Write four necessary and sufficient conditions of a deadlock to occur in a system. What is the minimum number of processes that can be in deadlock state at a time?

The four necessary and sufficient conditions of a deadlock to occur in a system:

1. **Mutual exclusion:** Each resource can be held by only one process at a time.
2. **Hold and wait:** A process can hold some resources and be waiting to acquire additional resources that are currently being held by other processes.
3. **No preemption:** A resource can only be released voluntarily by the process that holds it.
4. **Circular wait:** There is a set of processes P0, P1, ..., Pn such that:

- Process P0 is waiting for a resource held by process P1.
- Process P1 is waiting for a resource held by process P2.
- ...
- Process Pn-1 is waiting for a resource held by process P0.

The minimum number of processes that can be in deadlock state at a time is **2**. This is because a deadlock requires a circular wait, which means that there must be at least two processes involved.

In a priority driven CPU scheduling, priority of a process is proportional to its waiting time in the ready queue. Suggest an equivalent processor scheduling policy which will be equivalent to the given priority scheduling. Justify your answer.

Round Robin Scheduling

In round robin scheduling, each process is given a time quantum to execute. When the time quantum expires, the process is preempted and the next process is scheduled. The process that was preempted is placed at the end of the ready queue. This process will be scheduled again once it reaches the front of the ready queue.

What do you mean by internal and external fragmentation of main memory? Does segmentation with paging removes all types of fragmentation?

Internal fragmentation occurs when a process is allocated a memory block that is larger than the process actually needs. This can happen because memory is often allocated in fixed-sized blocks, and processes may not be able to use all of the space in a block.

External fragmentation occurs when there are holes of unused memory scattered throughout the main memory. This can happen when a process is removed from memory, and the space it was using is not immediately allocated to another process.

Segmentation divides memory into logical segments, and each process is allocated a set of segments. This allows processes to be allocated memory in different sizes, and it can help to reduce internal fragmentation.

Paging divides memory into fixed-sized pages, and each process is allocated a set of pages. When a process is not using a page, it can be swapped out to disk. This can help to reduce external fragmentation.

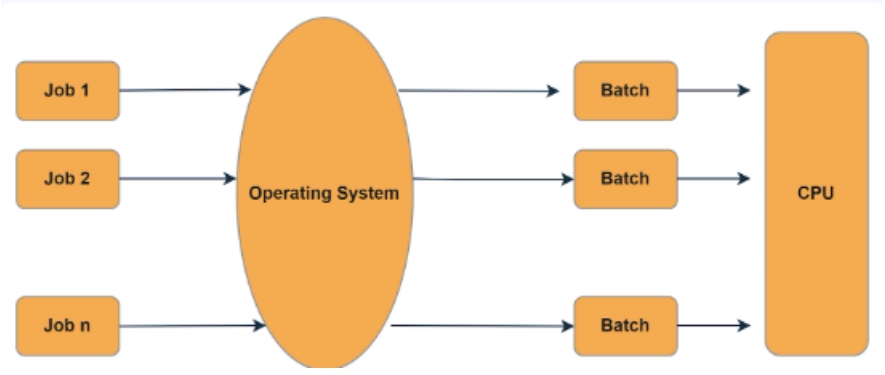
However, segmentation and paging cannot completely eliminate fragmentation. Internal fragmentation will always occur because of the fixed-sized pages, and external fragmentation can occur if processes are frequently removed from memory.

Differentiate parallel system with distributed system. Explain the concept of batch processing system and how is it helpful in maximizing resource utilization?

Parallel System	Distributed System
A parallel system is a system that has multiple processors that are connected to a shared memory.	A distributed system is a system that is made up of multiple computers that are connected over a network.
The processors access the same data and share resources.	Each computer has its own memory and resources.
Typically used for applications that require a lot of processing power, such as scientific computing applications.	Typically used for applications that require a lot of data or that need to be accessible from multiple locations.
It is also known as a tightly coupled system.	It is also known as loosely coupled systems.
All processors share a single master clock for synchronization.	It uses various synchronization algorithms.
E.g. Hadoop, Apache Cassandra	E.g. High-Performance Computing clusters

Here are the concepts of batch processing system and how it is helpful in maximizing resource utilization:

A **batch operating system** is an operating system that uses batch processing. Users don't directly interact with the computer in a batch operating system. Jobs with similar types are grouped and processed as a batch to save time. The operator groups the programs with similar requirements into sets or batches for efficient execution.



