**UNIT - 5**

**DISK SCHEDULING ALGORITHMS**

FCFS Disk Scheduling Algorithms

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if **First Come First Serve (FCFS)** disk scheduling algorithm is used.

**First Come First Serve (FCFS)**  
FCFS is the simplest [disk scheduling algorithm](https://www.geeksforgeeks.org/disk-scheduling-algorithms/). 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

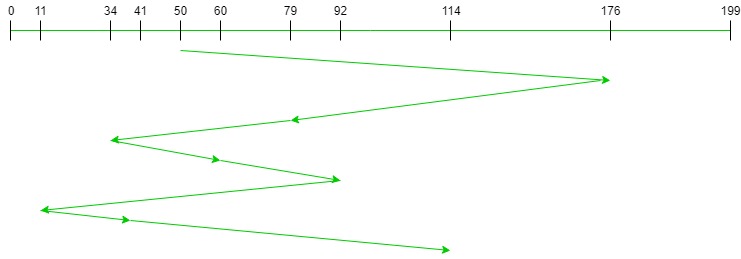
92

11

41

114

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



# SSTF disk scheduling algorithm

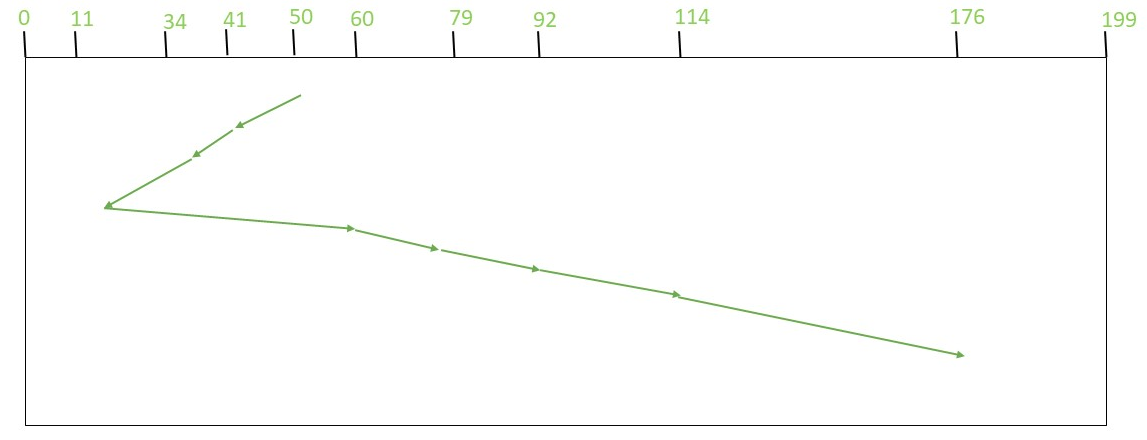
Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if **Shortest Seek Time First (SSTF)** is a disk scheduling algorithm is used.

**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 *minimise 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) Disk Scheduling Algorithms

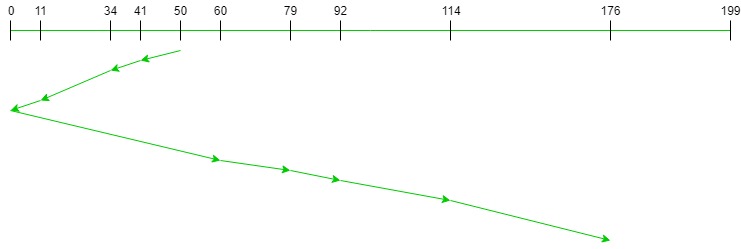
Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if SCAN disk scheduling algorithm is used.

**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.
9. **Example:**
10. **Input:**
11. Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
12. Initial head position = 50
13. Direction = left (We are moving from right to left)
14. **Output:**
15. Total number of seek operations = 226
16. Seek Sequence is
17. 41
18. 34
19. 11
20. 0
21. 60
22. 79
23. 92
24. 114
25. 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

# LOOK Disk Scheduling Algorithm

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if *LOOK* disk scheduling algorithm is used. Also, write a program to find the seek sequence using *LOOK* disk scheduling algorithm.

**LOOK Disk Scheduling Algorithm:**  
LOOK is the advanced version of [SCAN (elevator) disk scheduling algorithm](https://www.geeksforgeeks.org/scan-elevator-disk-scheduling-algorithms/) 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

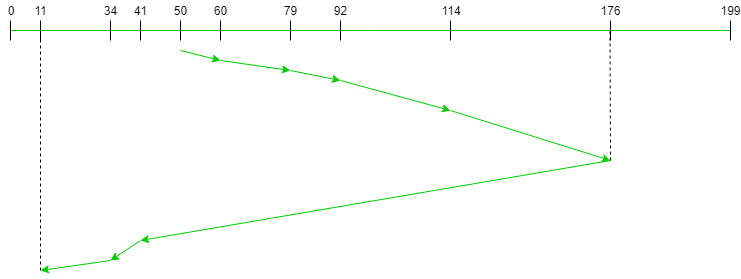
176

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 Disk Scheduling Algorithm

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if **C-LOOK** disk scheduling algorithm is used. Also, write a program to find the seek sequence using **C-LOOK** disk scheduling algorithm.

**C-LOOK (Circular LOOK) Disk Scheduling Algorithm:**  
**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.

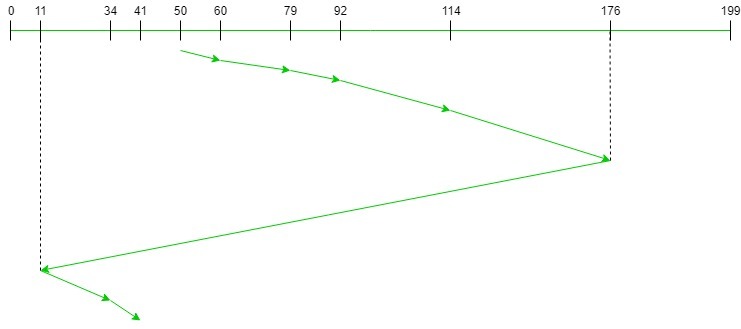
**Algorithm-**

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.

**Examples:**

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



# C-SCAN Disk Scheduling Algorithm

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if **C-SCAN** disk scheduling algorithm is used.

**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.

**Examples:**

**Input:**

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

**Output:**

Initial position of head: 50

Total number of seek operations = 190

Seek Sequence is

60

79

92

114

176

199

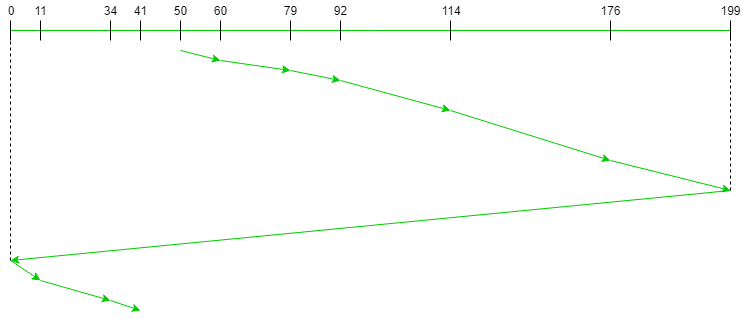
0

11

34

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)