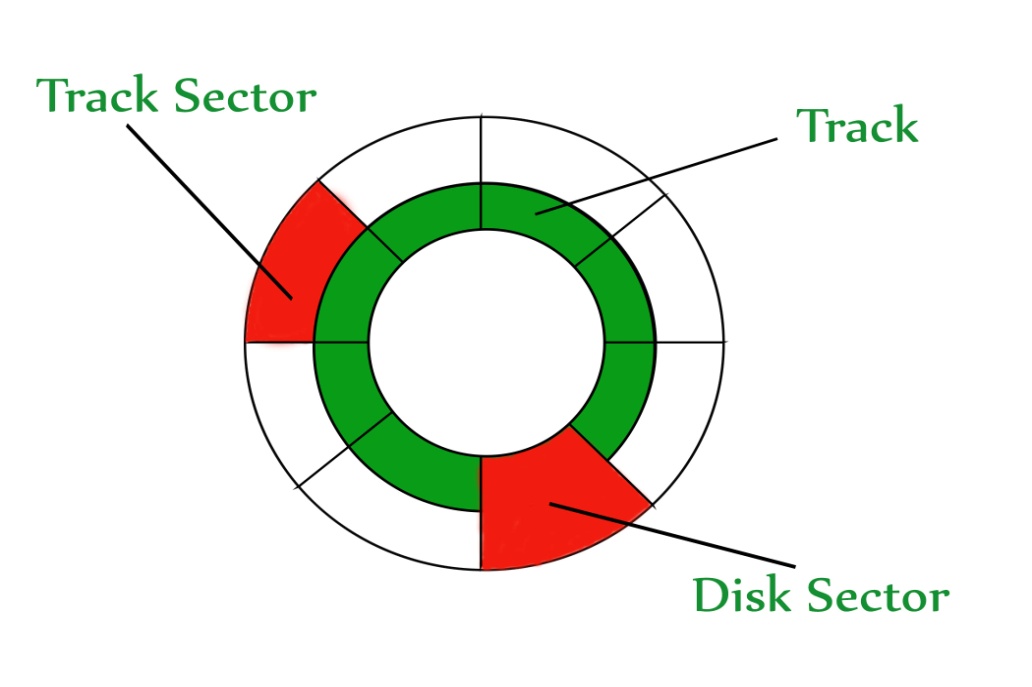
Hard Disk Drive (HDD) Secondary memory

Last Updated: 14-08-2019

A hard disk is a memory storage device which looks like this:



The disk is divided into **tracks**. Each track is further divided into **sectors**. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage capacity. This is because the storage density is high in sectors of the inner tracks where as the bits are sparsely arranged in sectors of the outer tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.

Read-Write(R-W) head moves over the rotating hard disk. It is this Read-Write head that performs all the read and write operations on the disk and hence, position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here:

1. **Seek time –** The time taken by the R-W head to reach the desired track from it’s current position.
2. **Rotational latency –** Time taken by the sector to come under the R-W head.
3. **Data transfer time –** Time taken to transfer the required amount of data. It depends upon the rotational speed.
4. **Controller time –** The processing time taken by the controller.
5. **Average Access time –** seek time + Average Rotational latency + data transfer time + controller time.

**Note:**Average Rotational latency is mostly 1/2\*(Rotetional latency).

In questions, if the seek time and controller time is not mentioned, take them to be zero.

If the amount of data to be transferred is not given, assume that no data is being transferred. Otherwise, calculate the time taken to transfer the given amount of data.

The average of rotational latency is taken when the current position of R-W head is not given. Because, the R-W may be already present at the desired position or it might take a whole rotation to get the desired sector under the R-W head. But, if the current position of the R-W head is given then the rotational latency must be calculated.