Batch operating systems can be very helpful in maximizing resource utilization in a computing environment. By grouping similar jobs together, scheduling jobs based on priority, and using a job

queue, batch operating systems can ensure that all available resources are used to process jobs efficiently.

Explain the concept of context switching with suitable example. What is a process control block of a process and how is it useful in processor scheduling?

Context switching is the process of storing the state of a running process so that it can be resumed later, and then switching to another process. This allows multiple processes to share a single CPU, which is essential for multitasking operating systems.

For example, let's say you are running two processes on your computer: a web browser and a word processor. The web browser is currently running, and you are typing a document in the word processor. If you receive an email notification, the operating system will interrupt the web browser and switch to the email program. The operating system will save the state of the web browser, including the current cursor position, the open tabs, and any other data that the web browser needs to keep track of. Then, the operating system will load the state of the email program, and you will be able to read your email.

Process Control block (PCB) is a data structure that stores information of a process. PCB is unique for every process which consists of various attributes such as process ID, priority, registers, program counters, process states, list of open files, etc.

When the operating system decides to switch from one process to another, it saves the current process's context in the PCB and loads the next process's context from its PCB. PCB ensures that the process can resume from where it left off, without losing any critical information.

What do you mean by race condition? Explain with suitable example. Further, write three conditions that need to be satisfied for an ideal solution to the critical section problem.

A race condition is a problem that occurs in an operating system (OS) where two or more processes or threads are executing concurrently. The outcome of their execution depends on the order in which they are executed.

For example:
If two threads are simultaneously accessing and changing the same shared resource, such as a variable or a file, the final state of that resource depends on the order in which the threads execute. If the threads are not correctly synchronized, they can overwrite each other's changes, causing incorrect results or even system crashes.

Three must rules which must enforce by critical section are:

- **Mutual Exclusion**: Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.
- **Progress**: Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.
- **Bounded Waiting**: Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

What do you mean by page fault and how is it handled in the system? Explain Optimal, LRU and second chance LRU approximation methods of page replacement with suitable examples.

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.

Page Fault Handling

1. At first, an internal table is created to process whether the reference was **valid or invalid** memory access.
2. The system gets terminated if the reference becomes invalid, if not, the page will be paged in.
3. After checking for validity, the free-frame list searches the system for a free frame.
4. The disk operation will be scheduled to get the required page from the disk.
5. The page table of the process will be updated with a new frame number, and the invalid bit will be changed after the completion of the **I/O operation**. Now it is a valid page reference.
6. Restart these steps upon finding any more page faults.

Optimal Page Replacement: In this algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future.

Example:

Let's take the page reference string 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames

- Initially, since all the slots are empty, **pages 3, 1, 2** cause a page fault and take the empty slots.

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, it is not in the memory, so a page fault occurs and **2 is removed** as it is not going to be used again.

Page faults = 4

- When page 5 comes, it is also not in the memory and causes a page fault. Similar to above **6 is removed** as it is not going to be used again.

