to be served?

-a 0

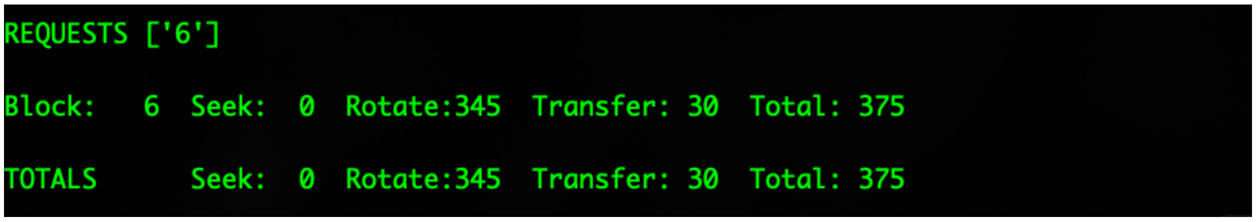


```
REQUESTS ['0']

Block:   0  Seek:  0  Rotate:165  Transfer: 30  Total: 195

TOTALS      Seek:  0  Rotate:165  Transfer: 30  Total: 195
```

-a 6

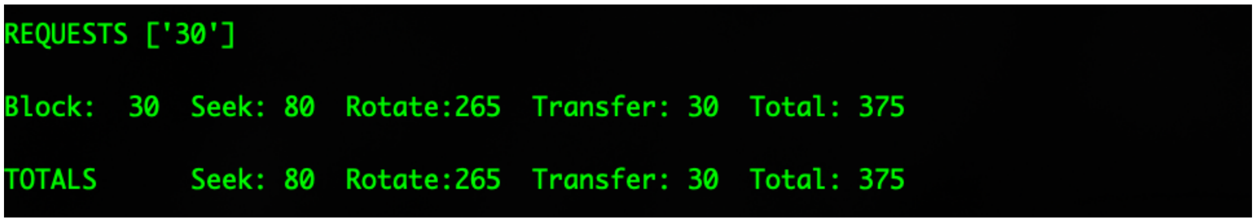


```
REQUESTS ['6']

Block:   6  Seek:  0  Rotate:345  Transfer: 30  Total: 375

TOTALS      Seek:  0  Rotate:345  Transfer: 30  Total: 375
```

-a 30

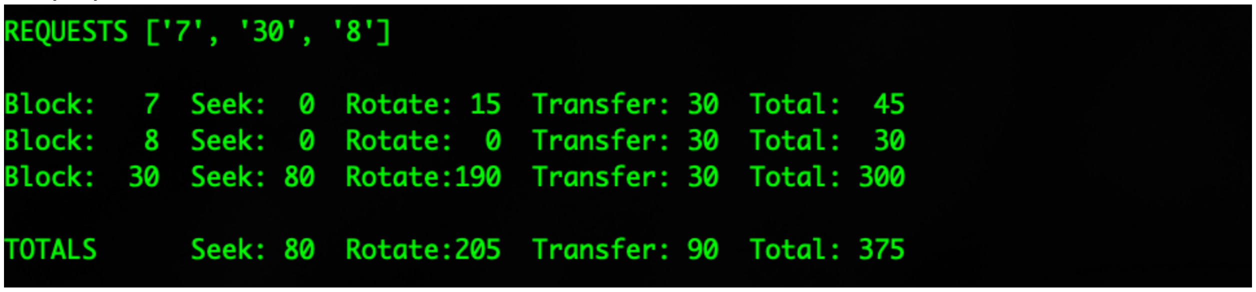


```
REQUESTS ['30']

Block:  30  Seek: 80  Rotate:265  Transfer: 30  Total: 375

TOTALS      Seek: 80  Rotate:265  Transfer: 30  Total: 375
```

-a 7,30,8

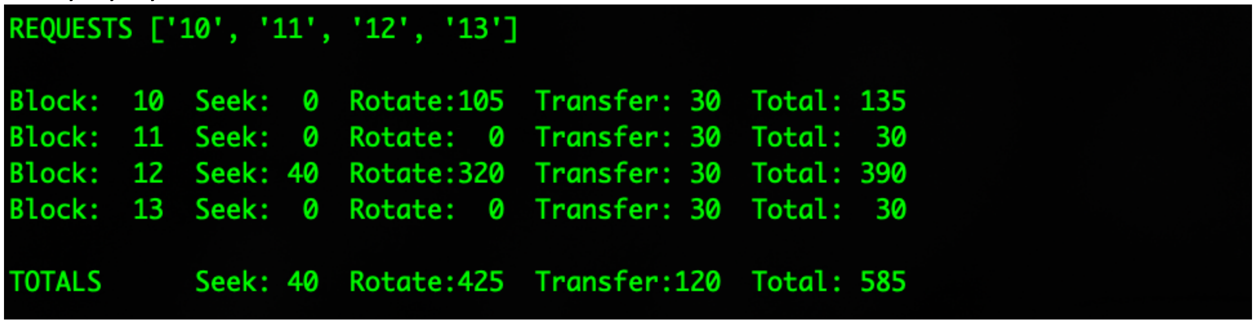


```
REQUESTS ['7', '30', '8']

Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:   8  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  30  Seek: 80  Rotate:190  Transfer: 30  Total: 300

TOTALS      Seek: 80  Rotate:205  Transfer: 90  Total: 375
```

-a 10,11,12,13



```
REQUESTS ['10', '11', '12', '13']

Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  11  Seek:  0  Rotate:  0  Transfer: 30  Total:  30
Block:  12  Seek: 40  Rotate:320  Transfer: 30  Total: 390
Block:  13  Seek:  0  Rotate:  0  Transfer: 30  Total:  30

TOTALS      Seek: 40  Rotate:425  Transfer:120  Total: 585
```

Shortest access time first scheduler (1 pt)

Now use the shortest access-time first (SATF) scheduler (-p SATF). This is similar to the shortest seek time first, but orders the requests based on the access time.
Does it make any difference for -a 7,30,8 workload?

