Write a C program to simulate disk head movement in a database server using the following **disk scheduling algorithms**:
i. **C-SCAN (Circular SCAN)**
ii. **LOOK**
iii. **C-LOOK**

Disk Scheduling Algorithms

What is Disk Scheduling Algorithm?
A Process makes the I/O requests to the operating system to access the disk. Disk Scheduling Algorithm manages those requests and decides the order of the disk access given to the requests.

Why is the Disk Scheduling Algorithm needed?
Disk Scheduling Algorithms are needed because a process can make multiple I/O requests and multiple processes run at the same time. The requests made by a process may be located at different sectors on different tracks. Due to this, the seek time may increase more. These algorithms help in minimizing the seek time by ordering the requests made by the processes.

Important Terms related to Disk Scheduling Algorithms

1. **Seek Time:** As we know, the data may be stored on various blocks of disk. To access these data according to the request, the disk arm moves and finds the required block. The time taken by the arm in doing this search is known as "Seek Time"**.**
2. **Rotational Latency:** The required data block needs to move at a particular position from where the read/write head can fetch the data. So, the time taken in this movement is known as "Rotational Latency". This rotational time should be as less as possible so, the algorithm that will take less time to rotate will be considered a better algorithm.
3. **Transfer Time:** When a request is made from the user side, it takes some time to fetch these data and provide them as output. This time taken is known as "Transfer Time".
4. **Disk Access Time**: It is defined as the total time taken by all the above processes. Disk access time = (seek time + rotational latency time + transfer time)
5. **Disk Response Time:** The disk processes one request at a single time. So, the other requests wait in a queue to finish the ongoing process of request. The average of this waiting time is called "Disk Response Time".
6. **Starvation:** Starvation is defined as the situation in which a low-priority job keeps

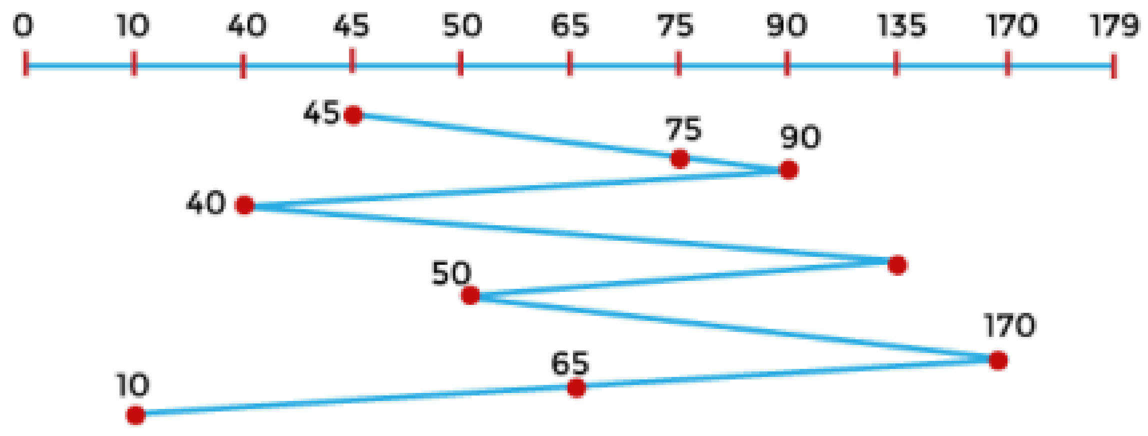
waiting for a long time to be executed. The system keeps sending high-priority jobs to the disk scheduler to execute first.

First Come First Serve (FCFS)

- FCFS stands for First-Come-First-Serve. It is a very easy algorithm among the all-disk scheduling algorithms.
- It is an OS disk scheduling algorithm that runs the queued requests and processes in the way that they arrive in the disk queue. It is a very easy and simple CPU scheduling algorithm.
- In this scheduling algorithm, the process which requests the processor first receives the processor allocation first. It is managed with a FIFO queue.

Example:
Let's take a disk with 180 tracks (0-179) and the disk queue having input/output requests in the following order: 75, 90, 40, 135, 50, 170, 65, 10. The initial head position of the Read/Write head is 45. Find the total number of track movements of the Read/Write head using the FCFS algorithm.