Page faults = 5

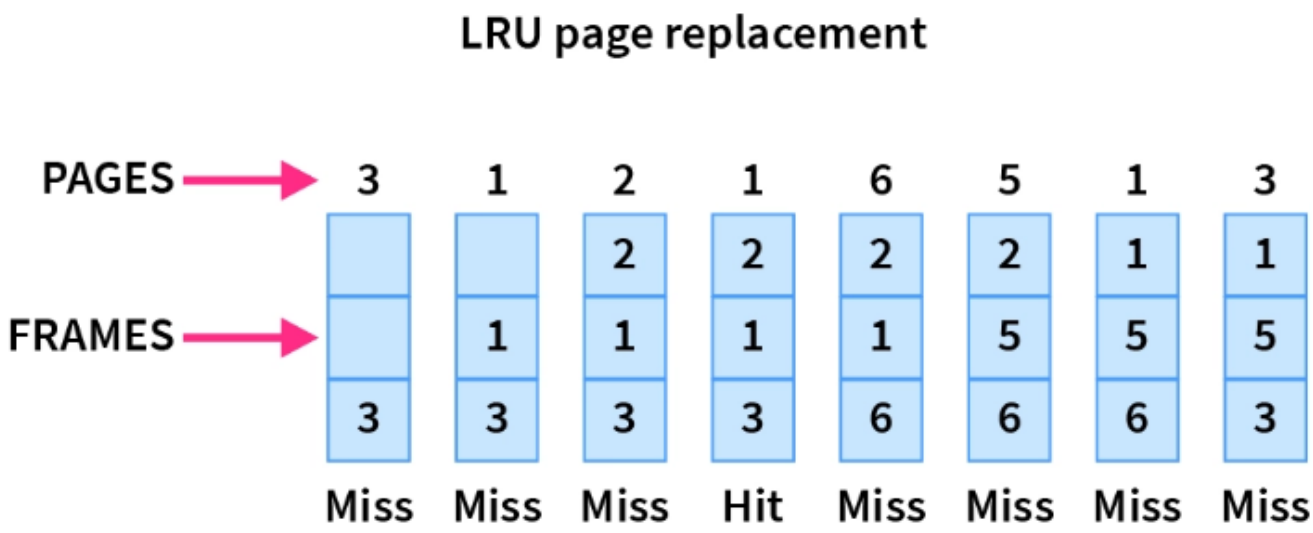
- When page 1 and page 3 come, they are in the memory so no page fault occurs.

Total page faults = 5

Least Recently Used (LRU) Page Replacement Algorithm: This algorithm works on the basis of the **principle of locality of a reference** which states that a program has a tendency to access the same set of memory locations repetitively over a short period of time. So pages that have been used heavily in the past are most likely to be used heavily in the future also.

In this algorithm, **when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page**.

Example: Let's consider the same reference string of 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames:



- Initially, since all the slots are empty, **pages 3, 1, 2** cause a page fault and take the empty slots.

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, it is not in the memory, so a page fault occurs and the least recently used page **3 is removed**.

Page faults = 4

- When page 5 comes, it again causes a page fault and page **1 is removed** as it is now the least recently used page.

Page faults = 5

- When page 1 comes again, it is not in the memory and hence page **2 is removed** according to the LRU.