Comparing with FIFO – yes. Comparing with SSTF – no.

Find a set of requests where SATF outperforms SSTF; more generally, when is SATF better than SSTF?

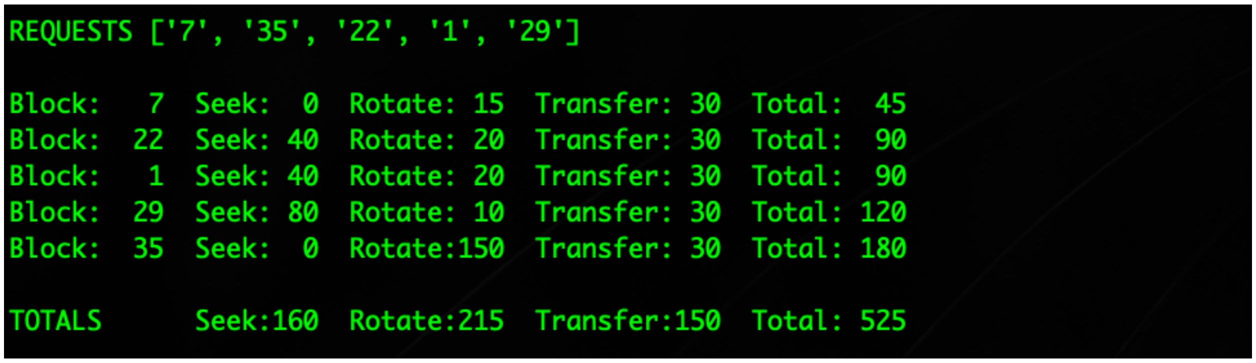
7,22,1
6,20,24
8,24,5
…

Outperforms when there is shortest access sector between sectors which are on the same track.

Starvation (1 bonus pt)

Create a series of requests to starve a particular request, assuming an SATF policy. Given that sequence, how does it perform if you use a bounded SATF (BSATF) scheduling approach?
In this approach, you specify the scheduling window (e.g., -w 4); the scheduler only moves onto the next window of requests when all requests in the current window have been serviced.

SATF

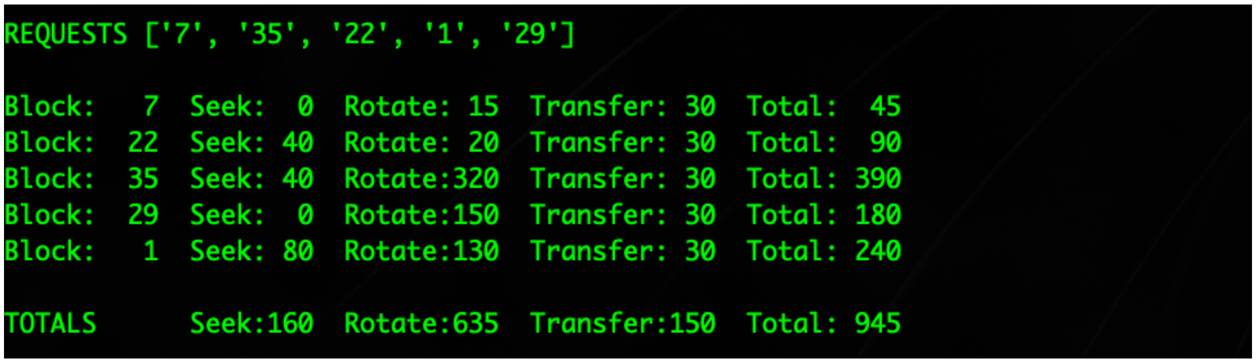


```
REQUESTS ['7', '35', '22', '1', '29']

Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  22  Seek: 40  Rotate: 20  Transfer: 30  Total:  90
Block:   1  Seek: 40  Rotate: 20  Transfer: 30  Total:  90
Block:  29  Seek: 80  Rotate: 10  Transfer: 30  Total: 120
Block:  35  Seek:  0  Rotate:150  Transfer: 30  Total: 180

TOTALS      Seek:160  Rotate:215  Transfer:150  Total: 525
```

BSATF -w 3



```
REQUESTS ['7', '35', '22', '1', '29']

Block:   7  Seek:  0  Rotate: 15  Transfer: 30  Total:  45
Block:  22  Seek: 40  Rotate: 20  Transfer: 30  Total:  90
Block:  35  Seek: 40  Rotate:320  Transfer: 30  Total: 390
Block:  29  Seek:  0  Rotate:150  Transfer: 30  Total: 180
Block:   1  Seek: 80  Rotate:130  Transfer: 30  Total: 240

TOTALS      Seek:160  Rotate:635  Transfer:150  Total: 945
```

Does this solve starvation?

It follows an 'elevator rule'. In our case we first go up and only after down.

How does it perform, as compared to SATF?

It takes almost twice longer time to process.

In general, how should a disk make this trade-off between performance and starvation avoidance?

It depends on the situation. If we are planning to use HDD as long as it is possible BSATF is better choice to keep our HDD more reliable. From the other side, if we need better performance, we probably should use SATF.