Solution:



FCFS Disk Scheduling Algorithm

Total head movements,
The initial head point is 45,
= (75-45) + (90-75) + (90-40) + (135-40) + (135-50) + (170-50) + (170-65) + (65-10)
= 30 + 15 + 50 + 95 + 85 + 120 + 105 + 55
= 555

Example 2 :

(NOTE: Please follow this output format for all the programs)

```
Enter the max range of disk
200
Enter the size of queue request
7
Enter the queue of disk positions to be read
82,170,43,140,24,16,190
Enter the initial head position 50
Total seek time is 642
Average seek time is 91.71
```

Advantages

- It is a very easy type of disk scheduling algorithm.
- It is easy to program.
- It provides a first come first served process.
- In FCFS, each process eventually has a chance to execute, therefore there is no starvation.

Disadvantages

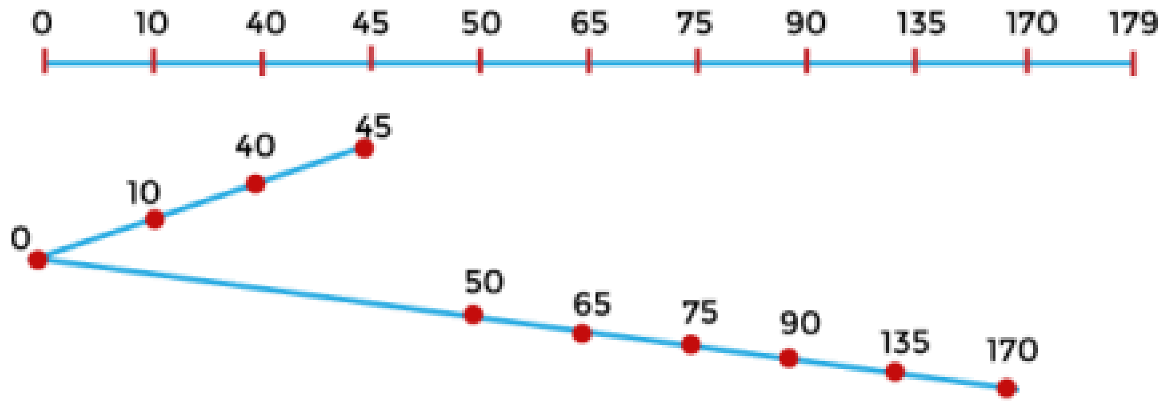
- It is not very efficient because of its simplicity.
- Its average waiting time is high.
- It is a Non-Preemptive CPU Scheduling Algorithm, which implies that once a process has been assigned to a CPU, it would never release the CPU until the process has completed executing.

<u>SCAN</u>

- It is also known as the Elevator algorithm.
- In this algorithm, the head may move in both directions, i.e., the disk arm begins to move from one end of the disk to the other end and servicing all requests until it reaches the other end of the disk.
- After reaching the other end, the head position direction is changed and further continues servicing the requests till the end of the disk.

Example:

Let's take a disk with 180 tracks (0-179) and the disk queue having input/output requests in the following order: 75, 90, 40, 135, 50, 170, 65, 10. The initial head position of the Read/Write head is 45 and will move on the left-hand side. Find the total number of track movements of the Read/Write head using the SCAN algorithm.



SCAN Disk Scheduling Algorithm

Solution:
Initial head point is 45,
= (45-40) + (40-10) + (10-0) + (50-0) + (65-50) + (75-65) + (90-75) + (135-90) + (170-135)
= 5 + 30 +10 +50 +15 + 10 +15 + 45 + 35
= 215

Advantages

- It is easy to use and understand.
- In the SCAN disk scheduling algorithm, low variance happens in the waiting time and response time.
- Starvation is avoided.

