Test your algorithms using the sample values in section 5.4.1. and show the results of your tests in a project report. If you have implemented the algorithms correctly, you should end up with the same results as given in the textbook for the order of cylinder seeks and the total cylinders.

```
Overview
========

Total requested seeks: 6
Mean: 18.0000
Standard deviation: 12.8452

First come, first served
========================

Starting position: 11
Total distance: 111

1, 36, 16, 34, 9, 12

Shortest seek first
===================

Starting position: 11
Total distance: 61

12, 9, 16, 1, 34, 36

Elevator algorithm
==================

Starting position: 11
Total distance: 60

12, 16, 34, 36, 9, 1

Conclusion
==========

Total requested seeks: 6
Mean: 18.0000
Standard deviation: 12.8452

Effective seek counts
=====================

First come, first served: 6
Shortest seek first: 6
Elevator algorithm: 6
```

Now, test the algorithm using 100,000 random seek requests fifteen times (depending on the programming language, this may take some time). Report the total number of seeks for each of these runs for each algorithm along with the average and standard deviation for those 100k requests across fifteen iterations. Include the results in your project report in a well laid out table.

Run	FCFS Seeks	SSF Seeks	Elevator Seeks	Mean	Standard Deviation
1	99,998	99,981	99,992	32,844.4212	18,880.1581
2	99,998	99,985	99,993	32,747.2712	18,920.7004
3	99,999	99,988	99,995	32,722.6040	18,931.5196
4	99,998	99,986	99,996	32,835.5476	18,904.9868
5	99,999	99,986	99,993	32,832.4428	18,891.7245
6	99,998	99,986	99,995	32,817.5627	18,894.7121
7	99,999	99,984	99,993	32,802.1314	18,954.1015
8	99,997	99,980	99,988	32,787.6592	18,887.8529
9	100,000	99,988	99,996	32,770.7335	18,888.9949
10	99,998	99,988	99,997	32,815.2271	18,931.3385
11	99,999	99,980	99,995	32,754.7075	18,925.4660
12	99,999	99,989	99,991	32,796.2154	18,918.0896
13	99,997	99,986	99,996	32,803.3537	18,918.2102
14	99,998	99,985	99,997	32,841.0585	18,906.0931
15	99,996	99,980	99,992	32,677.2255	18,913.1726

What conclusions can you draw from your implementation of these simulated disk arm movement algorithms? Give a well thought out answer in your project report. (This part of your project report should be multiple paragraphs.)

My ultimate conclusion is that we can be very glad that there are SSDs. Each of these algorithms is incredibly inefficient. As is to be expected, the naive first come, first served approach is the worst by a significant margin. The shortest seek first algorithm increases the efficiency of seeks substantially, but it is highly impracticable. This simulation benefited from processing fixed sets of seek requests, whereas in an actual environment, considerably more dynamism would be needed. The elevator algorithm proves to be the best solution of the three. While it does not reach the theoretical efficiency of the shortest seek first algorithm, it still offers a substantial performance boost, and it smoothly handles incoming requests.

While the elevator algorithm is arguably the best basic solution to the problem of disk arm scheduling, it occurs to me that there may be an alternative workable approach for some systems. Especially in environments consisting of highly predictable tasks, it may be possible to approach the efficiency of the shortest seek first algorithm using a probabilistic model. The filesystem could be structured around the modeled associations between data, or the system could maintain a cached model that serves as a map. In this way, the system could anticipate and efficiently organize future disk arm movements.

This would, of course, lead to potential inefficiency should an unexpected request be received. The disk arm could simply fall back to following an elevator algorithm, starting in the direction of the

unexpected request. The performance loss could be minimal. While handling requests, it would also be possible to proactive read and cache predicted data, saving time later.

Do we need to concern ourselves with these algorithms given SSD/NVMe drives, or will we need to consider such algorithms for the foreseeable future? Justify your answer in a well thought out paragraph.

If we are working with I/O devices, disk arm algorithms certainly may still be relevant. Hard disks are still in use, though their popularity wanes. Their relevancy, however, does not by itself necessitate much concern. It is not likely that substantially new implementations of these algorithms will be needed at any point soon. This is old and well established technology. Looking to the future, solid state technology has no disk arm to schedule, and there is no reason to believe that a similar feature will be repopularized.