Disk Management

Disk Structure - Modern disk drives are addressed as large one-dimensional arrays of **logical blocks,** where the logical block is the smallest unit of transfer. The size of the logical blocks is usually 512 bytes, although some disks can be low-level formatted to choose a different logical block size, such as 1,024 bytes.

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

Disk Access Time = Seek Time + Rotational Latency +Transfer Time

Disk Scheduling Algorithms

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

Advantages:

- Every request gets a fair chance
- No indefinite postponement Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Advantages:

- High throughput
- Low variance of response time
- Average response time Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

4. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

Advantages:

- Provides more uniform wait time compared to SCAN

5. **LOOK:** It is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

6. **CLOOK:** As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

First Come First Serve (FCFS):

FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Algorithm:

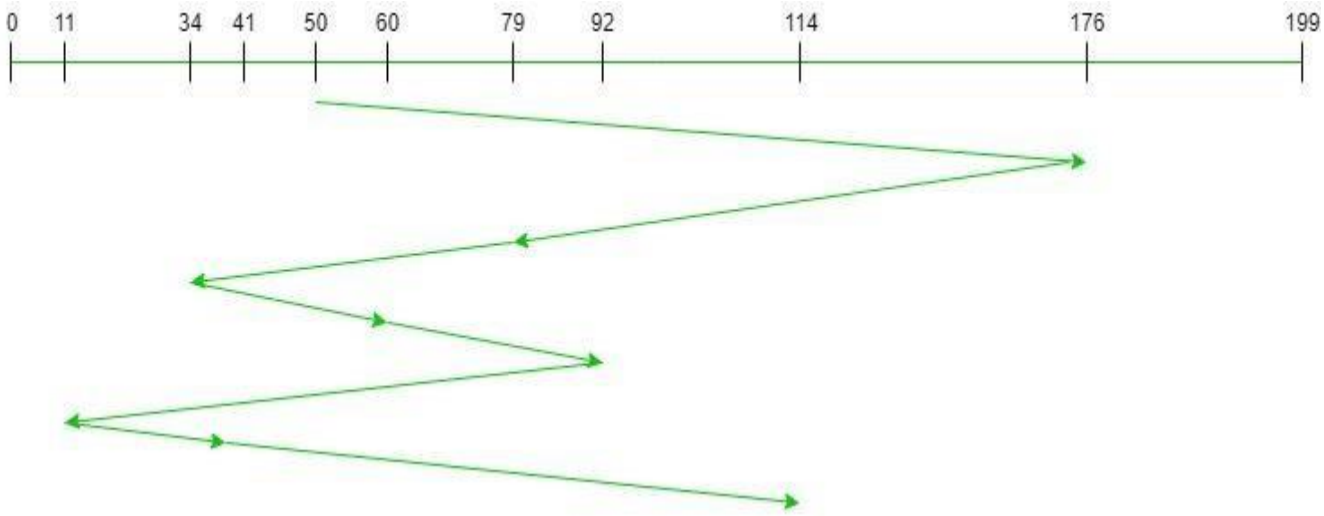
1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

Example:

Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50

Output:
Total number of seek operations = 510
Seek Sequence is
176
79
34
60
92
11
41
114

The following chart shows the sequence in which requested tracks are serviced using FCFS.



Therefore, the total seek count is calculated as:

= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41) = 510

Shortest Seek Time First (SSTF):

Basic idea is the tracks which are closer to current disk head position should be serviced first in order to *minimize the seek operations*.