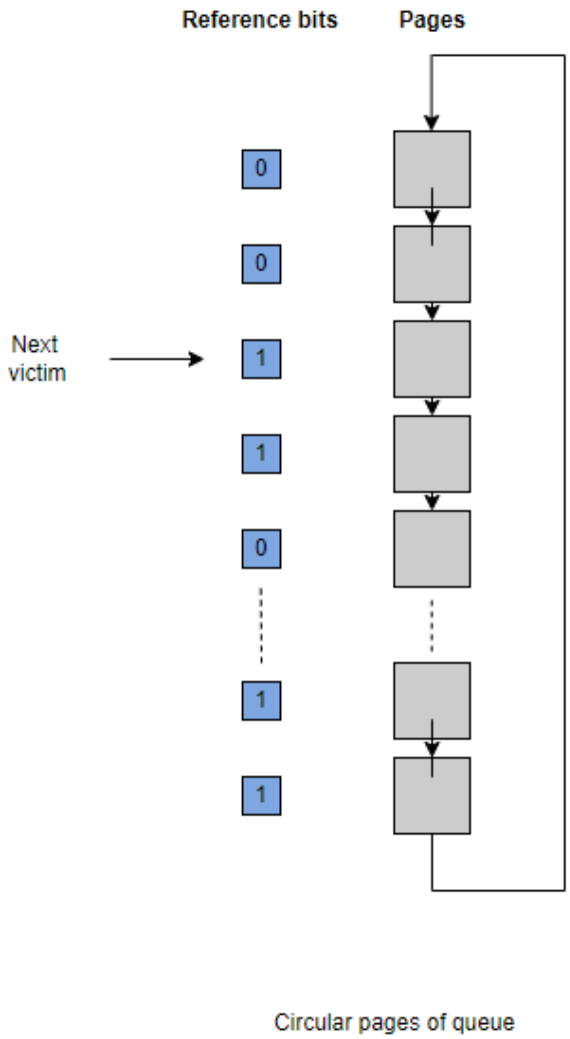
Page faults = 6

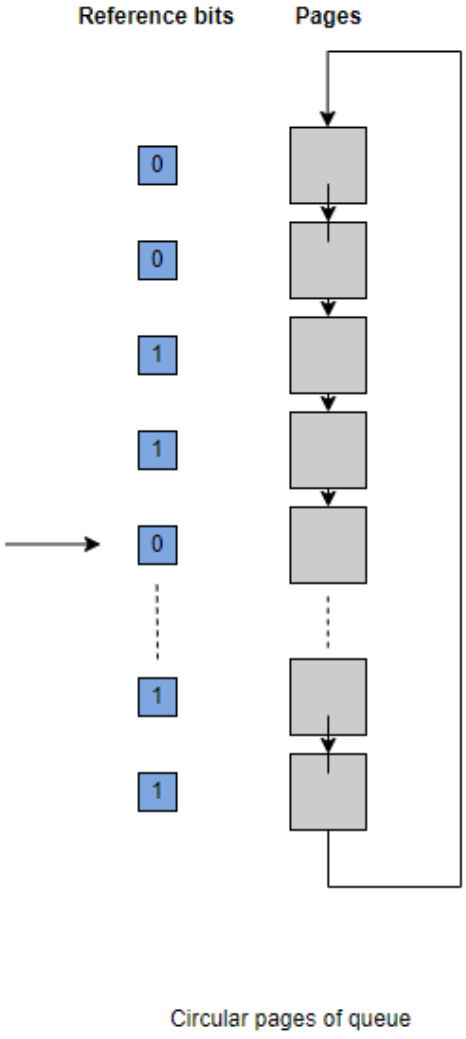
- When page 3 comes, the page fault occurs again and this time page **6 is removed** as the least recently used one.

Total page faults = 7

Second Chance LRU Page Replacement Algorithm

The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, we inspect its reference bit. If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced.





What is thrashing? Suggest its solution using working set model based approach.

Thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

Working Set Model –
This model is based on the concept of the Locality Model. The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

If D is the total demand for frames and WSS_i is the working set size for process i,

$$D = \sum WSSi$$

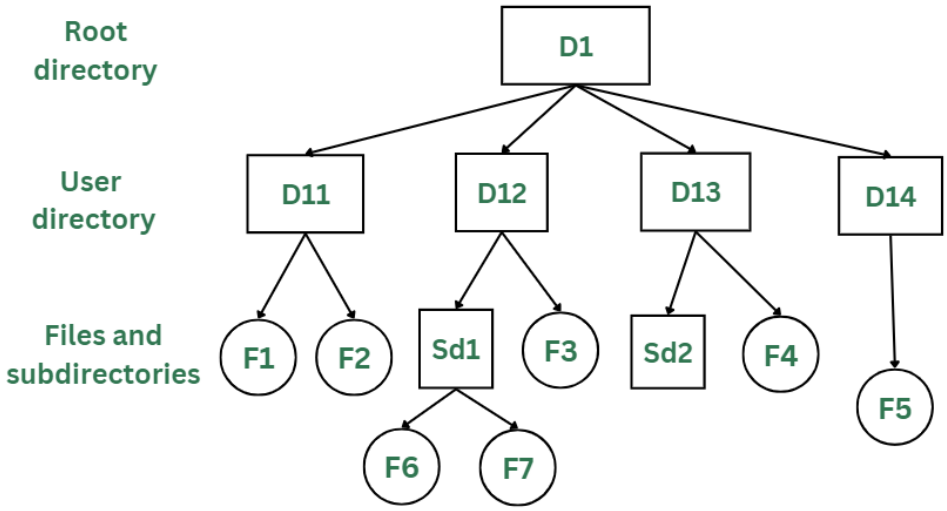
Now, if 'm' is the number of frames available in the memory, there are 2 possibilities:

- (i) D>m i.e. total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- (ii) D<=m, then there would be no thrashing.