Disadvantage

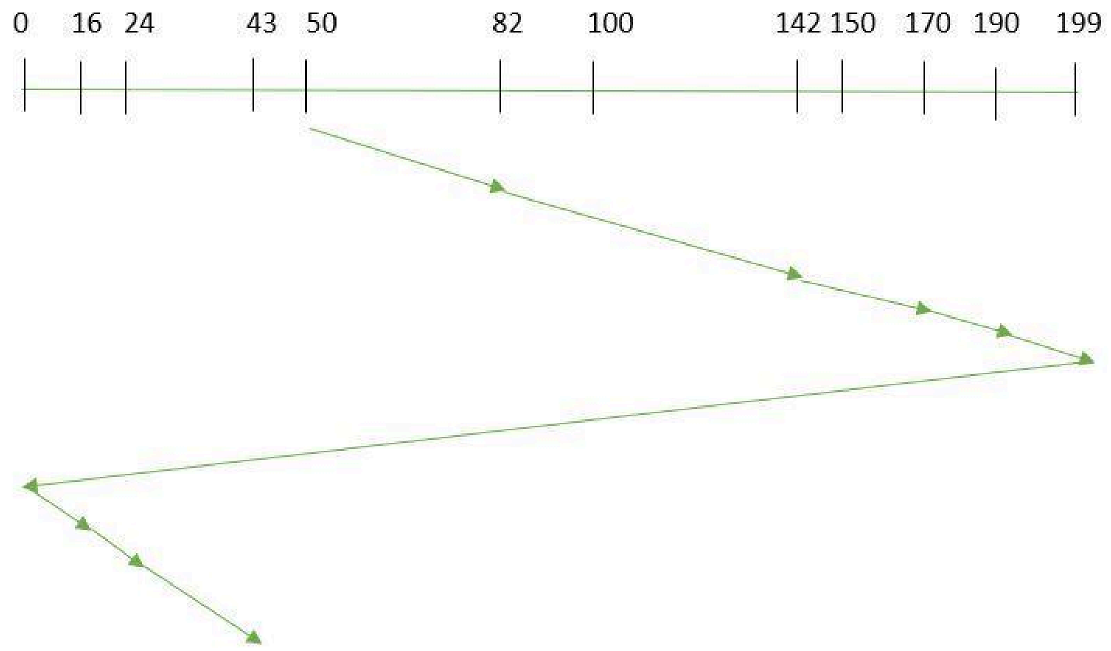
- If no requests remain to be serviced, the head moves till the end of the disk.

C-SCAN

In the 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.

These situations are avoided in the CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

Example:



Circular SCAN

Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50) + (199-0) + (43-0) = 391

Advantages of C-SCAN Algorithm:

- Eliminates starvation of requests
- Better performance for heavy disk load
- More fair than SCAN algorithm

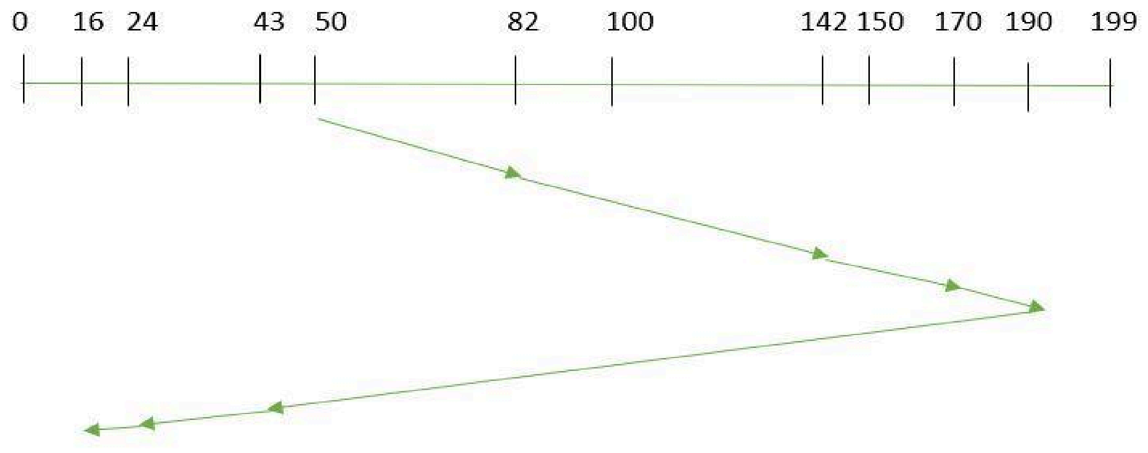
Disadvantages of C-SCAN Algorithm:

- Extra head movement due to return to start
- Increased seek time compared to SCAN in light load
- More overhead because of circular movement

LOOK

LOOK Algorithm 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.

Example:



LOOK Algorithm

Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

= (190-50) + (190-16) = 314

Advantages

- Reduced Unnecessary Movement: The disk arm only goes as far as the last request in each direction, avoiding travel to the disk's physical end (unlike SCAN).
- Faster Response: Less head movement leads to quicker service for requests.