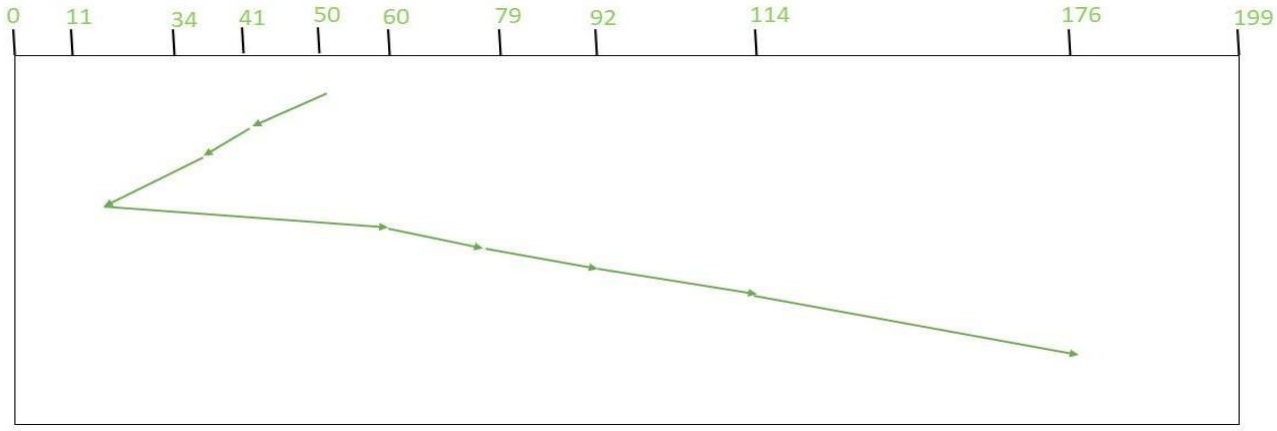
Algorithm –

1. Let Request array represents an array storing indexes of tracks that have been requested. 'head' is the position of disk head.
2. Find the positive distance of all tracks in the request array from head.
3. Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
4. Increment the total seek count with this distance.
5. Currently serviced track position now becomes the new head position.
6. Go to step 2 until all tracks in request array have not been serviced.

Example –

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50

The following chart shows the sequence in which requested tracks are serviced using SSTF.



Therefore, total seek count is calculates as:

= (50-41)+(41-34)+(34-11)+(60-11)+(79-60)+(92-79)+(114-92)+(176-114)

= 204

SCAN (Elevator) algorithm:

In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the **elevator algorithm**. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Algorithm-

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.

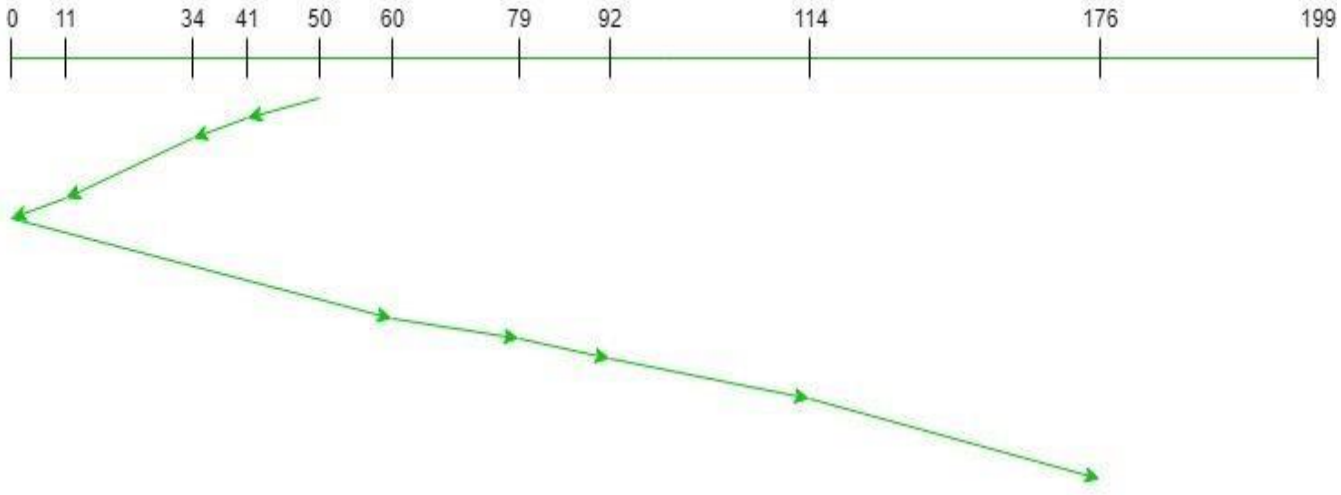
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which head is moving service all tracks one by one.
4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach at one of the ends of the disk.
8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

Example:

Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = left (We are moving from right to left)

Output:
Total number of seek operations = 226
Seek Sequence is
41
34
11
0
60
79
92
114
176

The following chart shows the sequence in which requested tracks are serviced using SCAN.