Explain tree and graph based directory structures with suitable example. How does the performance of a directory structure is measured?

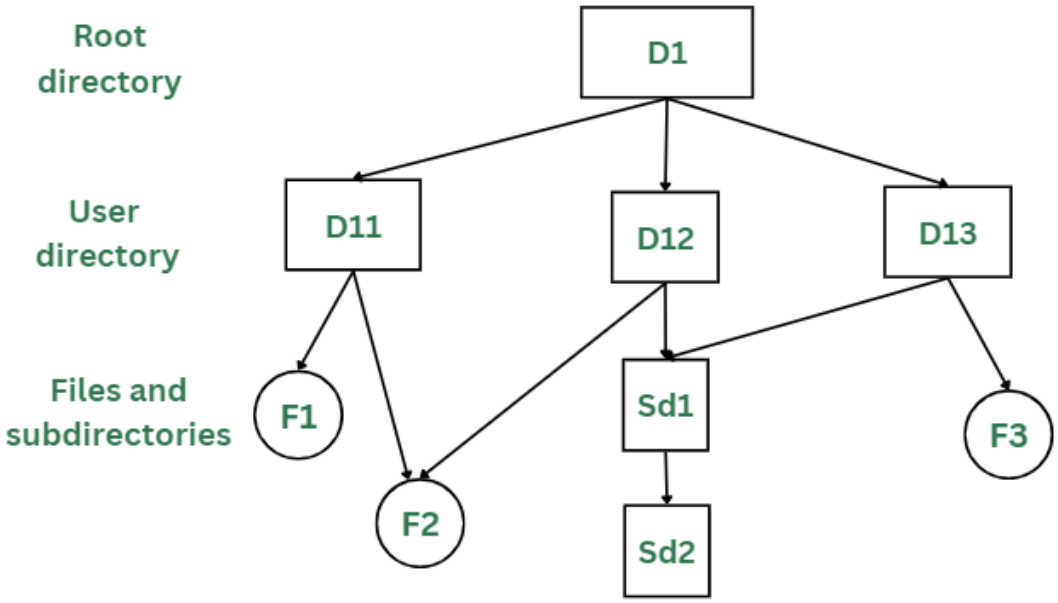
Tree-based directory structures: This directory structure resembles a real tree upside down, where the root directory is at the peak. This root contains all the directories for each user. The users can create subdirectories and even store files in their directory. A user do not have access to the root directory data and cannot modify it. And, even in this directory the user do not have access to other user's directories.

Example:



Graph-based directory structures: A file in one directory can be accessed from multiple directories. In this way, the files could be shared in between the users. It is designed in a way that multiple directories point to a particular directory or file with the help of links.

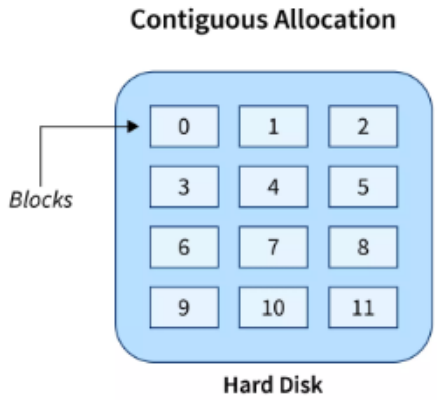
Example:



The performance of a directory structure is measured by how quickly it can be searched and how much space it takes up. Tree-based directory structures are generally faster to search than graph-based directory structures, but they can take up more space. Graph-based directory structures are generally slower to search than tree-based directory structures, but they can take up less space.

Explain contiguous and indexed file allocation methods. (b) What are SSTF and SCAN disk scheduling algorithms. Explain with suitable examples.