**Example –**  
Consider a hard disk with:  
4 surfaces  
64 tracks/surface  
128 sectors/track  
256 bytes/sector

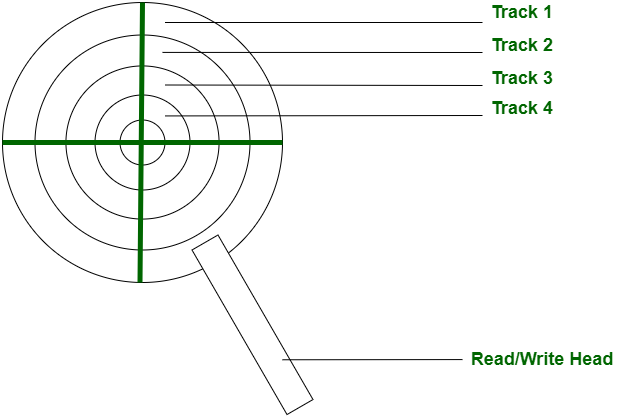
1. What is the capacity of the hard disk?  
   Disk capacity = surfaces \* tracks/surface \* sectors/track \* bytes/sector  
   Disk capacity = 4 \* 64 \* 128 \* 256  
   Disk capacity = 8 MB
2. The disk is rotating at 3600 RPM, what is the data transfer rate?  
   60 sec -> 3600 rotations  
   1 sec -> 60 rotations  
   Data transfer rate = number of rotations per second \* track capacity \* number of surfaces (since 1 R-W head is used for each surface)  
   Data transfer rate = 60 \* 128 \* 256 \* 4  
   Data transfer rate = 7.5 MB/sec
3. The disk is rotating at 3600 RPM, what is the average access time?  
   Since, seek time, controller time and the amount of data to be transferred is not given, we consider all the three terms as 0.  
   Therefore, Average Access time = Average rotational delay  
   Rotational latency => 60 sec -> 3600 rotations  
   1 sec -> 60 rotations  
   Rotational latency = (1/60) sec = 16.67 msec.  
   Average Rotational latency = (16.67)/2  
   = 8.33 msec.  
   Average Access time = 8.33 msec.

Difference between Seek Time and Disk Access Time in Disk Scheduling

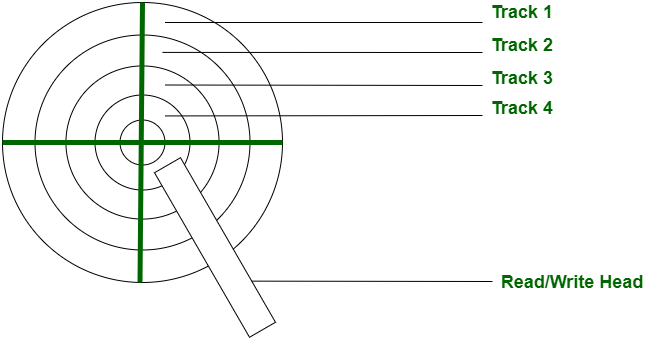
Last Updated: 01-04-2020

**Seek Time:**  
A [disk](https://www.geeksforgeeks.org/hard-disk-drive-hdd-secondary-memory/) is divided into many circular tracks. Seek Time is defined as the time required by the read/write head to move from one track to another.

Example,  
Consider the following diagram, the read/write head is currently on track 1.



Now, on the next read/write request, we may want to read data from Track 4, in this case, our read/write head will move to track 4. The time it will take to reach track 4 is the **seek time**.



**Disk Access Time:**  
Disk Access Time is defined as the total time required by the computer to process a read/write request and then retrieve the required data from the disk storage.

Disk Access Time is divided into 2 parts:

1. Access Time
2. Data Transfer Time

Disk Access Time = Access Time + Data Transfer Time

**1. Access Time:**  
Access Time is defined as the setup time before the actual data transfer takes place.  
For example, the read/write head is on track 1 but we need to read data from another track or segment. Thus, the read/write head will move to the data block location before the actual transfer can take place. This delay is called Access Time.

Access Time is calculated by summation of the following:

**(a).** Seek Time

**(b).** Rotational Latency

**(c).** Command Processing Time

**(d).** Settle Time

These are explained as following below in brief.

* **(a). Seek Time –**  
  It is the time required by the read/write head to move from the current track to the requested track.
* Seek Time

= (Number of tracks/cylinders crossed) \* (Time to cross one track/cylinder)

* **(b). Rotational Latency –**  
  It is the time required by the read/write head to move from the current sector to the requested sector.
* Rotational Latency

= (Angle by which disk is rotated) / (Angular Frequency)

* **(c). Command Processing Time –**  
  It is the time required by the disk device to process the command and establish a connection between the various components of the disk device to read/write data. It is due to the internal circuitry.
* **(d). Settle Time –**  
  Settle Time is the time required by read/write head to stop vibrating.

**Note:** Command Processing Time and Settle Time are not normally mentioned in numerical question. We take them as zero.

**2. Data Transfer Time:**  
Data Transfer Time is defined as the time required to transfer data between the system and the disk.  
Data Transfer Time is of two types:

**(a).** Internal Transfer Rate

**(b).** External Transfer Rate

These are explained as following below in brief.

* **(a). Internal Transfer Rate –**  
  It is defined as the time required to move data between the disk surface and hard disk cache.
* **(b). External Transfer Rate –**  
  It is defined as the time required to move data between the hard disk cache and the system.