Therefore, the total seek count is calculated as:

= (50-41)+(41-34)+(34-11) +(11-0)+(60-0)+(79-60) +(92-79)+(114-92)+(176-114)

= 226

C-SCAN algorithm:

What is C-SCAN (Circular Elevator) Disk Scheduling Algorithm? Circular SCAN (C-SCAN) scheduling algorithm is a modified version of SCAN disk scheduling algorithm that deals with the inefficiency of SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the "Circular Elevator Algorithm" as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

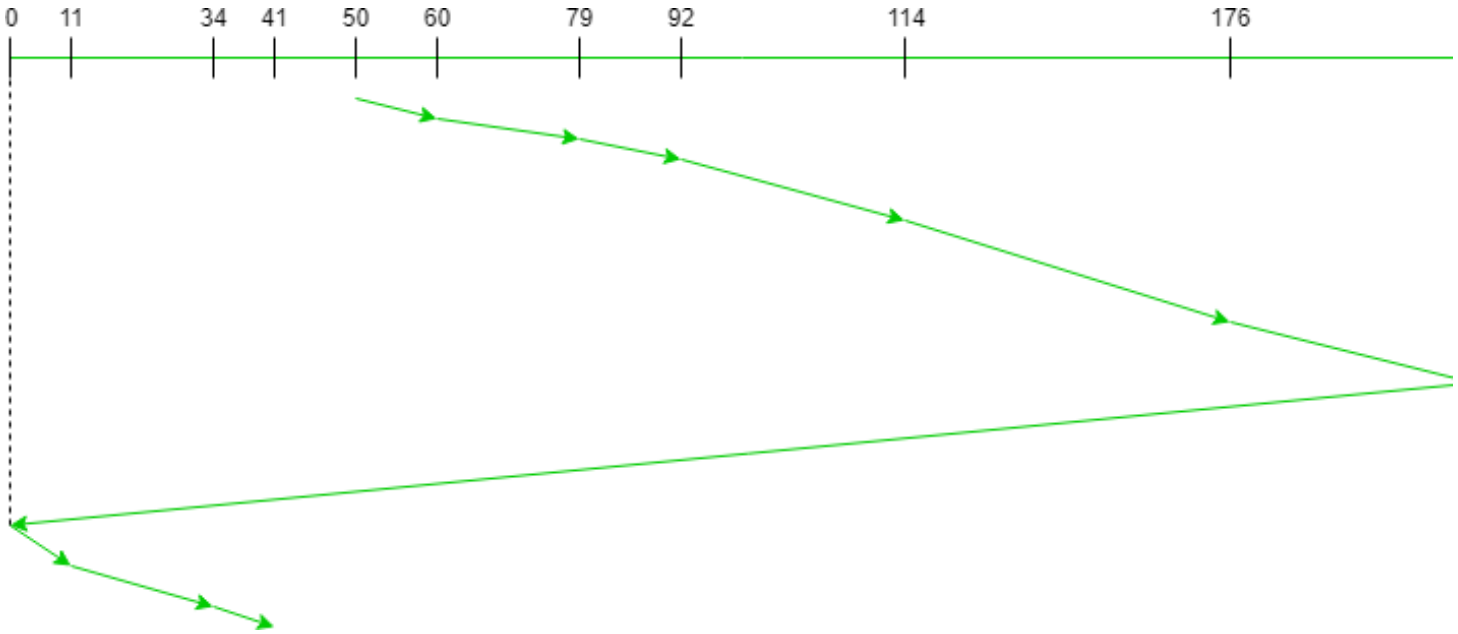
Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. The head services only in the right direction from 0 to size of the disk.
3. While moving in the left direction do not service any of the tracks.
4. When we reach at the beginning(left end) reverse the direction.
5. While moving in right direction it services all tracks one by one.
6. While moving in right direction calculate the absolute distance of the track from the head.
7. Increment the total seek count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach at right end of the disk.
10. If we reach at the right end of the disk reverse the direction and go to step 3 until all tracks in request array have not been serviced.

11. **Examples:**

12. **Input:**
13. Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
14. Initial head position = 50
15.
16. **Output:**
17. Initial position of head: 50
18. Total number of seek operations = 190
19. Seek Sequence is
20. 60
21. 79
22. 92
23. 114
24. 176
25. 199
26. 0
27. 11
28. 34
29. 41

The following chart shows the sequence in which requested tracks are serviced using SCAN.