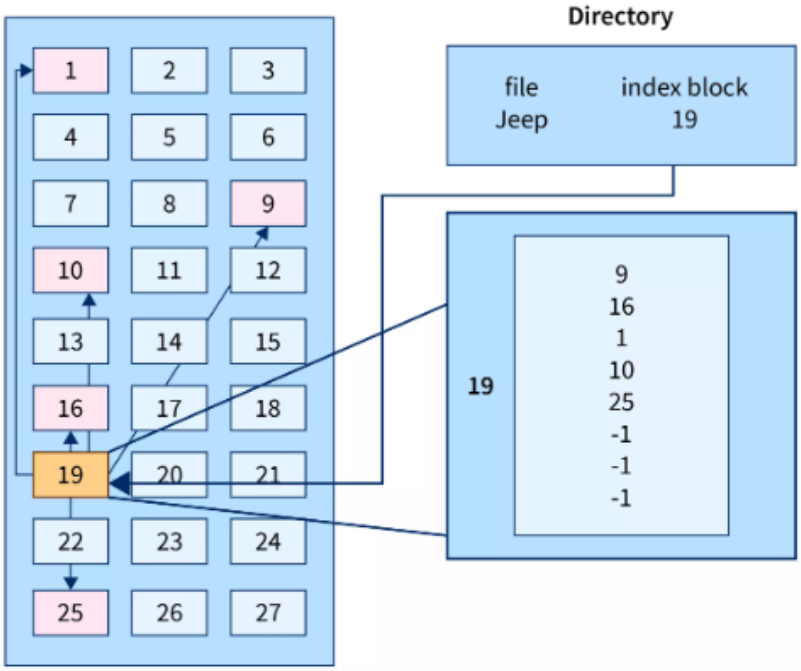
Contiguous file allocation: In contiguous file allocation, the block is allocated in such a manner that all the allocated blocks in the hard disk are adjacent.



File Name	Start	Length	Allocated Blocks
file1.txt	0	4	0,1,2,3
sun.jpg	5	3	5,6,7
mov.mp4	9	3	9,10,11

Directory

Indexed file allocation: In indexed file allocation, all the pointers are put together into one location which is called **index block**.



Shortest-Seek-Time-First (SSTF)

1. Arrange all the I/O requests in ascending order.
2. The head will find the nearest request (which has a minimum distance from the head) present in any direction (left or right) and will move to that request. Total head movement is calculated as
3. Current request - previous request (if the current request is greater)
4. Previous request - current request (if the previous request is greater)

5. Then the head will move another nearest request which has not been serviced present in any direction.
6. This process is repeated until all the requests are served and we get total head movement.

Example: Consider a disc queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The read-write head is initially at cylinder number 53. We will now use the SSTF algorithm to serve these I/O requests.

Input: I/O requests - { 98, 183, 37, 122, 14, 124, 65, 67 }
Initial head position - 53

Output: The following chart shows the sequence in which requests are served using the SSTF algorithm.

Total seek time = (65 - 53) + (67 - 65) + (67 - 37) + (37 - 14) + (98 - 14) + (122 - 98) + (124 - 122) + (183 - 124)
= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59
= 236

SCAN

1. Arrange all the I/O requests in ascending order.
2. The head will start moving in the right direction, i.e. from 0 to the size of the block.
3. As soon as a request is encountered, head movement is calculated as the current request - the previous request.
4. This process is repeated until the head reaches the end of the disc and the head movements are added.
5. As the end is reached, the direction of the head is reversed, and now the head will process the requests in the opposite direction, i.e. in the left direction.
6. This process is completed as soon as all the requests are processed and we get total head movement.

Example: Consider a disc queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The read-write head is initially at cylinder number 53. We will now use the SCAN algorithm to serve these I/O requests.

Input: I/O requests - { 98, 183, 37, 122, 14, 124, 65, 67 }
Initial head position - 53
Direction - towards the larger number of cylinders

Output: The following chart shows the sequence in which requests are served using the SCAN algorithm.

Total seek time = (65 - 53) + (67 - 65) + (98 - 67) + (122 - 98) + (124 - 122) + (183 - 124) + (199 - 183) + (199 - 37) + (37 - 14)
= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 162 + 23
= 331