EXPERIMENT NO. 8

AIM: Write a program to demonstrate Disk Scheduling Algorithms i.e FCFS, SCAN, C-SCAN.

RESOURCES REQUIRED:

H/W Requirements: P-IV and above, Ram 128 MB, Printer, Internet Connection.

S/W Requirements: Python Compiler.

THEORY:

DISK SCHEDULING:

One of the responsibilities of the operating system is the use the hardware efficiently. For the disk drivers, meeting this responsibility entails having fast access time and large disk bandwidth.

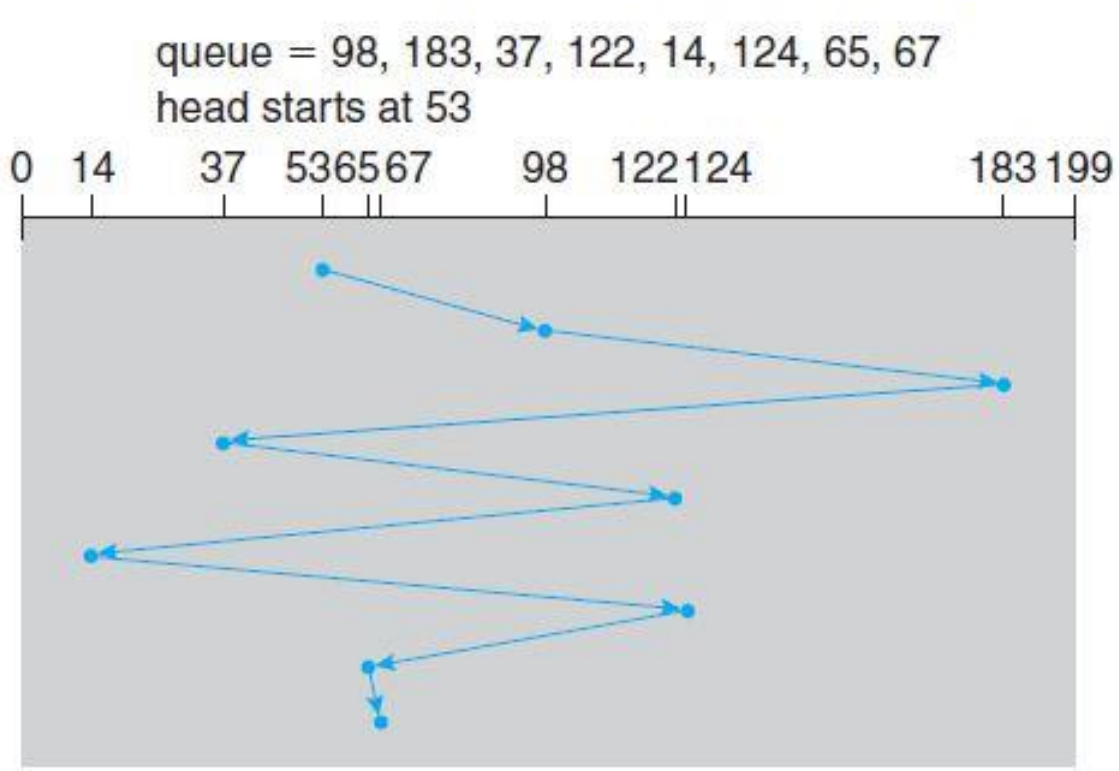
For magnetic disks, the access time has two major components: the time necessary to move the disk arm to the desired cylinder, called the **seek time**, and the time necessary for the desired sector to rotate to the disk head, called the **rotational latency**.

The disk **bandwidth** is the total number of the bytes transferred, divided by the total time between the first request for the service and the completion of the last transfer. We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

1.FCFS Scheduling:

The simplest form of disk scheduling is, of course, the first-come, first-serve (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service.

Consider, for example, a disk queue with requests for I/O to blocks on cylinders :- 98, 183, 37, 122, 14, 124, 65, 67 in the order. If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for total head movements of 640 cylinders.



2.SCAN:

