1.Compare the functionalities of FCFS, SSTF, C-SCAN and C-LOOK with example.

FCFS, SSTF, C-SCAN and C-LOOK are four disk scheduling algorithms that are used to optimize the performance of disk drives by minimizing the seek time and rotational latency. Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. 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. [The total distance (in cylinders) that the disk arm moves to satisfy all the pending requests is one of the measures of the efficiency of these algorithms1](https://www.geeksforgeeks.org/disk-scheduling-algorithms/)[2](https://www.geeksforgeeks.org/fcfs-disk-scheduling-algorithms/)[3](https://www.cs.cornell.edu/courses/cs414/2007sp/homework/hw4_soln.pdf)[4](https://github.com/cnydo/DiscAlgs).

**FCFS**

FCFS (First Come First Serve) is the simplest disk scheduling algorithm. In FCFS, the requests are addressed 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.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Simple and easy to implement | Does not try to optimize seek time |
| Every request gets a fair chance | May not provide the best possible service |
| No indefinite postponement |  |

Example: Suppose the order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50 So, total overhead movement (total distance covered by the disk arm) : = (82-50)+ (170-82)+ (170-43)+ (140-43)+ (140-24)+ (24-16)+ (190-16) =642

**SSTF**

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** | **Disadvantages** |
| --- | --- |
| Reduces the seek time | Complex to implement |
| Improves the performance | May cause starvation for some requests |
| Suitable for small number of requests |  |

Example: Suppose again that order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50 So, total overhead movement (total distance covered by the disk arm) : = (82-50)+ (82-43)+ (43-24)+ (24-16)+ (140-16)+ (140-170)+ (190-170) = 236

**C-SCAN**

C-SCAN (Circular SCAN) is a variant of SCAN algorithm. In SCAN algorithm, the disk arm again scans the path that has been scanned before going to other end of disk. That means SCAN algorithm unnecessarily scans some portion of disk twice. So C-SCAN algorithm overcomes this problem by scanning each portion of disk only once. In C-SCAN algorithm, when head reaches to one end it directly goes to other end without serving any request in between.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Reduces scan time | Complex to implement |
| Provides more uniform service | May cause long waiting time for some requests |
| Suitable for large number of requests |  |

Example: Suppose again that order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50 So, total overhead movement (total distance covered by the disk arm) : = (82-50)+ (190-82)+ (199-190)+ (199-16)+ (24-16)+ (43-24)+ (140-43) = 341

**C-LOOK**

C-LOOK (Circular LOOK) is another variant of SCAN algorithm. In LOOK algorithm, unlike SCAN algorithm stops its head movement when it finds no more requests further in its direction of motion. In C-LOOK algorithm also head moves only in one direction like C-SCAN but head movement stops when last request has been serviced in direction of motion and then head directly jumps towards first request in opposite direction.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Reduces scan time | Complex to implement |
| Provides more uniform service | May cause long waiting time for some requests |
| Suitable for large number of requests |  |

Example: Suppose again that order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is: 50 So, total overhead movement (total distance covered by the disk arm) : = (82-50)+ (190-82)+ (16-190)+ (16-24)+ (43-24)+ (140-43) = 273

2. What are files and explain the access methods for files?

Files are collections of data that are stored on secondary storage devices such as hard disks, flash drives, optical disks, etc. Files can contain different types of data such as text, images, audio, video, etc. [Files have names and attributes that identify them and allow users and programs to access them1](https://www.geeksforgeeks.org/file-access-methods-in-operating-system/)[2](https://www.javatpoint.com/os-file-access-methods).

File access methods define how data is accessed and modified within a file. There are different file access methods with their own set of strengths and limitations. The three primary file access methods are sequential access, random access, and direct access[1](https://www.geeksforgeeks.org/file-access-methods-in-operating-system/)[2](https://www.javatpoint.com/os-file-access-methods)[3](https://www.tutorialspoint.com/file-access-method).

**Sequential access**

Sequential access is the simplest and most common file access method. In sequential access, data is read or written in a linear order, one record after another. This mode of access is suitable for applications that require processing all the records in a file in a specific order. For example, editors, compilers, backup programs, etc. use sequential access.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Simple and easy to implement | Slow access to specific records |
| Efficient for reading large files | Difficult to insert or delete records |
| No extra space required for storing pointers or addresses | No grouping or sorting of records |

**Random access**

Random access is a file access method that allows accessing any record in a file directly without reading the previous records. This mode of access is suitable for applications that require frequent updates or modifications to specific records in a file. For example, databases, spreadsheets, indexing programs, etc. use random access.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Fast access to specific records | Complex to implement |
| Easy to insert or delete records | Requires extra space for storing pointers or addresses |
| Allows grouping or sorting of records | Prone to fragmentation or corruption |

**Direct access**

Direct access is a file access method that combines the advantages of sequential and random access methods. In direct access, data is read or written in any order but with some fixed pattern. This mode of access is suitable for applications that require both sequential and random processing of records in a file. For example, multimedia files, hashing programs, etc. use direct access.

| **Advantages** | **Disadvantages** |
| --- | --- |
| Flexible and efficient for accessing records | Complex to implement |
| Easy to insert or delete records | Requires extra space for storing pointers or addresses |
| Allows grouping or sorting of records | Prone to fragmentation or corruption |

Here is a neat sketch of the three file access methods:

Sequential access:

+-----------------+

| File |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| R1+---->+ R2+----> Records

+---+ +---+

/ \ / \

+---+ +---+

| R3+---->+ R4+---->

+---+ +---+

Random access:

+-----------------+

| File |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| R1<---->+ R2<----> Records

+---+ +---+

/ \ / \

+---+ +---+

| R3<---->+ R4<---->

+---+ +---+

Direct access:

+-----------------+

| File |

+-----------------+

/ \

/ \

/ \

+---+ +---+

| R1<---->+ R2<----> Records

+---+ +---+

/ \ / \

+-+-+ +-+-+

|R3 +---->+R4 +

+-+-+ +-+-+