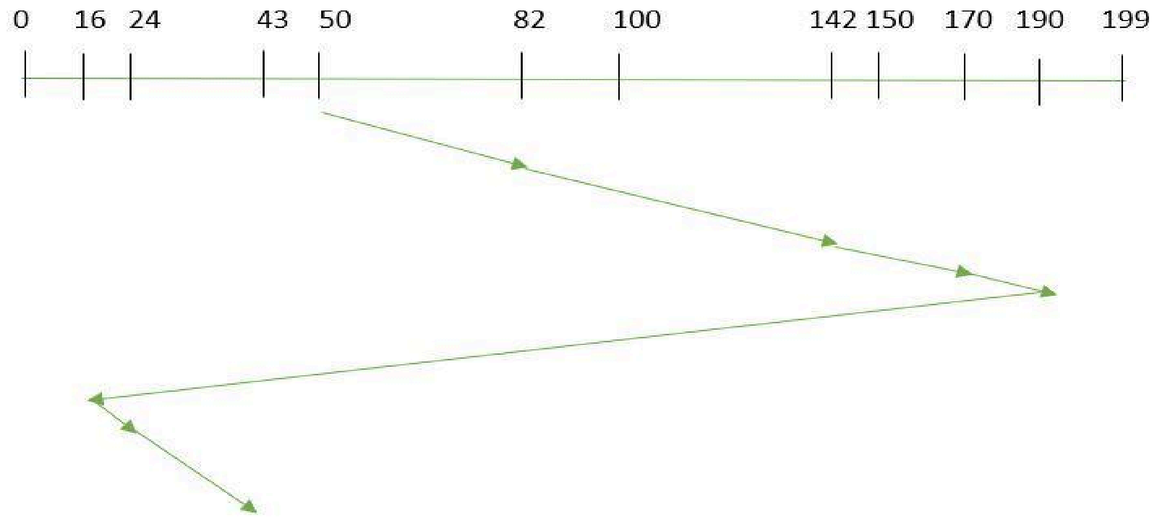
Disadvantages of LOOK Algorithm:

- Waiting time can still be high for some requests
- Performance depends on request distribution
- Not completely fair to newly arrived requests

C-LOOK

As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the 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.

Example: Suppose the requests to be addressed are-82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value"



C-LOOK

So, the total overhead movement (total distance covered by the disk arm) is calculated as

= (190-50) + (190-16) + (43-16) = 341

Advantages:

- Uniform Wait Time: Requests are serviced in a circular manner, so waiting times are more predictable and fair.
- Reduced Head Movement: The arm only goes as far as the last request in one direction, then jumps back, saving time compared to C-SCAN.

Disadvantages of the C-LOOK Algorithm:

- Extra head movement due to circular scanning
- Increased seek time compared to LOOK under light load
- Newly arrived requests may have to wait for a full cycle

Pseudocode for C-SCAN (Circular SCAN)

```
FUNCTION C_SCAN(RequestQueue, N, Head)
    Set TotalMovement ← 0
    Set Current ← Head

    FOR each request i in RequestQueue where i ≥ Head (in
ascending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    // Move to end of disk
    TotalMovement ← TotalMovement + |Current - MAX|
    Current ← MAX

    // Jump from MAX to 0
    TotalMovement ← TotalMovement + |MAX - 0|
    Current ← 0
    FOR each request i in RequestQueue where i < Head (in
ascending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    Display TotalMovement
END FUNCTION
```

Pseudocode for LOOK

```
FUNCTION LOOK(RequestQueue, N, Head)
    Set TotalMovement ← 0
    Set Current ← Head

    // Move towards higher tracks first
    FOR each request i in RequestQueue where i ≥ Head (in
ascending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    // Reverse direction at last request (not disk end)
    FOR each request i in RequestQueue where i < Head (in
descending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    Display TotalMovement
END FUNCTION
```

Pseudocode for C-LOOK

```
FUNCTION C_LOOK(RequestQueue, N, Head)
    Set TotalMovement ← 0
    Set Current ← Head

    FOR each request i in RequestQueue where i ≥ Head (in
ascending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    // Jump to lowest request (circular)
    Find MinRequest in RequestQueue
    TotalMovement ← TotalMovement + |Current - MinRequest|
    Current ← MinRequest

    FOR each request i in RequestQueue where i < Head (in
ascending order)
        TotalMovement ← TotalMovement + |Current - i|
        Current ← i
    END FOR

    Display TotalMovement
END FUNCTION
```