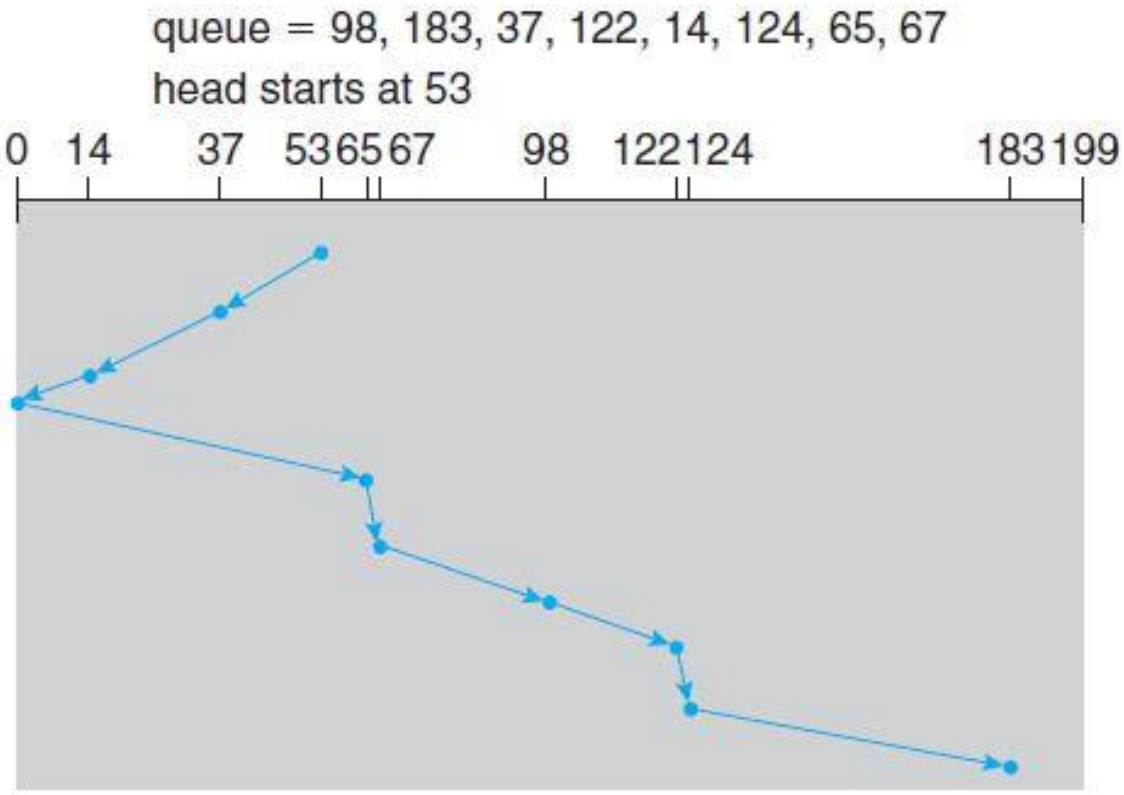
In the **SCAN algorithm**, the disk arm starts at one end of the disk and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues.

The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the **elevator algorithm**, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

Let's return to out example to illustrate. Before applying SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement in addition to the head's current position.

Assuming that the disk arm is moving toward 0 and that the initial head position is again 53, the head will next service 37 and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183. If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just

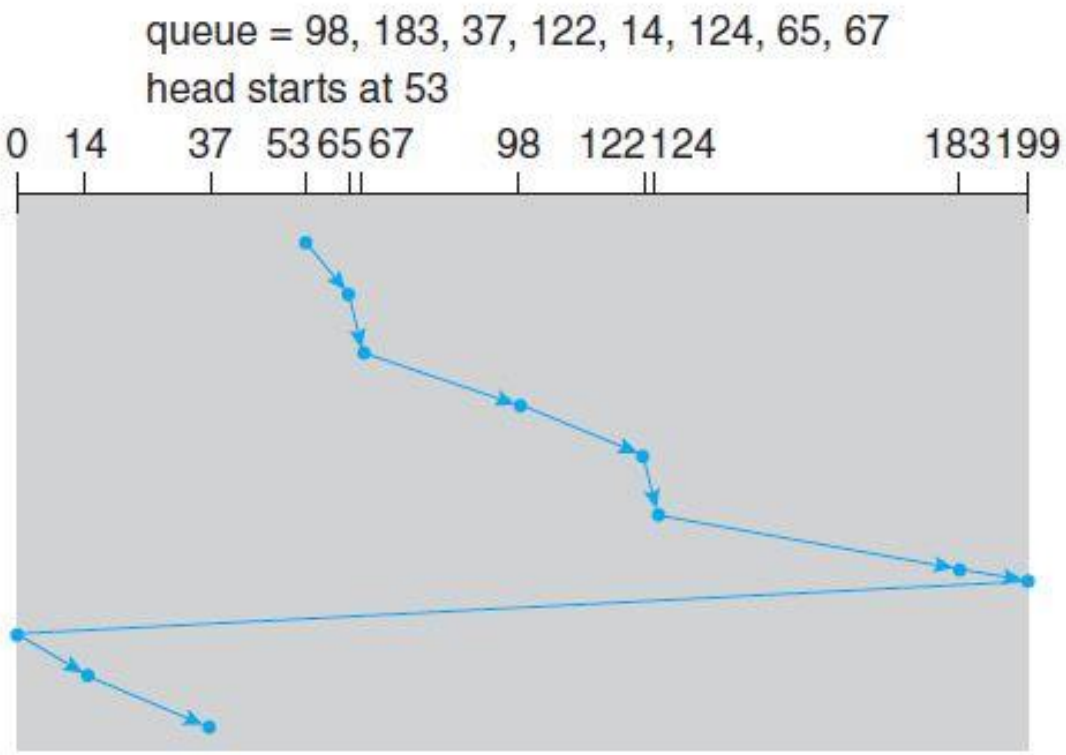
behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.