Let’s see the difference between Seek Time and Disk Access Time:

| **S.NO.** | **SEEK TIME** | **DISK ACCESS TIME** |
| --- | --- | --- |
| 1 | It is the time required by read/write head to move from one track to other. | It is the time required by the computer to process a read/write request and retrieve the required data. |
| 2 | It is always less than Disk Access Time. Since, it is a sub part of Disk Access time. | It is very large compared to Seek time. |
| 3 | It doesn’t consider transfer of data. | It considers the time required to transfer data.c |

# FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

## Disadvantages

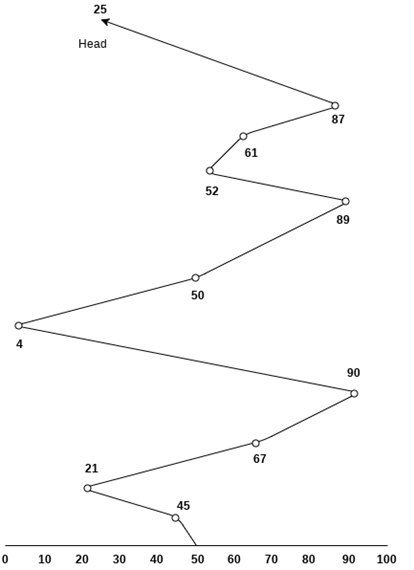
* The scheme does not optimize the seek time.
* The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

### Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

### Solution



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

# SSTF Scheduling Algorithm

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.

## Disadvantages

* It may cause starvation for some requests.
* Switching direction on the frequent basis slows the working of algorithm.
* It is not the most optimal algorithm.

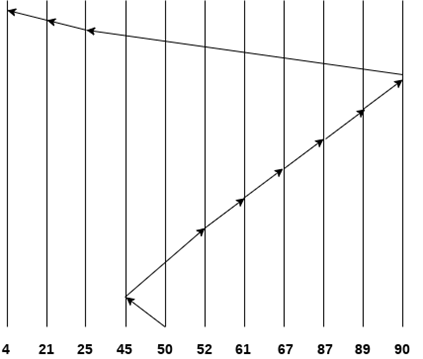
### Example

Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

### Solution:



Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

[**next →**](https://www.javatpoint.com/os-look-and-c-look-scheduling)[**← prev**](https://www.javatpoint.com/os-sstf-scheduling-algorithm)

# SCAN and C-SCAN algorithm

## Scan Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.

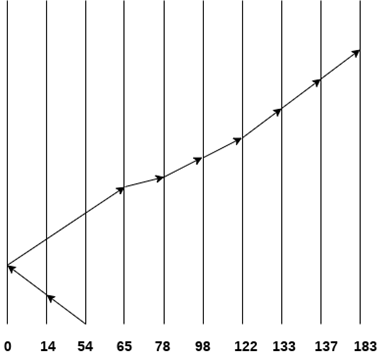
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

## C-SCAN algorithm

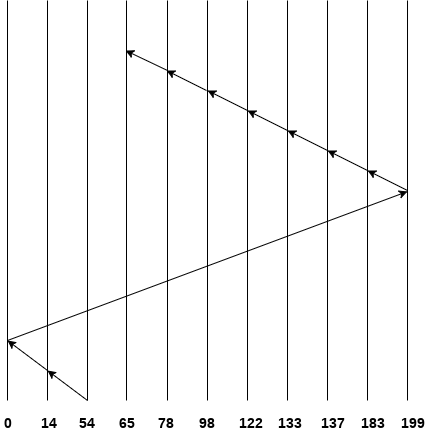
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

# Look Scheduling

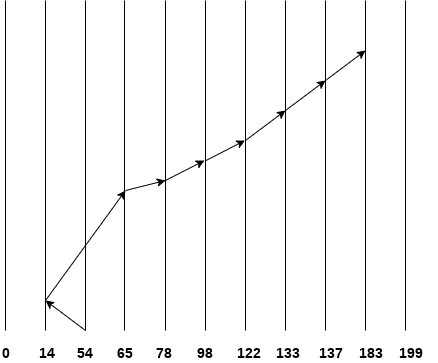
It is like SCAN scheduling Algorithm to some extant except the difference that, in this scheduling algorithm, the arm of the disk stops moving inwards (or outwards) when no more request in that direction exists. This algorithm tries to overcome the overhead of SCAN algorithm which forces disk arm to move in one direction till the end regardless of knowing if any request exists in the direction or not.

### Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using LOOK scheduling.



Number of cylinders crossed = 40 + 51 + 13 + +20 + 24 + 11 + 4 + 46 = 209

## C Look Scheduling

C Look Algorithm is similar to C-SCAN algorithm to some extent. In this algorithm, the arm of the disk moves outwards servicing requests until it reaches the highest request cylinder, then it jumps to the lowest request cylinder without servicing any request then it again start moving outwards servicing the remaining requests.

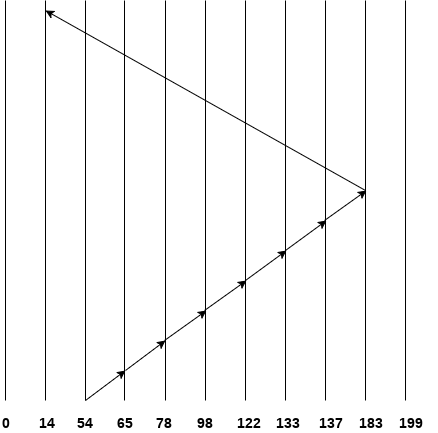
It is different from C SCAN algorithm in the sense that, C SCAN force the disk arm to move till the last cylinder regardless of knowing whether any request is to be serviced on that cylinder or not.

### Example

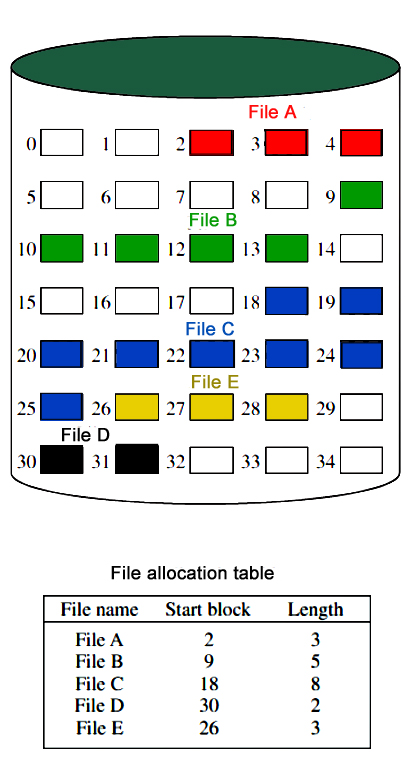
Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C LOOK scheduling.



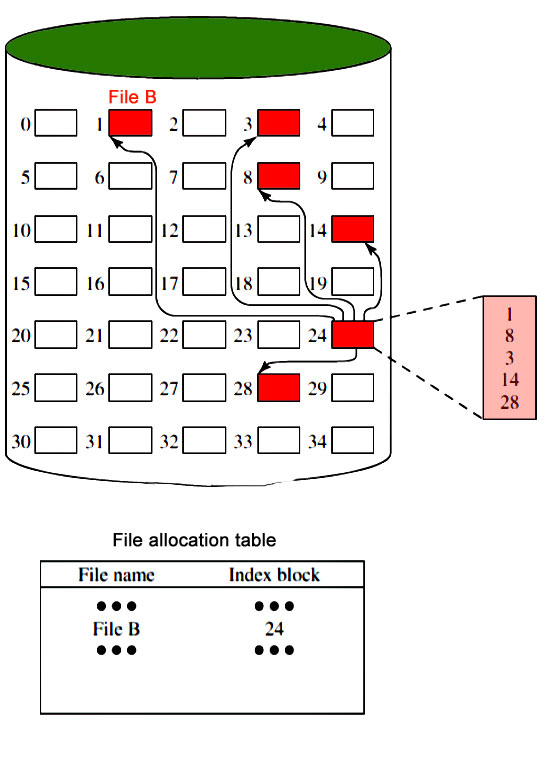
Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

**FILE ALLOCATION METHODS**  
**1. Continuous Allocation:** A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.  
[](https://media.geeksforgeeks.org/wp-content/uploads/directory-file-allocation.jpg)  
  
**Disadvantage**

* External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. Compaction algorithm will be necessary to free up additional space on disk.
* Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

**2. Linked Allocation(Non-contiguous allocation) :** Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.  
  
**Disadvantage:**

* Internal fragmentation exists in last disk block of file.
* There is an overhead of maintaining the pointer in every disk block.
* If the pointer of any disk block is lost, the file will be truncated.
* It supports only the sequencial access of files.

**3. Indexed Allocation:**  
It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.  
[](https://media.geeksforgeeks.org/wp-content/uploads/directory-indexing.jpg)