Therefore, the total seek count is calculated as:

= (60-50)+(79-60)+(92-79) +(114-92)+(176-114)+(199-176)+(199-0)+(11-0)+(34-11)+(41-34)

LOOK Disk Scheduling Algorithm:
LOOK is the advanced version of SCAN (elevator) disk scheduling algorithm which gives slightly better seek time than any other algorithm in the hierarchy *(FCFS->SRTF->SCAN->C-SCAN->LOOK)*. The LOOK algorithm services request similarly as SCAN algorithm meanwhile it also "looks" ahead as if there are more tracks that are needed to be serviced in the same direction. If there are no pending requests in the moving direction the head reverses the direction and start servicing requests in the opposite direction.

The main reason behind the better performance of LOOK algorithm in comparison to SCAN is because in this algorithm the head is not allowed to move till the end of the disk.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. The intial direction in which head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction head is moving.
4. The head continues to move in the same direction untill all the request in this direction are not finished.
5. While moving in this direction calculate the absolute distance of the track from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.

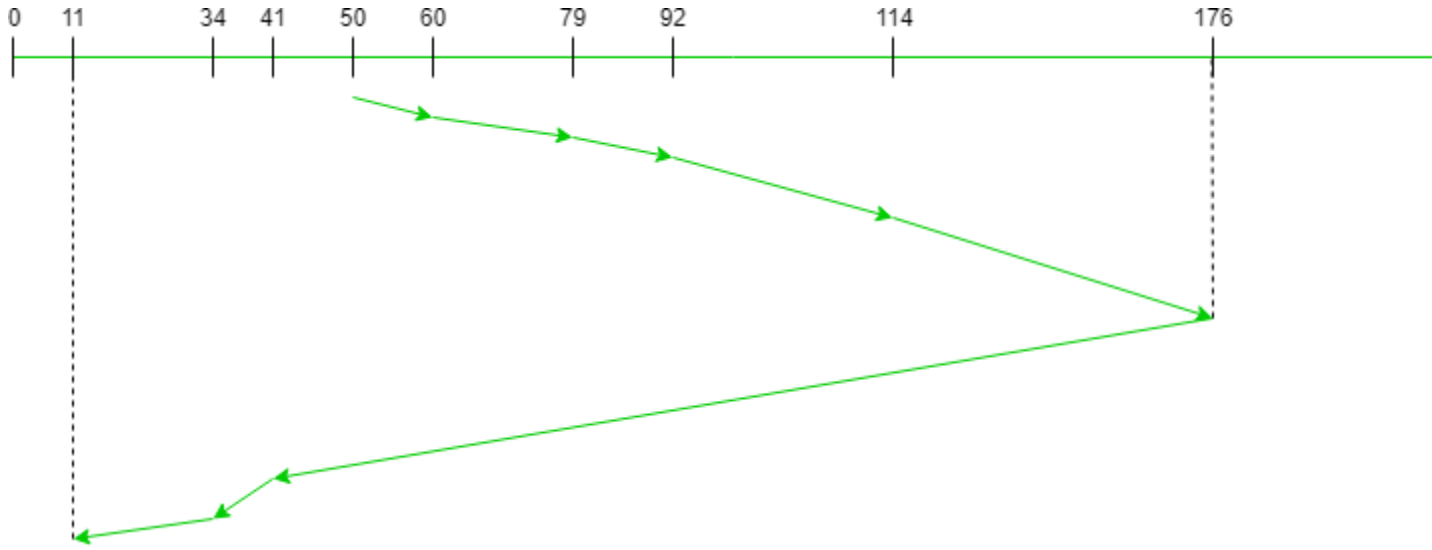
8. Go to step 5 until we reach at last request in this direction.
9. If we reach where no requests are needed to be serviced in this direction reverse the direction and go to step 3 until all tracks in request array have not been serviced.

Examples:

Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = right (We are moving from left to right)

Output:
Initial position of head: 50
Total number of seek operations = 291
Seek Sequence is
60
79
92
114
176
41
34
11

The following chart shows the sequence in which requested tracks are serviced using LOOK.



Therefore, the total seek count is calculated as:

= (60-50)+(79-60)+(92-79)+(114-92)+(176-114)+(176-41)+(41-34)+(34-11)

C-LOOK (Circular LOOK):

C-LOOK is an enhanced version of both **SCAN** as well as **LOOK** disk scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid starvation and services all the requests more uniformly, the same goes for C-LOOK.