3.C-SCAN:

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way.

When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip. The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.



CONCLUSION: Hence, we have implemented the programs of Disk Scheduling Algorithms using FCFS, SCAN, C-SCAN.

CODE:

1. FCFS:

```
def fcfs():
    thm = 0
    req = int(input("Enter the Number of Requests : "))
    hp = int(input("Enter the Initial Head Position : "))
    srate = int(input("Enter the Seek Rate : "))
    print("Enter the Requests : ")
    arr = [ int(input()) for i in range(req)]
    thm = thm + abs(hp - arr[0])
    print(str(hp)+" -> "+str(arr[0]), end='')
```

```
    for i in range(1, req):
        thm = thm + abs(arr[i] - arr[i-1])
        print(" -> "+str(arr[i]), end='')
    stime = srate * thm
    print("\nThe Total Head Movement is",thm)
    print("The Seek Time is",stime)

if __name__ == "__main__":
    print("55_Adnan Shaikh")
    fcfs()
```

2. SCAN:

```
def scan():
    thm = 0
    req = int(input("Enter the Number of Requests : "))
    hp = int(input("Enter the Initial Head Position : "))
    pos = hp
    srate = int(input("Enter the Seek Rate : "))
    print("Enter the Requests : ")
    arr = [ int(input()) for i in range(req)]
    start = 0
    end = int(input("Enter ending position of disk: "))
    print(hp, end='')
    if(hp<100):
        for i in range(pos, start-1, -1):
            if i in arr:
                thm+= abs(pos-i)
```

```
            pos = i
            print(" -> ",i, end='')
            arr.remove(i)
    thm+= abs(pos-start)
    pos = start
    print(" -> ", start, end='')
    for i in range(pos, end+1):
        if i in arr:
            thm+= abs(pos-i)
            pos = i
            print(" -> ", i, end='')
            arr.remove(i)
else:
    for i in range(pos, end+1):
        if i in arr:
            thm+= abs(pos-i)
            pos = i
            print(" -> ",i, end='')
            arr.remove(i)
    thm+= abs(pos-end)
    pos = end
    print(" -> ", end, end='')
    for i in range(pos, start-1,-1):
        if i in arr:
            thm+= abs(pos-i)
            pos = i
            print(" -> ", i, end='')
```

```
            arr.remove(i)
    stime = thm * srate
    print("\nThe Total Head Movement is",thm)
    print("The Seek Time is",stime)

if __name__ == "__main__":
    print("55_Adnan Shaikh")
    scan()
```

3. C-SCAN:

```
def CSCAN(arr, head,disk_size,size):
        seek_count = 0
        distance = 0
        cur_track = 0
        left = []
        right = []
        seek_sequence = []
        left.append(0)
        right.append(disk_size - 1)
        for i in range(size):
            if (arr[i] < head):
                left.append(arr[i])
            if (arr[i] > head):
                right.append(arr[i])
        left.sort()
        right.sort()
```

```
        for i in range(len(right)):
                cur_track = right[i]
                seek_sequence.append(cur_track)
                distance = abs(cur_track - head)
                seek_count += distance
                head = cur_track
        head = 0
        seek_count += (disk_size - 1)

        for i in range(len(left)):
                cur_track = left[i]
                seek_sequence.append(cur_track)
                distance = abs(cur_track - head)
                seek_count += distance
                head = cur_track
        print("Total number of seek operations =",
                seek_count)
        print("Seek Sequence is")
        print(*seek_sequence, sep="\n")

if __name__ == "__main__":
    print("55_Adnan Shaikh")
    arr = list(map(int,input("Enter requests: ").strip().split()))
    head = int(input("Enter head position: "))
    disk_size = int(input("Enter ending position of disk: ")) + 1
    print("Initial position of head:", head)
    CSCAN(arr, head, disk_size,len(arr))
```

OUTPUT:

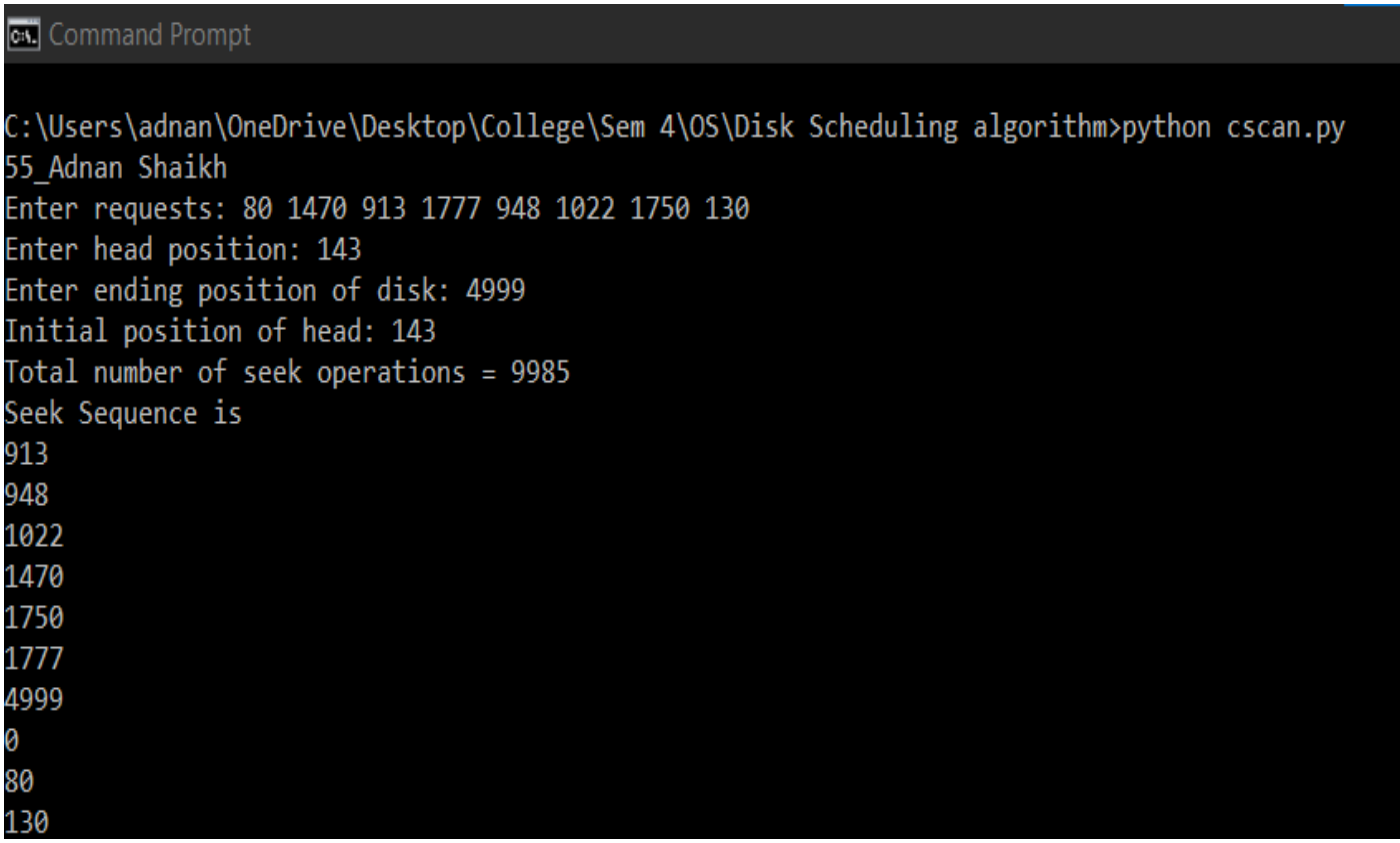
1. FCFS:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Disk Scheduling algorithm>python fcfs.py
55_Adnan Shaikh
Enter the Number of Requests : 8
Enter the Initial Head Position : 143
Enter the Seek Rate : 1
Enter the Requests :
80
1470
913
1777
948
1022
1750
130
143 -> 80 -> 1470 -> 913 -> 1777 -> 948 -> 1022 -> 1750 -> 130
The Total Head Movement is 6125
The Seek Time is 6125
```

2. SCAN:

```
Command Prompt

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Disk Scheduling algorithm>python scan.py
55_Adnan Shaikh
Enter the Number of Requests : 8
Enter the Initial Head Position : 143
Enter the Seek Rate : 1
Enter the Requests :
80
1470
913
1777
948
1022
1750
130
Enter ending position of disk: 4999
143 ->  913 ->  948 ->  1022 ->  1470 ->  1750 ->  1777 ->  4999 ->  130 ->  80
The Total Head Movement is 9775
The Seek Time is 9775
```

3. C-SCAN:


Command Prompt
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\OS\Disk Scheduling algorithm>python cscan.py
55_Adnan Shaikh
Enter requests: 80 1470 913 1777 948 1022 1750 130
Enter head position: 143
Enter ending position of disk: 4999
Initial position of head: 143
Total number of seek operations = 9985
Seek Sequence is
913
948
1022
1470
1750
1777
4999
0
80
130