In this algorithm, the head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

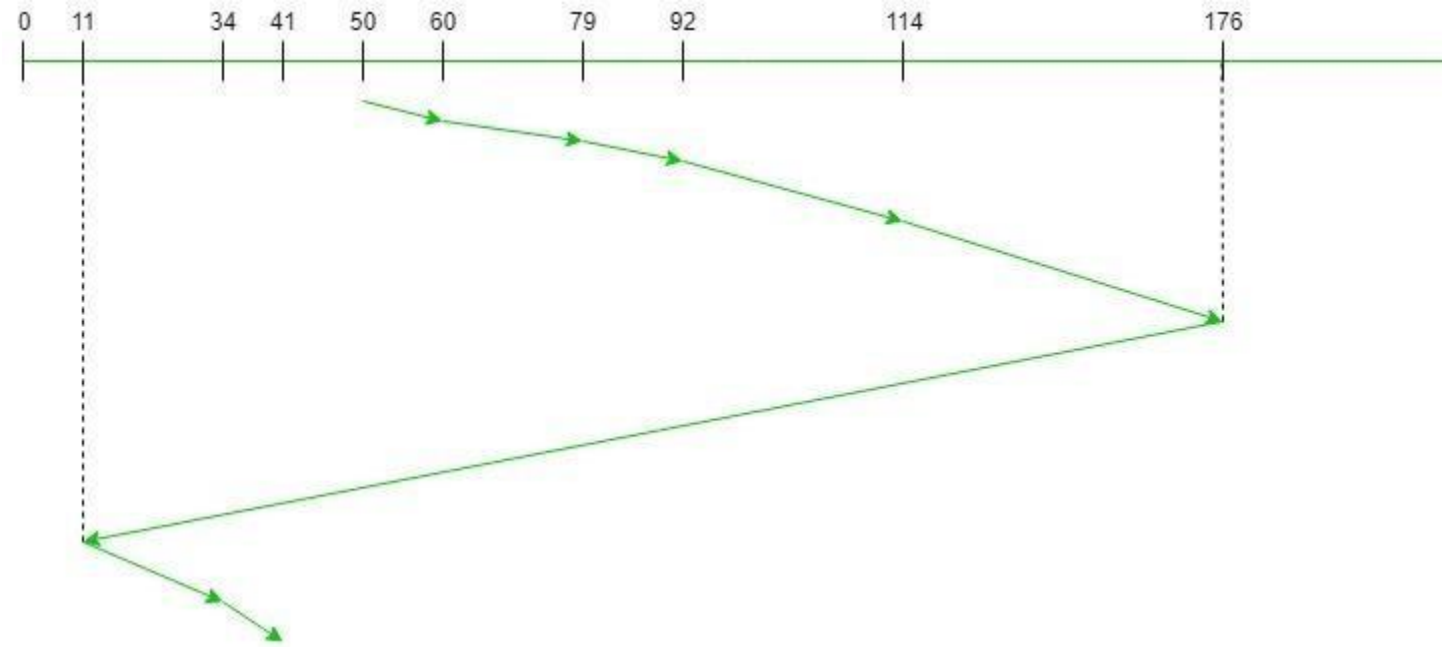
Algorithm-

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and **head** is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = right (We are moving from left to right)

Output:
Initial position of head: 50
Total number of seek operations = 156
Seek Sequence is
60
79
92
114
176
11
34
41

The following chart shows the sequence in which requested tracks are serviced using C-LOOK.



Therefore, the total seek count = (60 – 50) + (79 – 60) + (92 – 79) + (114 – 92) + (176 – 114) + (176 – 11) + (34 – 11) + (41 – 34) = 321

Disk Formatting: A new magnetic disk is a blank slate: It is just platters of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting (or physical formatting**). Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an error-correcting code (ECC). When the controller writes a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area.

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps. The first step is to partition the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files. After partitioning, the second step is logical formatting (or creation of a file system). In this step, the operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

Boot Block: For a computer to start running-for instance, when it is powered up or rebooted-it needs to have an initial program to run. This initial bootstrap program tends to be simple. Bootstrap is stored in read-only memory (ROM). This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And, since ROM is read only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips. For this reason, most systems store a tiny bootstrap loader program in the boot ROM, whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: A new version is simply written onto the disk. The full bootstrap program is stored in a partition called the boot blocks, at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

Bad Block: Because disks have moving parts and small tolerances, surface), they are prone to failure. Sometimes the failure is complete, and the disk needs to be replaced, and its contents restored from backup media to the new disk. More frequently, one or more sectors become defective. Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of ways.